BAB 2
TINJAUAN PUSTAKA

2.1. Nine Men's Morris

Nine men's morris merupakan salah permainan papan dengan 2 pemain, untuk lebih jelasnya dapat dilihat pada pengenalan nine men's morris, peraturan nine men's morris dan strategi nine men's morris.

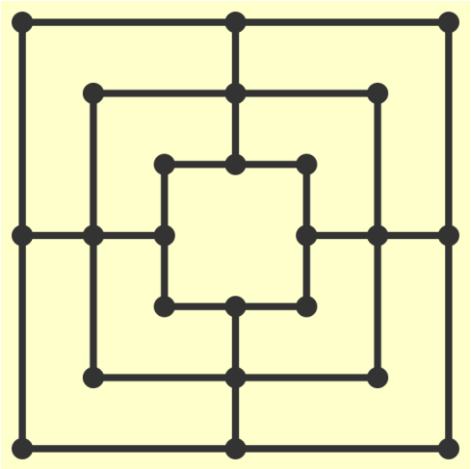
2.1.1. Pengenalan Nine Men's Morris

Nine Men's Morris adalah sebuah permainan yang sudah sangat tua, bahkan salah satu yang tertua sepanjang sejarah. Permainan ini banyak ditemukan di berbagai negara. Papan tertua yang pernah ditemukan berada di Mesir, diperkirakan berasal dari 1400 SM. Nama Nine Men's Morris sendiri tercipta ketika permainan ini mencapai Inggris, jauh setelah permainan ini tercipta, yaitu pada masa Middle Ages. Pada zaman inilah Nine Men's Morris meraih popularitas di antara kalangan masyarakat.

Permainan ini juga memiliki banyak versi. Nine Men's Morris memiliki 9 bidak untuk masing-masing pemain. Ada empat permainan sejenis yang memiliki aturan mirip tapi jumlah bidaknya berbeda, yaitu Three Men's Morris, Six Men's Morris, Eleven Men's Morris, dan Twelve Men's Morris. Selain itu, ada juga varian dengan aturan Nine Men's Morris yang dimodifikasi, contohnya Misere, Diagonals, dan No Fly.

Ada berbagai nama lain untuk Nine Men's Morris. Contohnya, di Jerman disebut Mühle, di Perancis disebut Jeu De Moulin atau Merelles, di India disebut Sujjua, di Cina disebut Sam K'i, di Mexico disebut Picaria, dan masih banyak lagi. Hal ini menunjukan opularitas Nine Men's Morris yang melintasi banyak benua.

Papan yang digunakan berupa 3 buah persegi berbeda ukuran dengan yang lebih kecil berada di dalam persegi yang lebih besar. Titik tengah sisi sebuah persegi terhubung dengan titik tengah sisi dua persegi lain dengan sebuah garis. Bidak, yang berjumlah 9 dengan warna biasanya masing-masing hitam dan putih, diletakkan pada perpotongan antara dua garis[8]. Adapun papan permainan dari nine men's morris dapat dilihat pada gambar 2.1 :

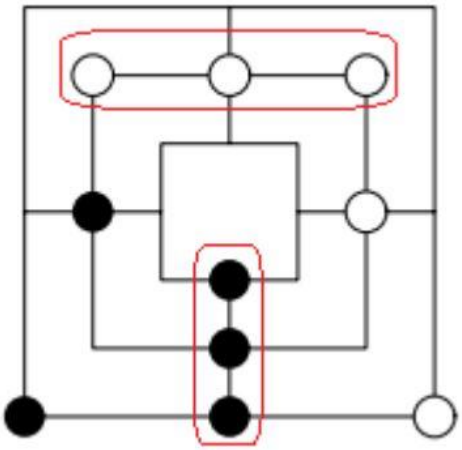


Gambar 2.1 Contoh Papan Permainan Nine Men's Morris

2.1.2. Peraturan Permainan

Pada permainan Nine Men's Morris, untuk mengalahkan lawan seorang pemain harus membentuk "mill" (atau disebut juga "morris") yaitu garis lurus dengan tiga bidak berwarna sama. Jika mill berhasil terbentuk, pemain berhak mengambil bidak pemain lain yang tidak berada di dalam garis mill.

Hanya ada dua kemungkinan bentuk mill yang dapat diraih. Kemungkinan pertama yaitu tiga bidak yang berada pada sisi yang sama dari persegi ketiga atau persegi kedua. Kemungkinan lain adalah tiga bidak yang berada pada garis pemisah dengan arah mata angin yang sama[8]. Adapun contoh terbentuknya mill dapat dilihat pada gambar 2.2.



Gambar 2.2 Contoh Terbentuknya Mill

Permainan ini terdiri dari tiga fase dan pada ketiganya pemain dapat membentuk mill dan mengambil bidak pemain lawan.

1) Fase Pertama : Meletakkan Bidak

   Pada fase ini, masing-masing pemain secara bergantian meletakkan bidaknya pada papan. Jika seorang pemain berhasil membentuk mill, pemain tersebut boleh mengambil bidak pemain lawan di papan. Fase ini berakhir ketika seluruh bidak telah diletakkan pada papan permainan atau diambil pemain lain.

2) Fase Kedua : Menggerakkan Bidak

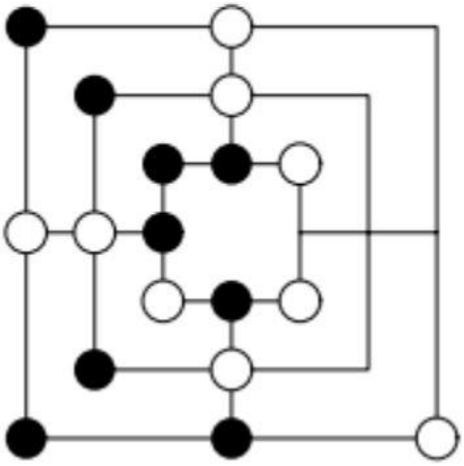
   Pada fase ini, pemain dapat menggeser bidaknya secara bergiliran sesuai dengan garis yang ada. Kedua pemain dapat saling berlomba untuk membentuk mill atau menghalangi lawan membentuk mill.

3) Fase Ketiga : Flying

   Fase ketiga dimulai ketika seorang pemain hanya tinggal memiliki tiga bidak. Dalam fase ini pemain tersebut dapat memindahkan bidaknya ke posisi manapun di dalam papan.

Permainan berakhir berdasarkan tiga kondisi, yaitu :

1) Kondisi pertama adalah salah satu pemain tidak bisa menggerakkan bidaknya kemanapun, maka pemain itu dinyatakan kalah. Contoh kondisi pertama dapat dilihat pada gambar 2.3.



Gambar 2.3 Pemain Hitam Tidak Bisa Bergerak

2) Kondisi kedua adalah jika seorang pemain memiliki kurang dari tiga bidak, maka pemain itu dinyatakan kalah.
3) Kondisi ketiga adalah ketika posisi seluruh bidak terulang, maka permainan dinyatakan seri.

2.1.3. Strategi Permainan

Ada beberapa strategi yang umum digunakan oleh para pemain dalam memenangkan permainan Nine Men's Morris :

1. Lokasi yang strategis

   Pada fase pertama, sangat penting untuk meletakkan para bidak di posisi yang strategis (mudah untuk digerakkan, tidak terhalang), dan tidak berfokus untuk membuat mill. Contoh posisi yang strategis adalah titik tengah di kotak terdalam, dimana bidak dapat bergerak pada tiga arah. Namun meletakkan bidak condong ke salah satu kotak juga tidak direkomendasi, distribusi bidak harus merata.

2. Menghalangi pemain lain

   Keunikan permainan ini adalah mill yang dibentuk dapat digunakan berkali-kali. Ketika seorang pemain sudah membentuk sebuah mill, dia bisa memindahkan bidaknya ke tempat lain pada giliran berikutnya, dan mengembalikannya ke tempat semula pada giliran selanjutnya, dan mill akan erbentuk kembali. Maka dia berhak mengambil dua bidak pada tiga giliran tersebut. Kondisi seperti ini harus dicegah oleh pemain lawannya, dengan cara

meletakkan bidaknya sendiri di posisi yang ditinggalkan pada giliran kedua. Taktik ini disebut dengan blocking. Atau jika blocking tidak memungkinkan, pemain tersebut juga bisa mencoba membentuk mill sendiri, sehingga pemain lawan harus memilih antara melanjutkan membentuk mill atau melakukan blocking.

3. Double mill

   Double mill dibentuk dengan dua mill yang bersisian sedemikian rupa sehingga pemain dapat menggerakkan bidak dari sebuah mill yang sudah terbentuk ke posisi yang menyebabkan mill lain terbentuk. Double mill adalah kondisi yang sangat menguntungkan bagi seorang pemain, karena dia bisa mengambil bidak lawan pada setiap giliran. Double mill harus menjadi prioritas blocking pemain lawan[8].

2.2. Kecerdasan Buatan

Kecerdasan buatan merupakan salah satu bidang ilmu komputer yang didefinisikan sebagai kecerdasan yang dibuat untuk suatu sistem dengan menggunakan algoritma-algoritma tertentu sehingga sistem tersebut seolah-olah dapat berpikir seperti manusia[2]. Terdapat beberapa tujuan akhir dari kecerdasan buatan yaitu :

1. Memahami pola pikir manusia, mencoba untuk mendapatkan pengetahuan ingatan manusia yang mendalam, kemampuan dalam memecahkan masalah, belajar, dan mengambil keputusan.
2. Otomatisasi, menciptakan sistem yang dapat menggantikan manusia dalam tugas-tugas intelegensi. Menggunakan sistem yang performanya sebaik manusia dalam melakukan pekerjaan.
3. Penguatan intelegensi, membangun sistem untuk membantu manusia agar mampu berpikir lebih baik dan lebih cepat.
4. Intelegensi manusia super, membangun sistem yang mempunyai kemampuan untuk melebihi intelegensi manusia.
5. Menyelesaikan permasalahan, sistem mampu menyelesaikan berbagai masalah yang luas.

6. Wacana koheren, mampu berkomunikasi dengan manusia menggunakan bahasa alami.
7. Belajar, mampu memperoleh data sendiri dan mengetahui bagaimana cara memperoleh data. Sistem mampu membuat hipotesis, penerapan atau pembelajaran secara heuristik dan membuat alasan dengan analogi.
8. Otonomi, mempunyai sistem intelegensi yang beraksi atas inisiatif sendiri.
9. Informasi, mampu menyimpan informasi dan mengetahui cara mengambil informasi[6].

2.3. Agen Cerdas

Agen adalah sesuatu yang dapat mengesan lingkungannya melalui sensorsdan mengambil tindakan terhadap lingkungannya melalui actuators. Dengan adanya agen cerdas, maka diharapkan sistem mampu berpikir dan menentukan pilihan langkah yang tepat[7]. Terdapat beberapa karakteristik lingkungan agen yaitu sebagai berikut :

1. *Fully observable – partially observable*

   Apabila sensor pada suatu agen dapat mengakses seluruh keadaan pada lingkungan, maka lingkungan itu dapat dikatakan *fully observable* terhadap agen. Lebih efektif lagi lingkungan dikatakan *fully observable* jika sensor dapat mendeteksi seluruh aspek yang berhubungan dengan pilihan aksi yang akan dilakukan. Lingkungan yang *fully observable* biasanya sangat memudahkan, karena agen tidak perlu mengurus keadaan internal untuk terus melacak keadaan lingkungan. Suatu lingkungan bisa menjadi *partially observable* akibat ada gangguan dan ketidakakurasian sensor ataupun karena ada bagian keadaan yang hilang dari data sensor. Permainan nine men’s morris bersifat *fully observable* karena seluruh keadaan pada papan permainan dan bidak-bidak yang ada semua dapat dipersepsi dengan baik.

2. *Deterministic – stochastic*

   Apabila keadaan lingkungan selanjutnya sepenuhnya bergantung pada keadaan sekarang dan juga tindakan yang akan dilakukan oleh agen, maka lingkungan tersebut bersifat *deterministic*. Sedangkan *stochastic* adalah

kebalikan dari *deterministic*, dimana keadaan selanjutnya tidak bergantung pada keadaan sekarang dan juga tindakan yang akan dilakukan oleh agen. Apabila lingkungan bersifat deterministic terkecuali untuk tindakan dari agen, maka lingkungan tersebut bersifat strategic. Permainan nine men's morris bersifat *deterministic* karena keadaan selanjutnya bergantung pada keadaan sekarang.

3. *Episodic – sequential*

   Untuk lingkungan yang bersifat *episodic*, pengalaman agen dibagi-bagi menjadi beberapa episode pendek. Tiap episode terdiri dari apa yang dirasakan agen dan kemudian melakukan satu tindakan tertentu. Kualitas dari tindakan agen hanya tergantung pada episode itu saja, karena tindakan selanjutnya tidak tergantung pada tindakan apa yang akan dilakukan di episode sebelumnya. Lingkungan *episodic* lebih sederhana karena agen tidak perlu memikirkan langkah-langkah pada keadaan selanjutnya. Sedangkan pada lingkungan *sequential*, tindakan saat sekarang dapat mempengaruhi tindakan selanjutnya. Permainan nine men's morris bersifat *sequential* karena agen berpikir untuk langkah- langkah selanjutnya dan seluruh langkah yang akan diambil oleh agen saling bergantung.

4. *Static – dynamic*

   Apabila lingkungan dapat berubah saat agen sedang mengambil keputusan, maka lingungan tersebut bersifat *dynamic*, sebaliknya bersifat *static*. Lingkungan yang bersifat *static* lebih mudah dihadapi karena agen tidak perlu memperhatikan lingkungannya saat dia sedang mengambil tindakan, maupun waktu yang terus berjalan. Apabila lingkungan tidak berubah seiring waktu berjalan, namun menyebabkan nilai kemampuan agen berubah-ubah, maka lingkungan tersebut bersifat *semidynamic*. Permainan nine men's morris bersifat *static* karena saat agen mengambil tindakan, lingkungan tidak berubah dan juga tidak mempengaruhi nilai kemampuan agen.

5. *Discrete – continuous*

   Apabila kesan dan tindakan yang akan diterima dan dilakukan oleh agen telah ditetapkan dengan jelas, maka lingkungan tersebut bersifat *discrete*. Catur bersifat *discrete*, karena langkah yang akan diambil terbatas dan tertentu. Sedangkan pengendara taxi bersifat *continuous,* karena kecepatan dan lokasi pada taksi untuk suatu jangka tertentu mempunyai nilai yang terus-menerus berubah. Permainan nine men's morris bersifat *discrete* karena seluruh kesan dan tindakan telah jelas ditetapkan sesuai dengan peraturan permainan.

6. *Single agent – multiagent*

   Agen pemecah permainan teka teki silang berada pada lingkungan yang bersifat *single agent*. Agen pemain catur berada pada lingkungan yang bersifat *multiagent*. Ada hal lain yang memberikan perbedaan lingkungan agen, yaitu akan hal apakah agen memberikan bantuan kepada agen lain atau apakah agen akan memaksimalkan kemampuannya bergantung pada prilaku agen lain. Permainan nine men's morris bersifat *multiagent* karena memikirkan langkah yang akan diambil oleh lawan[12].

2.4. Pohon Permainan

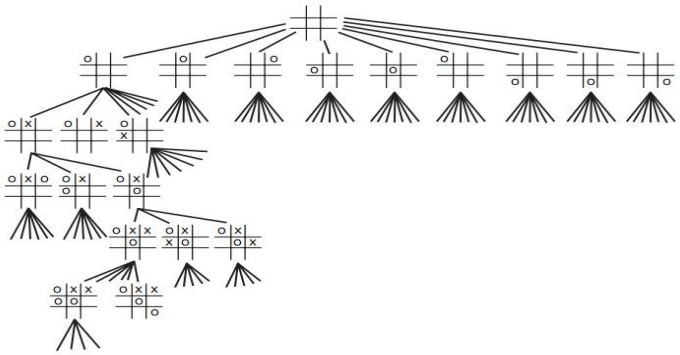
Pohon permainan mempresentasikan kepada pemain kondisi - kondisi yang mungkin dihadapi pada permainan dimulai dari kondisi yang sedang dihadapi sekarang hingga beberapa kondisi ke depan. Sebuah pohon permainan merupakan representasi grafis dari contoh permainan. Pohon permainan menyediakan informasi pemain, hasil, strategi, dan pilihan langkah.

Pohon permainan dapat direpresentasikan dengan baik pada permainan yang dilakukan secara giliran (turn-based-game). Pohon permaian memiliki root yang merupakan representasi dari kondisi langkah yang belum diambil, node pada pohon mempresentasikan keadaan-keadaan yang mungkin diambil pada permainan, dan Arcs yang mempresentasikan langkah.

Pengguna pohon permainan pada permainan yang dimainkan oleh dua pemain direpresentasikan dengan cara bergantian. Untuk edges dari tingkat pertama keringat kedua mempresentasikan langkah - langkah yang dapat diambil

oleh pemain pertama, sedangkan untuk edges dari tingkat kedua ketingkat ketiga mempresentasikan langkah-langkah yang diambil oleh pemain kedua, dan begitu seterusnya.

Leaf node pada pohon permainan mempresentasikan keadaan akhir pada permainan, Diana pemain tersebut dimenangkan, dikalahkan ataupun seri. Pada permainan sederhana, untuk mencapai leaf node mungkin dapat direpresentasikan, tetapi untuk permainan yang rumit seperti Damdaman, pencapaian leaf node angat tidak memungkinkan karena percabangan pada pohon permainan yang sangat besar.



Gambar 2.4 Contoh Pohon Permainan Tic Tac Toe[1]

Berikut penjelasan pohon permainan *tictactoe* pada gambar 2.4:

1. Terdapat *root* yang merupakan keadaan awal Dimana pemain belum mulai bermain dan langkah belum diambil.
2. *Edges* yang menghubungkan tingkat pertama (*root*) dengan tingkat kedua merupakan langkah pemain pertama dan begitu seterusnya. Sehingga ohon permainan tersebut merepresentasikan langkah kedua pemain secara bergantian.
3. Untuk *nodes* pada pohon tersebut merepresentasikan keadaan-keadaan yang dapat diambil oleh pemain yang akan melangkah.
4. Percabangan pertama yang dihasilkan adalah 9, kemudian untuk percabangan berikutnya adalah 8, dan begitu seterusnya hingga mencapai keadaan akhir (*leaf nodes*).

2.5. Algoritma Negascout

Negascout merupakan optimalisasi minimax dengan mempersempit ruang pencarian (*minimal search window*), dengan semakin sempitnya selisih nilai alpha dan beta maka semakin besar kemungkinan terjadinya pemotongan pencarian. Negascout memiliki dasar bahwa langkah-langkah setelah langkah pertama akan menghasilkan pemotongan, maka mengevaluasi semua langkah adalah sia-sia. Negascout akan mengecek dengan *null window* terlebih dahulu yang dinotasikan dengan m dan n dimana m adalah batas atas dan n adalah batas bawah, null window memiliki batas atas dan bawah yang berselisih satu. Ketika sebuah node memiliki nilai yang lebih tinggi dari m maka akan dilakukan pencarian ulang dengan menggunakan window yang lebih besar untuk mengetahui nilai yang terbaik[1]. Pada gambar 2.5 dapat dilihat pseudocode algoritma Negascout.

```
1 FUNCTION negascout (p: POSITION; alpha, beta, depth:
  INTEGER) : INTEGER;
2 VAR i,t,m,n: INTEGER;
3 BEGIN
4  IF depth = d THEN RETURN (evaluate(p))
5  ELSE
6  BEGIN m := -INFINITY;
7        n := beta;
8        FOR i := 1 TO b DO
9        BEGIN t:= -negascout (p.i,-n,-max(alpha,m),depth+1);
10             IF t > m THEN
11                IF (n = beta) OR (depth >= d-2)
12                THEN m := t
13                ELSE m := -negascout (p.i,-beta,-t,depth+1);
14             IF m >= beta THEN RETURN (m);
15             n := max (alpha, m) +1;
16       END;
17       RETURN (m);
18  END;
19 END;
```

Gambar 2.5 Pseudo-code algoritma Negascout[5]

Keterangan pseudo code:

p = kondisi papan

d = kedalaman maksimum dari pohon yang ditelusuri

b = lebar pohon (banyak langkah yang mungkin terjadi pada papan)

evaluate(p) = merupakan fungsi yang memberikan nilai evaluasi pada kondisi papan “p”.

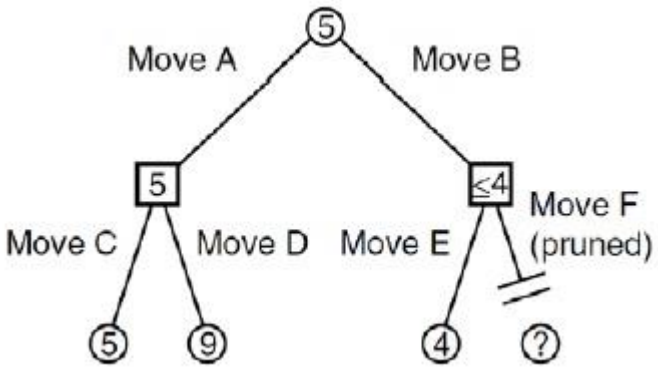
Gambar 2.5 merupakan *pseudocode* untuk algoritma Negascout, berdasarkan algoritma diatas dapat dijelaskan cara kerja algoritma negascout sebagai berikut, *statement* pertama sama seperti *alpha-beta*, yaitu jika posisi p adalah *node leaf* maka negascout akan mengembalikan nilai fungsi evaluasi (baris 4). Selain itu *variable* m dan n diinisialisasi dengan -∞ dan *beta* (baris 6 dan 7). Kemudian negascout akan melakukan "*scout*" pada node anak dari p dari kiri ke kanan. *Node* anak paling kiri akan dicari dengan menggunakan interval(-beta,-alpha) dan anak lainnya dicari dengan *zero-width window* (-m-1, -m) yang sudah diisi pada baris 15 sesudah melakukan pencarian anak paling kanan, karena *null window* ini tidak memiliki elemen, maka pencarian pasti akan gagal. Arah dari kegagalan ini menunjukkan apakah node tersebutdapat dipotong atau tidak.

Jika *null window* gagal karena t > m (baris 10), negascout harus mengecek kembali node tersebut dengan *search window* yang lebih lebar untuk mengetahui nilai aslinya. Hanya terdapat dua kasus dimana tidak diperlukan pencarian ulang yaitu, ketika n=beta (baris 11), dan negascout's "*fail-soft refinement*"selalu mengembalikan nilai minimax yang benar pada dua level terbawah. Pada kasus lainnya pencarian ulang harus dilakukan dengan menggunakan *search window* baru yaitu (-beta,-t) (baris 13).

Kondisi untuk pemotongan (baris 14) sama seperti alpha-beta : jika m ≥beta maka semua node anak lain dapat diabaikan[5].

2.5.1. Pemotongan Alpha

Gambar 2.6 menunjukan satu pohon permainan sebelum proses pemotongan dilakukan. Untuk lebih mudah melihat bagaimana nilai nilai di proses, akan dicoba dlakukan dengan algoritma Minimax untuk ilustrasi ini.



Gambar 2.6 Pohon Permainan dengan Proses Pemotongan *Alpha* [3]

Untuk proses pertama, sama halnya dengan algoritma Minimax. Jika pemain memilih langkah A, maka lawan akan membiarkan respons dengan langkah C, dan memberikan pemain nilai 5. Maka nilai 5 naik pada tingkat min. Jika pemain memilih langkah B maka lawan akan melangkah ke E, sehingga pemain akan mendapat nilai 4. Setelah melihat kondisi demikian pemain akan menyadari nilai yang didapatkan lebih rendah dari nilai sementara di tingkat max. Sehingga pemain akan menyimpulkan ini adalah langkah yang salah.

Untuk melakukan pemotongan dengan cara ini, perlu dilakukan pencarian nilai terbaik yang pasti akan tercapai. Bahkan, nilai ini membentuk batas bawah pada nilai yang akan tercapai. Sehingga urutan langkah yang lebih baik dapat ditemukan dalam pencarian, namun nilai yang paling rendah akan mustahil untuk didapatkan. Batas bawah ini dinamakan nilai alpha dan pemotongannya dinamakan dengan pemotongan alpha.

2.5.2. Pemotongan Beta

Pemotongan beta bekerja dengan cara yang sama. Nilai beta terus melacak nilai batas tertinggi dari nilai yang diharapkan. Nilai beta berubah ketika ditemukan satu langkah yang dipaksakan oleh lawan.

Pada posisi itu pemain mengetahui tidak ada nilai yang lebih besar dari beta. Jika urutan langkah yang bernilai lebih besar dari nilai beta ditemukan maka dari kondisi ini dapat diabaikan, karena kesempatan untuk mendapatkannya lebih kecil.

2.5.3. Pemotongan Alpha Beta

Bersamaan nilai – nilai dari alpha dan beta memberikan satu jendela kemungkinan langkah. Pemain etika pernah memilih untuk melakukan langkah yang dinilainya lebih kecil dari alpaha, dan lawan tidak akan pernah membiarkan ada langkah yang lebih besar dari beta. Nilai pada akhirnya harus terletak antara alpha dan beta. Ketika pencarian mulai dilakukan pada pohon, nilai - nilai dari alpha dan beta diperbaharui. Jika cabang dari pohon yang ditemukan di luar nilai - nilai ini, maka akan dilakukan pemotongan pada cabang.

2.6. OOP (*Object Oriented Programming*)

Objek adalah kesatuan entitas yang memiliki sifat dan tingkah laku. Dalam kehidupan sehari-hari, objek adalah benda, baik benda berwujud nyata seperti manusia, hewan, mobil, komputer, handphone, pena, ataupun benda yang tidak nyata atau konsep, seperti halnya tabungan bank, sistem antrian, sistem internet banking, dan sebagainya. Jadi pengertian OOP adalah konsep yang membagi program menjadi objek-objek yang saling berinteraksi satu sama lain. Objek adalah benda, baik benda yang berwujud nyata maupun benda yang tidak nyata (konsep). Jika kita menggunakan OOP maka akan ada enam keuntungan yang dapat diperoleh, yaitu [10] :

1. Alami (*Natural*)
2. Dapat diandalkan (*Reliable*)
3. Dapat digunakan kembali (*Reusable*)
4. Mudah untuk perawatan (*Maintainable*)
5. Dapat diperluas (*Extendable*)
6. Efisiensi waktu

Berikut ini beberapa bahasa pemrograman yang sudah menggunakan konsep OOP, adalah :

1. *C++*
2. *Visual C++*
3. *Visual Basic*
4. *Java*

2.7. Pemodelan Dengan UML

UML (*Unified Modeling Language*) adalah salah satu alat bantu yang sangat handal di dunia pengembangan sistem yang berorientasi obyek[10]. Hal ini disebabkan karena UML menyediakan bahasa pemodelan visual yang memungkinkan bagi pengembang sistem untuk membuat cetak biru atas visi mereka dalam bentuk yang baku, mudah dimengerti serta dilengkapi dengan mekanisme yang efektif untuk berbagi dan mengkomunikaskan rancangan mereka dengan yang lain.

UML merupakan kesatuan dari bahasa pemodelan yang dikembangkan oleh *Booch, Object Modeling Technique (OMT)* dan *Object Oriented Software Engineering (OOSE)*. Metode *Booch* dari *Grady Booch* sangat terkenal dengan nama metode *Design Object Oriented*. Metode ini menjadikan proses analisis dan design ke dalam empat tahapan iterative, yaitu: identifikasi kelas-kelas dan obyek-obyek, identifikasi semantic dari hubungan obyek dan kelas tersebut, perincian interface dan implementasi. Keunggulan metode *Booch* adalah pada detail dan kayanya dengan notasi dan elemen. Pemodelan *Object Modeling Technique* yang dikembangkan Rumbaugh didasarkan pada analisis terstruktur dan permodelan entity-relationship. Tahapan utama dalam metodologi ini adalah analisis, design sistem, design obyek dan implementasi. Keungulan metode ini aladah dalam penotasian yang mendukung semua konsep *Object Oriented Software Engineering*. Metode *Object Oriented Software Engineering Jacobson* lebih memberikan penekanan pada *use case*. *Object Oriented Software Engineering* memiliki tiga tahapan yaitu membuat model requirement dan analisis, design dan implementasi, dan model pengujian. Keungulan metode ini adalah mudah dipelajari karena memiliki notasi yang sederhana namun mencakup seluruh tahapan dalam rekayasa perangkat lunak.

Dengan UML, metode *Booch, Object Modeling Technique* dan *Object Oriented Software Engineerin*g digabungkan dengan membuang elemen-elemen yang tidak praktis ditambah dengan elemen-elemen dari metode lain yang lebih efektif dan elemen-elemen baru yang belum ada pada metode terdahulu sehingga UML lebih ekspresif dan seragam dari pada metode lainnya.

2.7.1. Diagram – Diagram UML

Didalam UML terdapat beberapa macam diagram yang dapat menggambarkan suatu sistem, berikut adalah beberapa diagram yang terdapat di dalam UML.

1. *Use Case* dan *Use Case Diagram*

   Use case adalah deskripsi fungsi dari sebuah sistem dari perspektif pengguna. *Use case* bekerja dengan cara mendeskripsikan tipikal antara pengguna sebuah sistem dengan sistemnya sendiri melalui sebuah cerita bagaimana sebuah sistem dipakai. Urutan langkah-langkah yang menerangkan antara pengguna dan sistem disebut *scenario*. Setiap *scenario* mendeskripsikan urutan kejadian. Setiap urutan diinisialisasi oleh orang, sistem yang lain, perangkat keras atau urutan waktu. Dengan demikian secara singkat bisa dikatakan, *use case* adalah serangkaian *scenario* yang digabungkan bersama-sama oleh tujuan umum pengguna.

   Use case diagram adalah penggambaran interaksi pengguna sistem (*actor*) dengan kasus (*use case*) yang telah disesuaikan dengan langkah-langkah (*scenario*). Diagram *use case* menunjukan 3 aspek dari system yaitu *actor*, *use case* dan sistem, sub sistem *boundary*. *Actor* mewakili perang orang, sistem yang lain atau alat ketika berkomunikasi dengan *use case*.

2. *Class Diagram*

   Class dalam notasi UML digambarkan dengan kotak. *Class* adalah sebuah spesifikasi yang jika diinstansiasi akan menghasilkan sebuah obyek dan merupakan inti dari pengembangan dan desain berorientasi obyek. *Class* menggambarkan keadaan (*atribut* atau *properti*) suatu sistem, sekaligus menawarkan layanan untuk memanipulasi keadaan tersebut (metoda atau fungsi).

   Class mempunyai beberapa bagian yang menjelaskan isi dari *class*:

   a. *Attribute* adalah *property* dari sebuah *class*. *Attribute* ini melukiskan batas nilai yang mungkin ada pada obyek *class*. Sebuah *class* mungkin mempunyai nol atau lebih *attribute*.

b. *Operation* adalah sesuatu yang bisa dilakukan oleh sebuah *class* atau *class* yang lain dapat lakukan untuk sebuah *class*.
c. *Responsibility* adalah keterangan tentang apa yang akan dilakukan *class* yaitu apa yang akan dicapai oleh *attribute* dan *operation*.

3. *Statechart diagram*

Menelusuri individu-individu obyek melalui keseluruhan daur hidupnya, menspesifikasikan semua urutan yang mungkin dari pesan-pesan yang akan diterima obyek tersebut, bersama-sama dengan tanggapan atas pesan-pesan tersebut.

Statechart diagram menggambarkan transisi dan perubahan keadaan (dari satu state ke state lainnya) suatu obyek pada sistem sebagai akibat dari stimulasi yang diterima [10]. Pada umunya statechart diagram menggambarkan *class* tertentu (satu *class* dapat memiliki lebih dari satu *statechart diagram*).

Simbol UML untuk state diagram adalah segiempat yang tiap pojoknya dibuat rounded, Titik awalnya menggunakan lingkaran solid yang diarsir dan diakhiri dengan mata.

4. *Activity Diagram*

Activity Diagram adalah teknik untuk mediskripsikan logika procedural, proses bisnis dan aliran kerja dalam banyak kasus [10]. *Activity diagram* mempunyai peran seperti halnya *flowchart*, akan tetapi perbedaannya dengan *flowchart* adalah *activity* diagram bisa mendukung perilaku parallel sedangkan *flowchart* tidak bisa.

Activity diagram tidak menunjukan apa yang terjadi, tetap tidak menunjukan siapa yang melakukan apa. *Activity* diagram menggambarkan berbagai alir *aktivitas* dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, decision yang mungkin terjadi dan bagaimana mereka berakhir, *Activity diagram* juga dapat menggambarkan proses parallel yang mungking terjadi pada beberapa eksekusi. Simbol-simbol yang sering digunakan pada pembuatan *activity diagram*.

5. *Sequence Diagram*

Sequence diagram digunakan untuk menggambarkan perilaku pada sebuah *scenario*. Diagram ini menunjukan sejumlah contoh obyek dan *message* pesan yang diletakan diantara obyek-obyek ini dalam *use case*.

Sequence diagram menggambarkan interaksi antar obyek di dalam dan di sekitar sistem berupa *message* yang digambarkan terhadap waktu [10]. *Sequence diagram* terdiri atas dimensi *vertical* (waktu) dan dimensi *horizontal* (obyek-obyek yang terkait). Sequence diagram biasa digunakan untuk menggambarkan *scenario* atau rangkaian langkah-langkah yang dilakukan sebagai respons dari sebuah event untuk menghasilkan *output* tertentu. Diawali dari apa yang men-*trigger* aktivitas tersebut, proses dan perubahan apa saja yang terjadi secara *internal* dan *output* apa yang dihasilkan.

Komponen utama sequence diagram terdiri atas obyek yang dituliskan dengan kotak segiempat bernama. *Message* diwakili oleh garis dengan tanda panah dan waktu yang ditunjukan dengan *progress vertical*. Penjelasan dari komponen utama *sequence diagram* yaitu :

a. Obyek atau *Participant*

Obyek atau *participant* diletakan di dekat bagian atas diagram dengan urutan dari kiri ke kanan. Mereka diatur dalam urutan guna menyederhanakan diagram. Setiap *participant* terhubung dengan garis titik-titik yang disebut *lifeline*. Sepanjang *lifeline* ada kotak yang disebut *activation*. *Activation* mewakili sebuah eksekusi *operation* dari *participiant*. Panjang kotak ini berbanding lurus dengan durasi *activation*.

b. *Message*

Sebuah *message* bergerak dari satu *participant* ke *participant* yang lain dan dari satu *lifeline* ke *lifeline* yang lain. Sebuah *participant* bisa mengirim sebuah *message* kepada dirinya sendiri.

c. *Time*

Time adalah diagram yang mewakili waktu pada arah *vertical*. Waktu dimulai dari atas kebawah. *Message* yang lebih dekat dari atas akan

dijalankan terlebih dahulu dibandingkan message yang lebih dekat ke bawah.

6. *Collaboration diagram*

   Collaboration diagram adalah perluasan dari obyek diagram. Obyek diagram menunjukan obyek-obyek dan hubungannya satu dengan yang lain. Collaboration diagram menunjukan *message-message* obyek yang dikirimkan satu sama lain [10].

   Collaboration diagram juga menggambarkan interaksi antar obyek seperti *sequence diagram,* tetapi lebih menekankan pada peran masing-masing obyek dan bukan pada waktu penyampaian *message*. Setiap *message* memiliki *sequence number*, dimana *message* dari level tertinggi memiliki nomor satu *Message* dari level yang sama memiliki *prefix* yang sama.

   Dengan *collaboration diagram* memungkinkan untuk memodelkan pengiriman sebuah message ke banyak obyek pada *class* yang sama. Demikian juga halnya untuk menunjukan adanya obyek aktif yang mengendalikan aliran *message*.

7. *Component diagram*

   Component diagram merepresentasikan dunia nyata item yaitu *component software*. *Component* diagram menggambarkan struktur dan hubungan antar komponen piranti lunak, termasuk ketergantungan (*dependency*) diantaranya. *Component software* adalah modul berisi *code*, baik berisi *source code* maupun *binary code*, baik *library* maupun *executable*, baik yang muncul pada *compile time*, *link time*, maupun run time. Umumnya komponen terbentuk dari beberapa *class* dan atau *package*, tapi dapat juga dari komponen-komponen yang lebih kecil. Komponen dapat juga berupa *interface*, yaitu kumpulan layanan yang disediakan sebuah komponen untuk komponen lain. *Component diagram* mengandung *component*, *interface* dan *relationship* [10].

8. *Deployment diagram*

   Deployment diagram menunjukan tata letak sebuah sistem secara fisik, menampakan bagian-bagian *software* yang berjalan pada bagian-bagian *hardware* [10].

   Deployment diagram menyediakan gambaran bagaimana sistem secara fisik akan terlihat. Sistem terdiri dari *node-node* dimana setiap *node* diwakili untuk sebuah kubus. Garis yang menghubungkan antara dua kubus menunjukan hubungan diantara kedia *node* tersebut. Tipe *node* bisa berupa *device* yang berwujud *hardware* dan bisa juga *processor* (yang mengeksekusi *component*) atau *execution environment* (*software* yang menjadi *host* atau mengandung *software* yang lain).

2.8. Java

Java adalah bahasa pemrograman yang disusun oleh James Gosling yang dibantu oleh rekan-rekannya seperti Patrick Naughton, Chris Warth, Ed Rank, dan Mike Sheridan di suatu perusahaan perangkat lunak yang bernama Sun Microsystems pada tahun 1991. Bahasa pemrograman ini mula-mula diinisialisasi dengan nama “Oak”, namun pada tahun 1995 diganti namanya menjadi “Java”.