BAB II

TELAAH LITERATUR

2.1 Rekayasa Perangkat Lunak

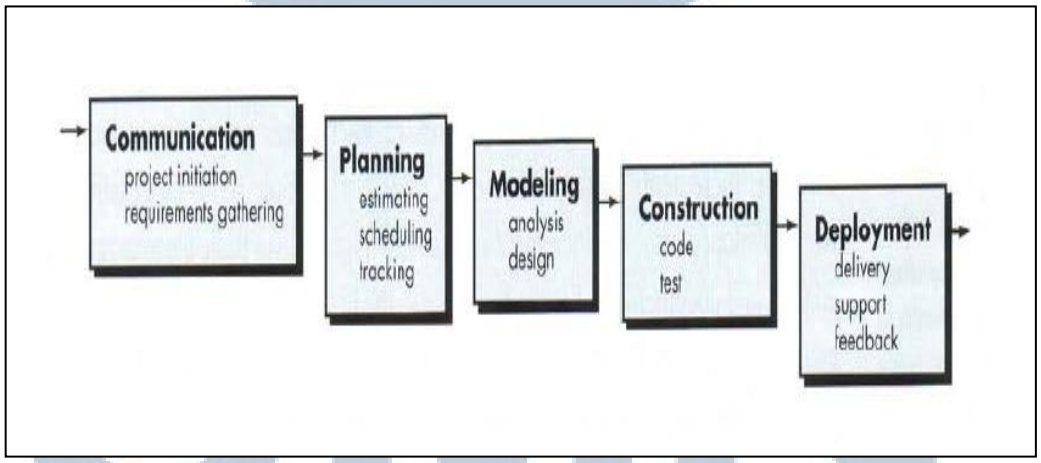
Dalam membangun sebuah perangkat lunak (*software*) diperlukan serangkaian proses, metode dan berbagai hal lain atau yang lebih dikenal dengan rekayasa perangkat lunak. Pengertian dari rekayasa perangkat lunak itu sendiri diungkapkan oleh IEEE secara lebih komprehensif dalam *Standard Collections*-nya (1993) bahwa definisi rekayasa perangkat lunak ada dua definisi. Definisi pertama menjelaskan bahwa rekayasa perangkat lunak merupakan aplikasi dari sebuah pendekatan kuantifiabel, disiplin, dan sistematis kepada pengembangan, operasi, dan pemeliharaan perangkat lunak. Definisi kedua menjelaskan bahwa rekayasa perangkat lunak merupakan studi tentang pendekatan-pendekatan kuantifiabel, disiplin, dan sistematis.

Dengan adanya rekayasa perangkat lunak diharapkan dapat diperoleh biaya produksi perangkat lunak yang rendah; menghasilkan perangkat lunak yang kinerjanya tinggi, andal, dan tepat waktu; serta menghasilkan perangkat lunak yang biaya perawatannya rendah.

Selanjutnya, Pressman (2005, p.73-74) mengungkapkan untuk menyelesaikan suatu masalah pembangunan perangkat lunak, pelaku harus menggabungkan strategi pengembangan yang melingkupi lapisan proses, metode, dan alat-alat bantu. Strategi ini sering diacukan sebagai model proses atau paradigma rekayasa perangkat lunak. Model proses yang dipilih didasarkan pada

sifat aplikasi dan proyeknya, metode dan alat-alat bantu yang dipakai, serta kontrol dan penyampaian yang dibutuhkan. Hal ini penting dilakukan untuk men-*generate* setiap aktivitas kerangka kerja dan mendefinisikan setiap tindakan.

Salah satu metode yang dapat digunakan adalah metode air terjun atau yang lebih dikenal dengan sebutan *waterfall model*. *Waterfall model* terkadang disebut sebagai siklus hidup klasik (*the classic lifecycle*). Model ini menyarankan pendekatan sistematis, berurutan untuk pengembangan perangkat lunak yang dimulai dari spesifikasi pelanggan. *Waterfall model* berkembang melalui inisiasi proyek (*communication*), perencanaan (*planning*), permodelan (*modeling*), konstruksi (*constuction*), dan penyebaran (*deployment*) (Pressman, 2005, p. 79-80).



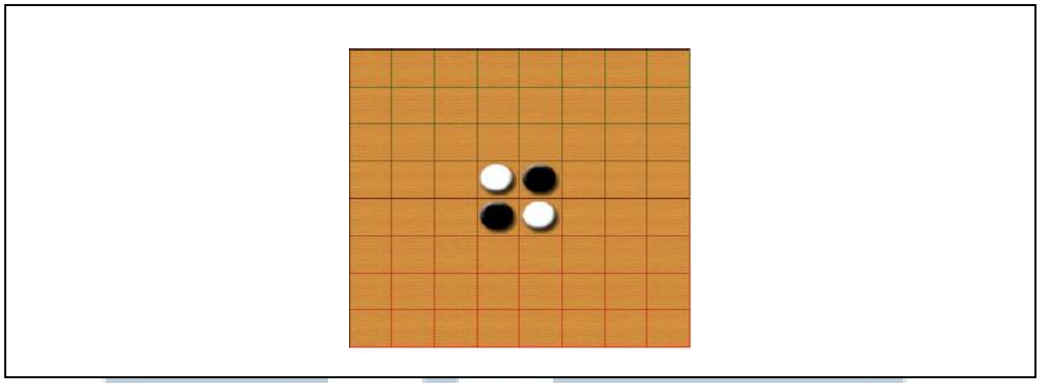
Gambar 2.1 Model Air Terjun (*Waterfall Model*)
Rekayasa Perangkat Lunak (Sumber : Pressman, 2005, p. 79)

Melalui model ini, dilakukan inisiasi pengembangan proyek yang kemudian dilanjutkan dengan perencanaan. Kemudian setelah merencanakan proyek dilakukan analisis dan desain permodelan. Setelah selesai, dilakukan *coding* dan *testing,* hingga akhirnya dilakukan komparasi algoritma berdasarkan hasil uji terhadap proyek yang telah diimplementasikan.

2.2 Permainan Othello

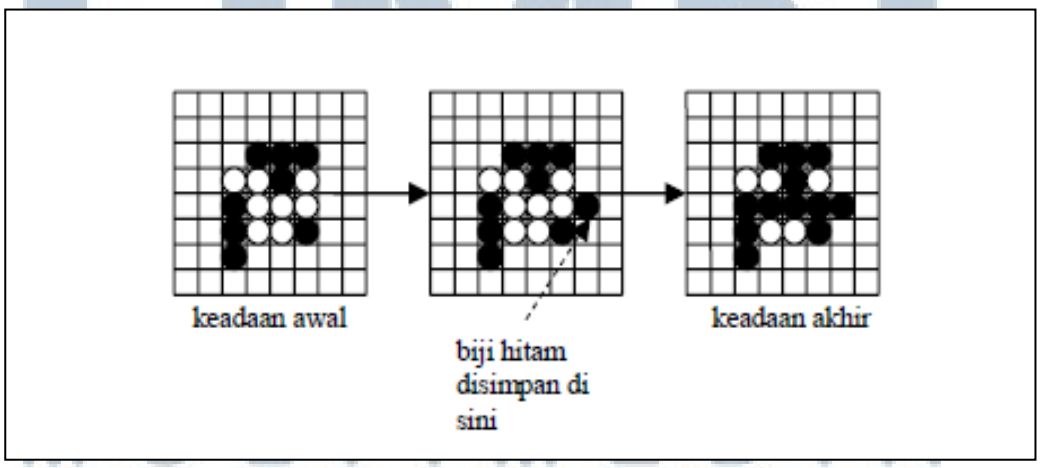
Rachmah (2008, p.1) mengungkapkan dalam *paper*-nya yang berjudul "Implementasi Algoritma *Greedy* Pada Permainan *Othello*" bahwa permainan *othello* adalah permainan yang dilakukan diatas sebuah papan berukuran 8x8 yang biasanya dimainkan oleh dua orang pemain. Prabawa (2009, p.1) menyebutkan bahwa permainan *othello* ditemukan pada tahun 1883 oleh dua orang berkebangsaan Inggris, yaitu Lewis Waterman dan John Mollet. Disebutkan pula bahwa permainan *othello* menjadi populer baik karena kesederhanaannya maupun karena kompleksitasnya. Bahkan Russel dan Norvig (2009, p.186) mengungkapkan bahwa kini permainan *othello* mungkin tidak lagi dikenal sebagai permainan papan, melainkan lebih populer sebagai permainan komputer.

Permainan *othello* biasanya diawali dengan dua bidak putih milik seorang pemain dan dua bidak hitam milik lawan yang diletakkan bergantian secara diagonal seperti terlihat pada gambar 2.2.



Gambar 2.2 Posisi Awal Permainan *Othello* (Sumber : Prabawa, 2009, p.2)

Pada permainan *othello*, pemain berusaha memenuhi papan yang ada, baik kolom maupun baris dengan bidak masing-masing. Pemain berusaha mengalahkan lawan dengan sebanyak mungkin memiliki bidak dibanding bidak milik lawan di papan permainan tersebut. Masing-masing pemain akan meletakkan bidak berseberangan dengan bidak miliknya untuk mengubah warna bidak musuh. Sebagai contoh apabila pemain yang memiliki bidak hitam meletakkan bidak berseberangan dengan bidak hitam yang sudah dimilikinya di atas papan, maka bidak putih yang diapit oleh bidak hitam tersebut akan berubah warnanya menjadi bidak hitam seperti terlihat pada ilustrasi gambar 2.3.



Gambar 2.3 Ilustrasi Peletakkan Bidak Hitam
(Sumber : Putra dkk, 2006, p.1)

Proses peletakkan masing-masing bidak diletakkan secara bergantian antara bidak putih milik pemain dengan bidak hitam milik lawan. Permainan *othello* terus berlanjut hingga seluruh papan terisi penuh dengan oleh masing-masing bidak, baik bidak hitam maupun bidak putih. Pada akhir permainan, pemain yang memiliki bidak yang lebih banyak jumlahnya yang akan menjadi pemenang dalam permainan *othello* ini.

2.3 Android dan Bahasa Pemrogramannya

Pada tahun 2005 Google mengakuisisi Android *Inc.* Kemudian pada tanggal 12 November 2007 Google bersama *Open Handset Alliance* (OHA) yaitu konsorsium perangkat *mobile* terbuka, merilis Google Android SDK setelah mengumumkannya seminggu sebelumnya (Mulyadi, 2010, p.2-3).

Platform android bersifat *open source.* Menurut www.organisasi.org *open source software* adalah istilah yang digunakan untuk *software* yang membuka/membebaskan *source code*-nya untuk dilihat oleh orang lain dan membiarkan orang lain mengetahui cara kerja *software* tersebut dan sekaligus memperbaiki kelemahan-kelemahan yang ada pada *software* tersebut. Dengan demikian "*programmer* atau *developer* secara penuh akan bisa mengkustomisasi perangkat androidnya" (Mulyadi, 2010, p.3).

Saat ini android sudah memiliki beberapa versi, beberapa versi diantaranya yang terkenal adalah

1. Android versi 1.5 atau yang biasa disebut versi *Cupcake.*
2. Android versi 1.6 atau yang biasa disebut versi *Donut.*

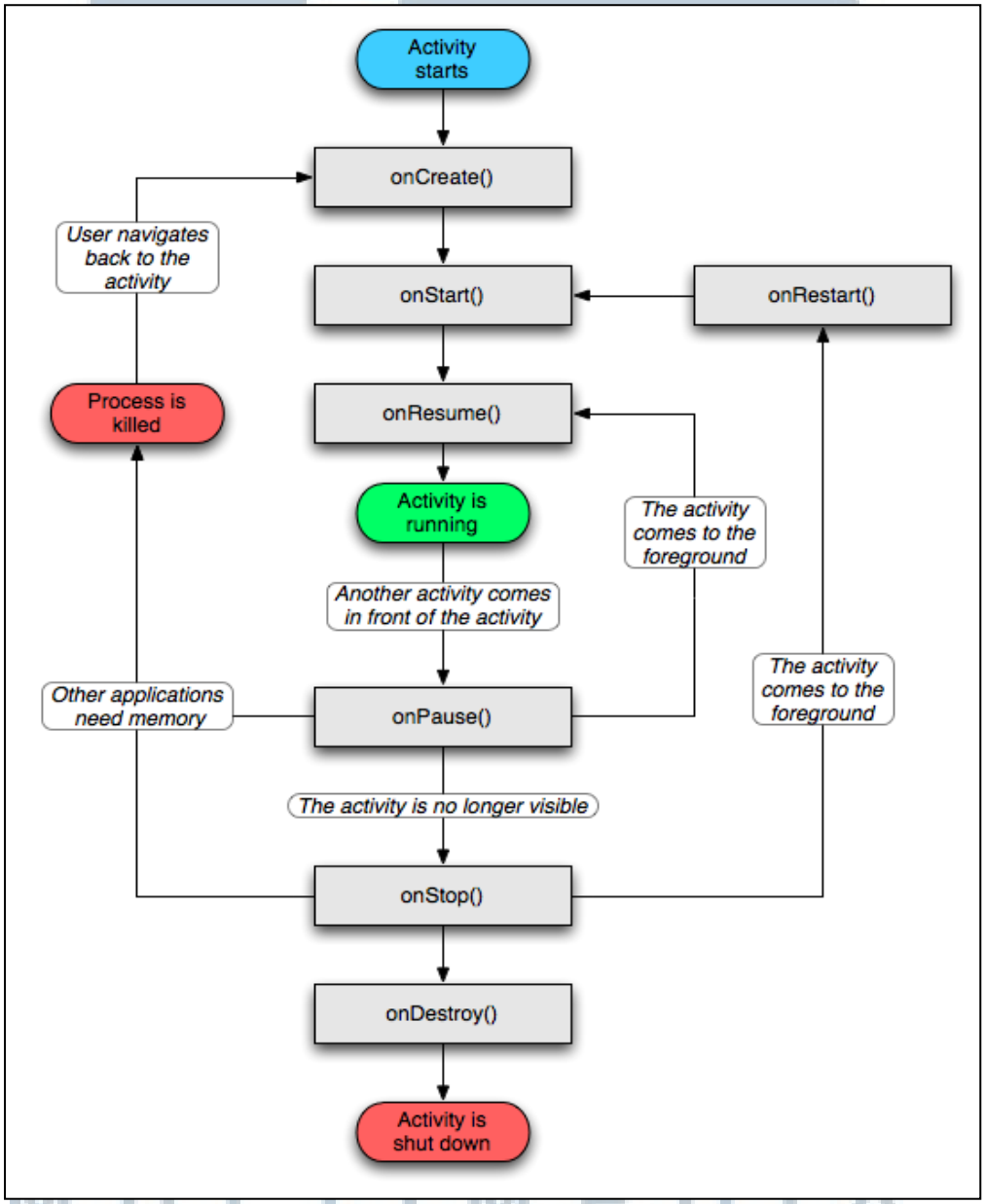
3. Android versi 2.0 / 2.1 atau yang biasa disebut versi *Eclair*.
4. Android versi 2.2 atau yang biasa disebut versi *Froyo / Frozen Yoghurt.*
5. Android versi 2.3 atau yang biasa disebut versi *Gingerbread.*
6. Android versi 3.0/3.1 atau yang biasa disebut versi *Honeycomb.*

Murphy (2010 p.2) mengungkapkan bahwa android menggunakan bahasa pemrograman yang umum yaitu bahasa pemrograman Java dengan beberapa *libraries* yang umum digunakan (seperti *Apache Common APIs*) dan *support tools* yang dapat digunakan yaitu Eclipse.

Murphy (2010, p.2-3) juga mengungkapkan bahwa ada 4 komponen utama yang digunakan dalam aplikasi android, yaitu.

1. *Activities*: yaitu menyajikan *user interface* bagi para penggunanya. Pada gambar 2.4 dapat dilihat *loop* diagram *activity* dengan *state.*
2. *Content providers*: yaitu menyediakan sebuah level abstraksi untuk digunakan dalam berbagai penyimpanan data dan diakses oleh berbagai aplikasi. Model pengembangan android mendorong setiap pengembangnya untuk membuat data sendiri yang tersedia untuk aplikasi lain dan tentunya pengembang itu sendiri.
3. *Services*: *services* tidak memiliki tampilan visual, melainkan berjalan di latar belakang. *Services* digunakan untuk dijalankan bila diperlukan secara independen oleh suatu *activity*, sehingga baik *activities* maupun penyedia *content* dapat ditutup setiap saat bila *services* diperlukan. Intinya *services* berjalan di latar belakang untuk waktu yang tak terbatas.

4. *Intent*: yaitu sebuah pesan sistem, berjalan di sekitar bagian dalam perangkat *device,* memberitahukan notifikasi pada aplikasi dengan berbagai cara. Sebagai contoh apabila terdapat pesan singkat atau SMS yang masuk.



Gambar 2.4 *Loop* Diagram *Activity* Dengan *State*
(Sumber : http://developer.android.com)

"Sebelum android dapat menjalankan sebuah komponen aplikasi, ia harus mengetahui bahwa komponen tersebut ada" (Mulyadi, 2010, p.17). Maka selain bahasa pemrograman Java, terdapat pula struktur *file* XML yang digunakan oleh aplikasi untuk mendeklarasikan komponennya dalam *file manifest.* Dimana "tugas pokok *file manifest* adalah untuk menginformasikan kepada android tentang komponen yang ada pada aplikasi android" (Mulyadi, 2010, p.17).

2.4 Bahasa Pemrograman Java

Bahasa pemrograman Java adalah bahasa pemrograman modern yang dapat digunakan pada sejumlah sistem operasi yang berbeda. (Setiadi, 2008, p.179) Menurut Gosling, bahasa pemrograman Java dibuat untuk tujuan yang umum, yaitu sebagai bahasa pemrograman yang berorientasi objek dan berbasis kelas (2005, p.1).

Bahasa pemrograman Java memiliki keterkaitan dengan bahasa pemrograman C dan C++ tetapi diatur agak berbeda, karena ada sejumlah aspek C dan C++ yang dihilangkan dan beberapa ide dari bahasa pemrograman lain yang digabungkan. Bahasa pemrograman Java merupakan bahasa pemrograman yang relatif tingkat tinggi, yang biasanya di kompilasi ke kumpulan instruksi *bytecode* dan format biner yang didefinisikan dalam spesifikasi mesin *Java virtual* (Gosling, 2005, p.1).

Java berperan sebagai *compiler* dan juga sebagai *interpreter*. Konsepnya diawali dengan kode program yang ditulis dengan bahasa pemrograman Java (berekstensi *.java*) yang dikompilasi (oleh *compiler*) menjadi suatu kode objek.

Dimana dalam terminologi Java, kode objek ini disebut dengan istilah *bytecode* (berekstensi *.class*), bukan *file .exe*. Kemudian *bytecode* akan dieksekusi baris demi baris (oleh *interpreter*). Dengan demikian proses kompilasi hanya dilakukan sekali, akan tetapi proses *interpreter* akan dilakukan setiap program di eksekusi (Raharjo, 2007, p.2).

Kumpulan instruksi *bytecode* yang dapat dianggap sebagai sekumpulan perintah dalam bahasa mesin untuk sebuah mesin *virtual Java*, menandakan bahwa program yang dibuat dengan Java tidak mungkin dapat dijalankan di dalam komputer maupun alat lain yang tidak memiliki *Java Virtual Machine* (JVM) (Raharjo, 2007, p.3).

Dengan adanya konsep *bytecode*, dalam terminologi Java dikenal sebuah istilah "*write once, run anywhere*". Ini berarti bahwa sekali program Java ditulis dan dikompilasi, maka *bytecode*-nya dapat dijalankan di dalam *platform* manapun selama *platform* tersebut memiliki mesin virtual Java atau *Java Virtual Machine* (JVM) (Raharjo, 2007, p.3).

2.5 Struktur File Extensible Markup Language (XML)

Menurut www.w3.org, *extensible markup language* (XML) mendeskripsikan sebuah kelas objek data yang disebut dokumen XML dan sebagian lain menggambarkan perilaku program komputer yang memproses XML. XML adalah suatu profil aplikasi atau bentuk yang dibatasi dari SGML (*Standard Generalized Markup Language*) – ISO 8879, dimana konstruksi dokumen XML sesuai dengan dokumen SGML.

Dokumen XML terdiri dari unit penyimpanan yang disebut entitas, yang berisi data yang diurai maupun tidak. Data yang diuraikan terdiri dari karakter, beberapa *form* yang membentuk data karakter, dan beberapa *form markup*. XML juga menyediakan mekanisme untuk memaksakan kendala pada tata letak dan struktur logis.

Pada android, struktur *file manifest* yang digunakan untuk menginformasikan kepada android tentang komponen yang ada pada aplikasi android merupakan struktur *file* XML dan selalu bernama AndroidManifest.xml untuk semua aplikasi. (Mulyadi, 2010, p.17) Contoh deklarasi XML pada android untuk menyatakan *activity* dapat dinyatakan sebagai berikut :

```
<? xml version = "1.0" encoding = "UTF-8" ?>
<?xml version = "1.0" encoding = "utf-8" ?>
<manifest ... >
      <application ...>
            <activity
android:name="com.example.project.FreneticActivity"
                  android:icon="@drawable/small_pic.png"
                  android:label="@string/freneticLabel"
                  ...>
            </activity>
            ...
      </application>
</manifest>
```

2.6 Artificial Intelligent

Russell dan Norvig (2009, p.1-2) mengelompokkan pengertian dari *artificial intelligent* ke dalam 4 dimensi, yaitu.

1. Sistem yang berpikir seperti manusia.
2. Sistem yang berpikir secara rasional.

3. Sistem yang berperilaku seperti manusia.
4. Sistem yang berperilaku secara rasional.

Pekerjaan yang sekarang diakui sebagai *artificial intelligent* dilakukan pertama kali oleh Warren McCulloch dan Walter Pitts pada tahun 1943. Mereka menggunakan tiga sumber. yaitu pengetahuan tentang fisiologi dasar, fungsi neuron otak, dan teori perhitungan komputasi Turing. Mereka mengusulkan sebuah model neuron buatan dimana setiap neuron dicirikan sebagai tombol "*on*" dan "off". Dari ide inilah *artificial intelligent* mulai dikembangkan.

Tahun 1956 disebutkan oleh Russel dan Norvig (2009, p.17-18) sebagai tahun kelahiran *artificial intelligent.* John McCarthy. salah satu tokoh yang berpengaruh dalam *artificial intelligent*, bertemu dengan para peneliti dari Amerika Serikat pada lokakarya di tahun 1956. Hingga akhirnya berdasarkan lokakarya tersebut, dapat dilihat bahwa *artificial intelligent* diadopsi sebagai suatu *field* yang terpisah.

Di tahun 1952 hingga 1969 *artificial intelligent* memperoleh antusias yang tinggi dan memperoleh ekspektasi yang sangat luar biasa. Hingga akhirnya *artificial intelligent* mulai digunakan dalam dunia industri pada tahun 1980 hingga sekarang. Dimana saat itu *expert system* komersial yang pertama kali sukses dimulai di *Digital Equipment Corporation* pada tahun 1982. Program tersebut sukses membantu sistem pemesanan komputer baru (Russel dan Norvig, 2009, p. 18-24).

Artificial intelligent mengadopsi metode ilmiah sejak tahun 1987 hingga sekarang. (Russel dan Norvig, 2009, p. 25). Kini *artificial intelligent* sudah banyak digunakan untuk menyelesaikan berbagai permasalahan seperti digunakan dalam perencanaan dan penjadwalan, permainan komputer, otonomi kontrol, diagnosa, perencanaan logistik, robotik, hingga pemahaman bahasa dan pemecahan masalah (Russel dan Norvig, 2009, p. 28-29).

Di dalam permainan komputer, sudah banyak *artificial intelligent* yang diciptakan dan terbukti dapat mengalahkan juara-juara dunia dalam permainan tersebut. Salah satunya adalah pada tahun 1997, "logistello", salah satu program *artificial intelligent*, mampu mengalahkan juara dunia Takeshi Murakami dalam permainan *othello* (Russel dan Norvig, 2009, p.186). Karena tidak adanya penjelasan mengenai algoritma apakah yang digunakan untuk menciptakan *artificial intelligent* bernama "logistello", maka dilakukan studi komparasi terhadap algoritma *greedy* dan algoritma *minimax* untuk mengetahui algoritma manakah yang lebih baik ketika diterapkan sebagai *artificial intelligent* untuk melawan manusia.

2.7 Algoritma Greedy

Putra, dkk (2006, p.2) dalam *paper*-nya yang berjudul "Penerapan Algoritma *Greedy* Pada Permainan Othello" mengemukakan bahwa algoritma *greedy* adalah algoritma sederhana dimana penyelesaian dilakukan dengan cara *straight forward* dan membentuk solusi langkah per langkah (*step by step*).

Rachmah (2009, p.2) juga mengungkapkan bahwa algoritma *greedy* merupakan metode yang paling populer untuk memecahkan masalah optimasi.

Algoritma ini biasanya digunakan untuk perolehan nilai optimum dari suatu kasus. Pendekatan yang digunakan di dalam algoritma *greedy* adalah membuat pilihan yang tampak memberi perolehan terbaik, yaitu dengan membuat pilihan optimum lokal pada setiap langkah dengan harapan akan mengarah ke solusi optimum global. Prinsip yang digunakan dalam algoritma *greedy* adalah "*take what you can get now*" tanpa memperhatikan konsekuensi ke depan, dan berharap bahwa dengan memilih optimum lokal pada setiap langkah akan menghasilkan optimum global pada akhir proses.

"Metode *greedy* sangat berguna apabila pencarian solusi yang 100% sempurna memakan waktu yang terlalu lama dengan menggunakan komputer." (Setiadi, 2008, p.96) Selain itu Russel dan Norvig (2009, p.92) mengungkapkan bahwa algoritma *greedy* cenderung mengarah pada solusi cepat.

Rachmah (2009, p.2) mengungkapkan bahwa persoalan optimasi algoritma *greedy* disusun oleh elemen-elemen berikut.

1. Himpunan kandidat, yang berisi elemen-elemen pembentuk solusi.
2. Himpunan solusi, yang berisikan kandidat-kandidat yang terpilih sebagai solusi persoalan.
3. Fungsi seleksi, dinyatakan dengan predikat 'seleksi' memilih kandidat yang paling memungkinkan mencapai solusi optimal pada setiap langkah.

4. Fungsi kelayakan, dinyatakan dengan predikat ‘layak’, memeriksa apakah suatu kandidat yang telah dipilih dapat memberikan solusi yang layak dengan tidak melanggar *constraints* yang ada.

5. Fungsi objektif, yang memaksimumkan atau meminimumkan nilai solusi.

Adapun skema umum algoritma *greedy* diungkapkan oleh Rachmah dapat digambarkan melalui *pseudo-code* yang terdapat pada algoritma 2.1.

Algoritma 2.1 Algoritma Greedy

```
function greedy (input C:h_kandidat) -> h_kandidat
{
      Mengembalikan solusi dari persoalan optimasi
      dengan algoritma greedy. Masukan : himpunan
      kandidat C. Keluaran : himpunan solusi yang
      bertipe himpunan_kandidat.
}
Deklarasi
      x : kandidat
      S : h_kandidat

      function SELEKSI (C : h_kandidat) -> kandidat
      (me-return sebuah kandidat yang dipilih dari
      C berdasarkan kriteria tertentu)

      function SOLUSI (S : h_kandidat) -> boolean
      (true jika S adalah solusi dari persoalan;
      false jika S belum menjadi solusi)

      function LAYAK (S : h_kandidat) -> boolean
      (true jika S merupakan solusi yang tidak
      melanggar kendala ; false jika S melanggar
      kendala )
```

Algoritma

```
S <- {}    {Inisialisasi S dengan kosong}
while (not SOLUSI(S)) and (C ≠ {}) do
     x <- SELEKSI (C) {pilih 1 kandidat dari C
     C <- C - {x}
     if LAYAK (S U {x}) then
          S <- S U {x}
     endif
endwhile
{SOLUSI(S) or C = {}  }

if SOLUSI (S) then
     return S
else
     write ('tidak ada solusi')
endif
```

2.8 Algoritma Minimax

Prabawa (2009, p.2) mengemukakan bahwa algoritma *minimax* adalah algoritma yang "melihat beberapa langkah ke depan", dengan cara melakukan pencarian pada pohon permainan (*Game Tree*), yang menyimpan posisi-posisi papan. Algoritma *minimax* berusaha menciptakan sebuah pohon permainan dimana setiap cabangnya merepresentasikan sebuah nilai dari setiap langkah yang diambil.

Pohon permainan yang diciptakan dengan algoritma *minimax* selalu berasumsi bahwa setiap pemain akan mengambil langkah terbaik pada setiap giliran. Untuk menciptakan pohon permainan, algoritma *minimax* diimplementasikan secara rekursif.

Menurut Prabawa (2009, p. 3), membangun seluruh ruang solusi dengan pohon permainan sampai posisi akhir papan permainan tidak mungkin dilakukan dalam permainan *othello* dan kedalaman pohon permainan harus dibatasi. Jika

tidak dibatasi, maka pemrosesan waktu akan meningkat, walaupun memang semakin dalam pohon permainan akan semakin baik langkah yang diambil.

Secara garis besar, algoritma *minimax* memiliki konsep dengan berusaha meminimalkan kemungkinan kekalahan dan memaksimalkan kemungkinan kemenangan. Adapun skema umum algoritma *minimax* diungkapkan oleh Prabawa dapat digambarkan melalui *pseudo-code* yang terdapat pada algoritma 2.2.

Algoritma 2.2 Algoritma Minimax

```
function MiniMax (input M : Matrix, input
depth.Player : integer) -> integer
{
     GameOver() adalah fungsi yang memeriksa apakah kondisi
     papan permainan telah menggambarkan akhir permainan.
     Evaluation() adalah fungsi yang menghitung nilai
     sebuah simpul daun.
     CanPlay() adalah fungsi yang memeriksa apakah seorang
     pemain dapat bergerak.
     ValidMoves() adalah fungsi yang menghasilkan senarai
     langkah yang dapat dilakukan seorang pemain.
     Sort() adalah fungsi yang mengurutkan isi senarai.
     CopyMatrix() adalah fungsi yang menyalin isi Matriks.
     Move() adalah fungsi yang mengeksekusi langkah seorang
     pemain pada papan permainan.
     EmptyMove() adalah fungsi yang memeriksa apakah sebuah
     langkah null atau tidak.
}

{ KAMUS }
MaxDepth : integer
otherPlayer, currentPlayer : integer
Value, bestValue : integer
i : integer
moves : ListMove
N : Matrix
bestMove : Move
```

```
{ ALGORITMA }
 if(GameOver(M) or (depth > MaxDepth)) then
    → Evaluation(B, currentPlayer)
 if(not CanPlay(Player)) then
    → MiniMax(M, depth, otherPlayer)
 moves ← ValidMoves(Player)
 if(Player = currentPlayer) then
    bestValue ← -999999999
 else
    bestValue ← 999999999

 i traversal [1..moves.Count]
    N ← CopyMatrix(M)
    Move(N,moves[i].x,moves[i].y,Player)
    Value ← MiniMax (N, depth+1, otherPlayer)
    if (Player = currentPlayer) then
         if(Value > bestValue) then
              bestValue ← Value
              bestMove ← moves[i]
    else
         if(Value < bestValue) then
              bestValue ← Value
              bestMove ← moves[i]
 if(not EmptyMove(bestMove)) then
    Move(M, bestMove.x,bestMove.y,Player)
 → bestValue
```