BAB 2
LANDASAN TEORI

2.1 Permainan/ Game

Game merupakan sebuah bentuk seni dimana penggunanya disebut dengan pemain (*player*) yang diharuskan membuat keputusan-keputusan dengan tujuan mengelola sumber daya yang diperoleh dari kesempatan-kesempatan bermain dan untuk mencapai tujuan tertentu. *Video game* adalah bentuk *game* yang interaksi umumnya melibatkan media *video* dan *audio* [5].

Menurut Andang Ismail terdapat dua pengertian *game* (permainan). Pertama, *game* (permainan) adalah sebuah aktifitas bermain yang murni mencari kesenangan tanpa mencari menang atau kalah. Kedua, permainan diartikan sebagai aktifitas bermain yang dilakukan dalam rangka mencari kesenangan dan kepuasan, namun ditandai pencarian menang atau kalah.

Berdasarkan representasinya, *game* dapat dibedakan menjadi 2 jenis yaitu *game* 2 dimensi (2D) dan 3 dimensi (3D). *Game* 2D adalah *game* yang secara matematis hanya melibatkan 2 elemen koordinat kartesius yaitu x dan y, sehingga konsep kamera pada *game* 2D hanya menentukan gambar pada *game* yang dapat dilihat oleh pemain. Sedangkan *game* 3D adalah *game* yang selain melibatkan elemen x dan y juga melibatkan elemen z pada perhitungannya sehingga konsep kamera pada *game* 3D benar-benar menyerupai konsep kamera pada kehidupan nyata.

2.2 Game Congklak

Congklak merupakan suatu permainan tradional yang yang dimainkan oleh 2 orang dengan menggunakan media papan. Biasanya dalam permainan ini menggunakan batu atau sejenis kulit kerang yang digunakan sebagai alat permainan [6].

Papan congklak terdiri dari sejumlah lubang kecil dan dual bang besar yang terdapat di masingmasing tiap ujung papan. Setiap pemain memiliki daerahnya sendiri. Pada awal permainan, semua lubang yang ada dipapan diisi dengan jumlah

yang sama (batu, biji atau kerang). Jumlah isi di setiap lubang biasanya sama dengan jumlah lubang di salah satu sisi. Tujuan dari permainan ini adalah untuk mengumpulkan isi lubang sendiri dan mengosongkan isi lubang lawan yang ada di papan permainan. Pemain yang memenangkan permainan adalah pemain yang mengumpulkan batu / kulit kerang terbanyak.

2.2.1 Sejarah Perkembangan Congklak

Congklak memiliki asal-usul dalam baik Afrika atau dunia Arab, tergantung pada teori sarjana Anda memilih untuk percaya. Beberapa bukti tertua ditemukan di National Geographic disponsori penggalian arkeologi dating kembali ke 7.000 menjadi 5.000 SM di zaman sekarang Yordania. Penggalian sebuah rumah menemukan sebuah lempengan batu kapur dengan dua baris paralel depresi melingkar. Tata letak adalah mudah dikenali oleh seorang arkeolog pada penggalian sebagai dewan bermain Congklak. Murray, seorang sarjana mencatat, menelusuri asal-usul untuk Usia Kekaisaran Mesir kuno (sekitar 15 sampai abad 11 SM). Banyak ahli menduga bahwa Congklak mungkin sebenarnya papan permainan tertua yang pernah ada [6].

Tulisan-tulisan paling awal tercatat menggambarkan permainan yang ditemukan di referensi dalam teks-teks agama Mancala Arab ke Abad Pertengahan. Beberapa sarjana percaya bahwa permainan berasal dari Timur Tengah dan menyebar dari sana ke Afrika. Kemudian, permainan menyebar ke Asia dengan pedagang Arab dan datang ke Karibia sekitar 1.640 melalui perdagangan budak Afrika. Ahli lain menempatkan asal-usul di Afrika Tengah.

Saat ini, game ini dikenal dengan berbagai nama di seluruh dunia. Namanama yang diambil dari budaya lokal dengan menggunakan kata-kata yang mencerminkan di mana permainan ini dimainkan, cara untuk menang, modus dari bermain dan papan atau counter digunakan. Hal ini disebut dalam bahasa Inggris saat Count dan Capture.

Di negara-negara Arab, nama yang paling umum adalah Mancala (makna kata Arab dalam bahasa Inggris "bergerak"). Di beberapa negara Afrika Barat yang depresi di papan disebut sebagai Warri atau Awari, yang berarti rumah, sehingga memberikan yang Wari nama. Di Nigeria permainan dikenal sebagai Adi, yang juga

merupakan nama dari bibit yang digunakan untuk bermain game. Karena popularitas yang luasnya, para penggemar telah mengembangkan situs web banyak mendokumentasikan berbagai versi dari permainan, untuk daftar link yang baik untuk banyak situs-situs, lihat http://www.cs.uu.nl/~hansb/d.gam/mancala.html. Lain situs web yang baik, http://www.myriadonline.com/en/resources/awalink .html, daftar 279 nama dari permainan yang ditemukan melalui penelitian mereka. Ketika Anda surfing melalui situs-situs web, itu menarik untuk melihat bagaimana masing-masing negara atau budaya memiliki adat istiadat yang berkaitan dengan pembuatan papan, yang bisa bermain, serta bagaimana dan kapan permainan ini dimainkan.

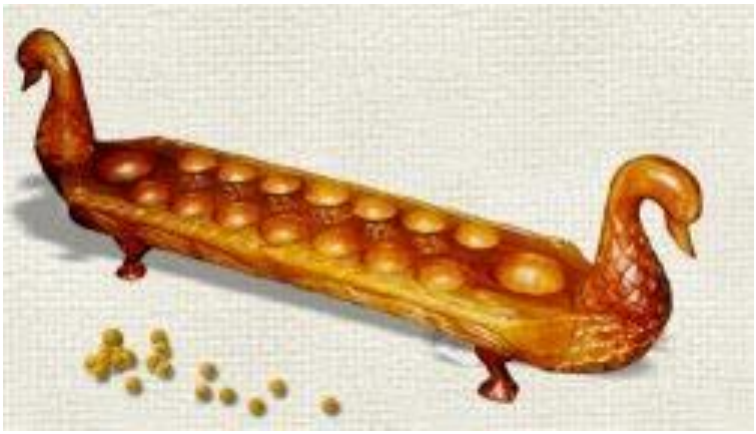
Beberapa fitur situs on-line versi dari permainan juga, jika Anda ingin bermain dengan penggemar lain di seluruh dunia. Pada topik congklak online, telah menyarankan bahwa game congklak terinspirasi mungkin mulai muncul di kasino online, yang sudah menawarkan permainan tradisional seperti sic bo dan Baccarat. Permainan ini sangat populer sehingga Museum British Museum Manusia menampilkan sebuah pameran Mancala, Wari dan papan Congklak lain bermain pada tahun 1997.

2.2.2 Congklak di Indonesia

Bahkan di Indonesia, Congklak dikenal dengan nama yang berbeda dari daerah ke daerah. Nama yang paling umum, Congklak, diambil dari shell cowrie, yang biasa digunakan untuk bermain game. Di Malaysia, permainan ini dikenal sebagai Congkak, nama yang digunakan dalam provinsi Sumatera juga banyak. Di Jawa, permainan ini dikenal sebagai Congklak, dakon, dhakon atau dhakonan.Di Lampung, permainan ini disebut, dentuman lamban. Di Sulawesi, permainan ini disebut sebagai Mokaotan, Maggaleceng, Aggalacang dan Nogarata [6].

Referensi historis untuk congklak merujuk pada permainan yang dimainkan oleh gadis-gadis muda dari bangsawan Jawa.Hal ini paling mungkin bahwa pedagang asing, karena kontak dekat mereka dengan kelas atas, diperkenalkan Congklak kepada mereka. Dengan berlalunya waktu, popularitas congklak tumbuh sampai sekarang yang banyak dimainkan oleh rakyat biasa juga. Di sebagian besar wilayah, Congklak bermain terbatas pada gadis-gadis muda, remaja dan wanita di

waktu luang mereka dan dilihat sebagai 'permainan wanita'. Dalam beberapa daerah adalah Congklak dimainkan oleh laki-laki dan anak laki-laki juga. Berikut gambar dari papan Congklak.



Sumber gambar : Expat [6]

Gambar 2.1 Papan Game Congklak

Di Sulawesi, historis, permainan disediakan untuk bermain hanya selama periode berduka, setelah kematian orang yang dicintai. Ini dianggap tabu untuk bermain game pada waktu lainnya. Di Jawa Tengah, pada masa pra-sejarah kali, Congklak digunakan oleh petani untuk menghitung musim, untuk mengetahui kapan harus menanam dan panen, serta untuk memprediksi masa depan.

2.3 Kecerdasan Buatan

Definisi kecerdasan buatan atau *artificial intelligent* yang paling tepat untuk saat ini adalah *Acting rationally* dengan pendekatan Rational agent. Hal ini berdasarkan pemikiran bahwa komputer bisa melakukan penalaran secara logis dan juga bisa melakukan aksi secara rasional berdasarkan hasil penalaran tersebut [7].

Kecerdasan buatan atau AI merupakan kegiatan membuat komputer agar dapat berpikir dan mengerjakan kegiatan yang dapat dilakukan oleh manusia maupun binatang. Saat ini dapat ditemukan program komputer yang memiliki kemampuan menangani masalah seperti *aritmatic, sorting, searching* [7]. Bahkan komputer juga dapat bermain beberapa board game seperti catur lebih baik dari pada manusia. Namun masih banyak hal yang tidak dapat dilakukan dengan baik oleh komputer. Seperti, mengenali wajah, berbicara bahasa manusia, menentukan sendiri apa yang harus dilakukan, dan bertingkah kreatif. Hal itu semua merupakan

domain dari AI untuk mencoba menentukan algoritma apa yang dibutuhkan untuk memenuhi kebutuhan diatas.

Dalam bidang akademik, beberapa peneliti AI termotivasi oleh filosofi, yaitu memahami alam pikiran dan alam kecerdasan dan membangun program untuk memodelkan bagaimana proses berpikir. Beberapa juga termotivasi oleh *psychology*, bertujuan untuk memahami mekanisme otak manusia dan proses mental. Dan lainya termotivasi oleh *engineering*, dengan tujuan membangun algoritma untuk melakukan kegiatan seperti manusia atau hewan. Dalam pembangunan *Game*, umumnya akan cenderung hanya pada sisi *engineering* yang bertujuan membangun algoritma yang dapat membuat *Game* karakter mengerjakan kegiatan seperti yang dilakukan manusia atau binatang [7].

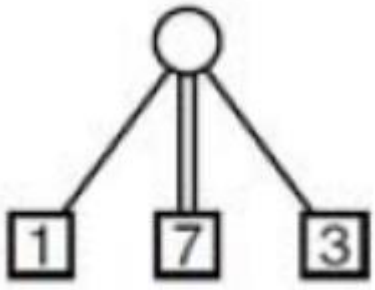
2.4 Algoritma Minimax

Pada permainan yang tidak terlalu kompleks, terdapat algoritma yang digunakan untuk mencari langkah terbaik dari situasi sekarang. Algoritma yang paling terkenal ialah algoritma *Minimax*. Algoritma *Minimax* ialah satu metode yang sangat bermanfaat untuk tipe permainan bergiliran dengan 2 pemain. Metode ini memilih langkah terbaik yang diberikan, Dimana kedua pemain berusaha mengatur sebuah strategi untuk memenangkan permainan. Sehingga semua langkah selanjutnya akan dapat diketahui [8].

Minimax merupakan algoritma yang digunakan untuk menentukan pilihan agar kemungkinan kehilangan nilai maksimal dapat diminalisir. Algoritma ini dapat diterapkan dengan baik pada permainan yang melibatkan dua pemain yang saling bergantian seperti *tic-tac-toe*, *chekers*, catur, dan permainan yang menggunakan strategi atau logika lainnya. Permainan tersebut memiliki karakteristik yang sama yakni “permainan dengan informasi lengkap”. Setiap pemain mengetahui kemungkinan langkah yang akan dilakukan lawan. Sehingga pemain bisa tetap “memantau” kondisi permainan sewaktu permainan sedang berlangsung.

Jika pemain memilih satu langkah, maka pemain akan memilih yang terbaik. Sehingga secara nalar pemain akan mentukan posisi yang terbaik untuk melangkah.

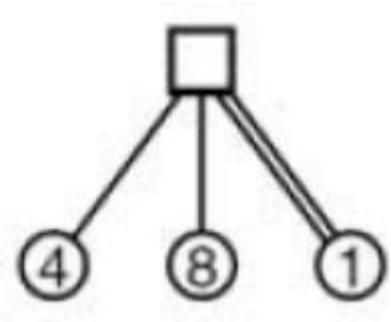
Dengan kata lain, ketika pemain mendapat giliran bermain maka pemain akan mencoba memaksimalkan nilainya, perhatikan gambar 2.2.



Sumber gambar : Coppin [9]

Gambar 2.2 Langkah Pemain pada Algoritma Minimax

Ketika lawan mendapat giliran bermain, pemain akan mengganggap bahwa lawan akan memilih langkah yang akan memberikan pemain posisi terburuk yang tersedia. Lawan sedang berusaha meminimalisir nilai dari pemain, perhatikan gambar 2.3.



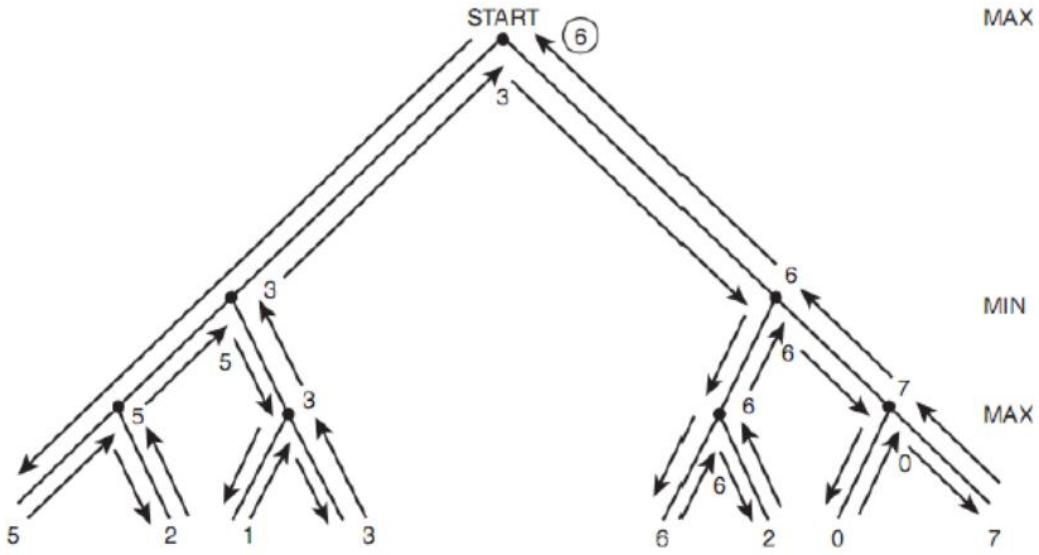
Sumber gambar : Coppin [9]

Gambar 2.3 Langkah Lawan pada Algoritma Minimax

Algoritma *Minimax* adalah salah satu implementasi dari pencarian DFS (*Dept-First Search*), sehingga dalam melakukan pencarian pada pohon *Minimax* mirip dengan apa yang dilakukan dengan DFS. DFS akan melakukan penelusuran simpul paling dalam terlebih dahulu, setelah simpul akar dibangkitkan, algoritma ini akan membangkitkan simpul pada tingkat kedua, yang akan dilanjutkan pada tingkat ketiga, dst [8].

Dalam representasi pohon pada algoritma *Minimax*, terdapat dua jenis node, yaitu node *Min* dan node *Max*. Maka akan memilih langkah dengan nilai tertinggi dan min akan memilih langkah dengan nilai tererendah. Untuk proses dan cara kerja

algoritma yang lebih jelasnya lagi, dapat dilihat pada gambar 2.4 yang merepresentasikan cara kerja algoritma *Minimax*.



Sumber gambar : Coppin [9]

Gambar 2.4 Illustrasi Cara Kerja Algoritma Minimax

Dari gambar 2.4, proses pencarian dimulai dari jalur paling kiri terlebih dahulu, sehingga DFS akan menelusuri simpul paling kiri bawah yaitu 5. Nilai 5 disimpan sebagai nilai maksimum sementara karena berada ditingkat *Max*, kemudian DFS melakukan backtrak dan menelusuri simpul berteangga dengan simpul 5 yaitu 2. Karena nilai 5 lebih besar dari nilai 2, maka nilai 2 tidak disimpan. Lalu DFS akan melakukan backtrack keringat min sehingga nilai 5 yang diperoleh akan disimpan sebagai nilai minimum sementara. Untuk simpul 1 dan 3, nilai 3 akan disimpan di node *Min* karena merupakan nilai maksimum di tingkat *Max*. Saat mencapai tingkat *Min*, sudah ada nilai minimum sementara yaitu 5,namun karena nilai 3 lebih kecil dari nilai 5, maka nilai 5 akan digantikan dengan nilai 3. Nilai 3 akan disimpan sebagia nilai maksimum sementara di tingkat paling atas karena merupakan tingkat *Max*. Lalu penelusuranjalur kanan akan dilakukan dengan cara yang sama dengan jalur kiri, sehingga diperoleh nilai 6. Karena nilai maksimum menara pada tingkat paling atas adalah 3, maka nilai 3 akan digantikan dengan nilai 6 karena nilai 6 lebih besar dari pada 3.dengan demikian, jalur yang akan dipilih menggunakan algoritma *Minimax* adalah jalur belah kanan karena untuk kondisi

terburuknya, pemain akan mendpat nilai 6 sedangkan jika pemain memilih jalur kiri, pemain hanya akan mendapat nilai 3 [8].

Pada dasarnya, algoritma *Minimax* sangat handal untuk menyelesaikan segala masalah dalam pencarian langkah untuk permainan komputer dengan jumlah kemungkinan penyelesaian yang kecil. Tetapi jika, algoritma *Minimax* digunakan untuk permainan dengan jumlah kemungkinan penyelesaian yang besar seperti permainan Damdaman, algoritma *Minimax* ini memelukan waktu lama untuk membangun pohon penyelesaian.

Oleh karena itu, beberapa metode lanjutan dari algoritma Minimax telah dikembangkan untuk membatasi melonjaknya jumlah simpul dalam pembangunan pohon penyelesaian. Berbagai jenis metode telah ditemukan untuk meningkatkan kinerja algoritma *Minimax,* salah satunya adalah *Negascout*. Dengan menggunakan metode ini maka diharapkan sistem dapat bekerja lebih baik. *Pseudocode Minimax* Dapat dilihat pada gambar 2.5 [8].

```
1   function minimax(node,depth)
2   if depth <= 0 then
3       {positive values are good for the maximizing player}
4       {negative values are good for the minimizing player}
5       return objective_value(node)
6   end
7       {maximizing player is (+1)}
8       {minimizing player is (-1)}
9   local alpha = -node.player * INFINITY
10  local child = next_child(node,nil)
11  while child ~= nil do
12      local score = minimax(child,depth-1)
13      alpha = node.player==1 and math.max(alpha,score) or
        math.min(alpha,score)
14      child = next_child(node,child)
15  end
16  return alpha
17  end
```

Gambar 2.5 Pseudocode Minimax

2.5 Algoritma Negascout

Pada tahun 1982 dan 1983 dibuat dua algoritma yang mirip, yaitu algoritma PVS dan Algoritma *Negascout.* Algoritma negascout diperkenalkan oleh Alexander Reinefeld yang menggabungkan konsep algoritma SCOUT dan algoritma *alpha-beta* dalam notasi *negamax* [3].

Negascout game tree mirip dengan SCOUT, tetapi menggunakan notasi negamax dan menggunakan beberapa optimasi. Implementasinya mengasumsikan nilai integer dari fungsi evaluasi, seperti dalam gambar 2.6. Dalam memperkecil jendela pencarian agar mendapat jendela pencarian dengan lebar nol. *Negascout*

akan memotong hampir semua cabang-cabang dari pohon, untuk membuat suatu pencarian yang sangat tepat. Tetapi sayangnya, ia akan memotong semua cabang-cabang berguna bersamaan dengan yang tidak berguna. Hal ini tidak berlaku jika algoritma dimulai dengan hasil yang benar. Satu jendela nol dapat dilihat sebagai uji coba. Uji coba dilakukan jika nilai sebenarnya sama dengan nilai yang ditebak.

```
1   function Negascout(s, α, β, depth){
2   if (depth == MAX_DEPTH) then return evaluate(s);
3   m := -∞;
4   n := β;
5   for each {successor si of s}
6       t := -Negascout(si, -n, -max(α, m), depth + 1);
7       if (t > m) then
8           if (n == β or depth >= MAX_DEPTH – 2) then m := t;
9           else m := -Negascout(si, -β, -t, depth + 1);
10          if (m ≥ β) then return m;
11      n = max(α, m) + 1;
12 return m;}
```

Gambar 2.6 Pseudocode Algoritma *Negascout*

Keterangan pseudo code:

s	=	kondisi papan
MAX_DEPTH	=	kedalaman maksimum pohon yang ditelusuri
successor s_i of s	=	lebar pohon(banyak langkah yang mungkin terjadi pada papan)
evaluate(*s*)	=	merupakan fungsi yang memberikan nilai evaluasi pada kondisi papan "*s*"

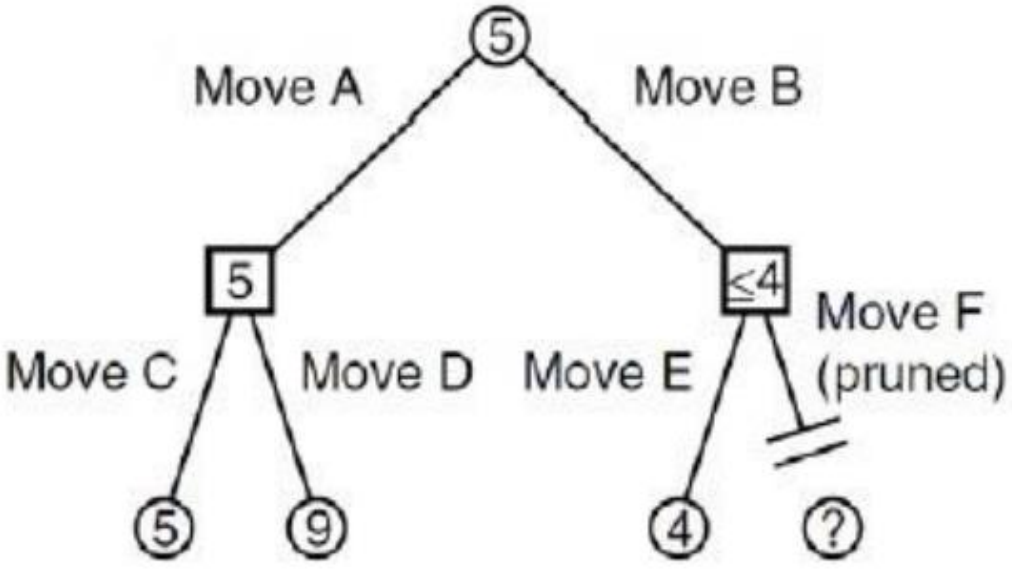
Gambar 2.6 merupakan pseudocode untuk algoritma *Negascout*, berdasarkan algoritma diatas dapat dijelaskan cara kerja algoritma *Negascout* sebagai berikut, *statement* pertama sama seperti *alpha-beta*, yaitu jika posisi s adalah node *leaf* maka *Negascout* akan mengembalikan nilai fungsi evaluasi (baris 2). Selain itu variabel *m* dan *n* diinisialisasi dengan -∞ dan beta (baris 3 dan 4).

Kemudian *Negascout* akan melakukan "*scout*" pada *node* anak dari *s* 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 11 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* tersebut dapat dipotong atau tidak [8].

Jika *null window* gagal karena $t > m$ (baris 7), *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 8), 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 9). Kondisi untuk pemotongan (baris 10) sama seperti *alpha-beta* : jika $m \geq beta$ maka semua *node* anak lain dapat diabaikan [8].

2.5.1 Pemotongan Alpha

Gambar 2.7 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 [8].



Sumber gambar : Coppin [9]

Gambar 2.7 Pohon Permainan dengan Proses Pemotongan Alpha

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 [8].

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* [8].

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 [8].

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 [8].

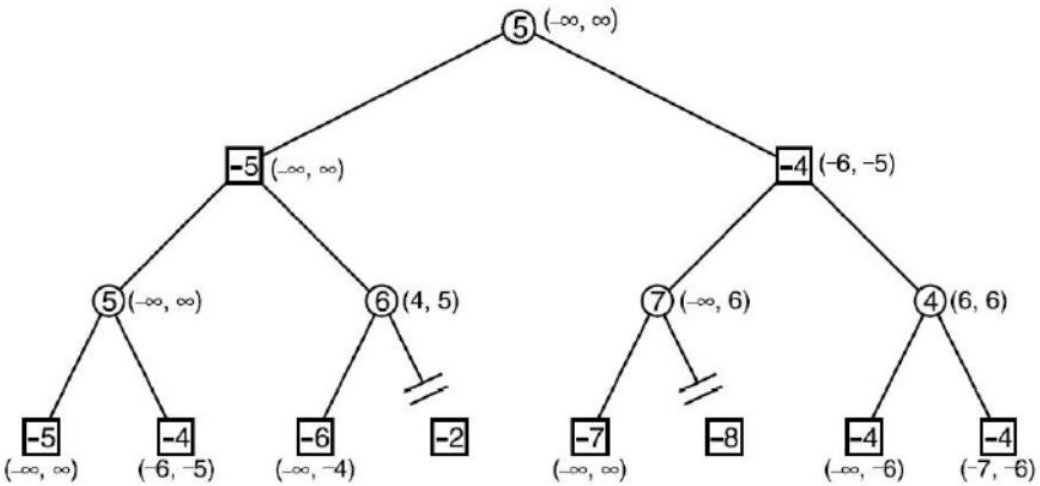
2.5.4 Zero-Width Test

Pada algoritma *Negascout* ada yang disebut dengan kondisi *fail soft*, kondisi ini menggambarkan pemotongan *alpha* dan *beta* yang telah diketahui. Pada

dasarnya pemotongan *alpha beta* hanya mengembalian baik nilai *alpha* maupun nilai *beta* sebagai nilai, jika gagal berdasarkan gagal karena nilai yang didapatkan lebih tinggi atau lebih rendah dari nilai maksimum sementara pada tingkat pencarian tertentu. Dengan kondisi *fail soft* dapat membantu untuk menemukan satu solusi. Dengan kondisi ini memungkinkan untuk menebak nilai awal dan megulangi pencarian dengan jendela yang lebih masuk akal (*Zero width test*) [9].

Pada dasarnya algoritma *Negascout* menggabungkan pencarian *Minimax* dengan pemotongan *alpha beta* dan pemanggilan *zero-width Test*. *Negascout* tidak selebar pencarian algoritma *Minimax*. *Negascout* menggunakan pemotongan *alpha* dan *beta* untuk uji coba tersebut.

Negascout bekerja dengan melakukan pemerikasaan secara menyeluruh pada langkah pertama dalam setiap posisi papan. Hal ini dilakukan pada satu jendela pencarian luas, sehingga algoritma tidak gagal. Berturut-turut langkah diuji menggunakan satu jalur berdasarkan nilai dari langkah pertama, jika berhasil, maka akan diulangi pada jendela dengan lebar sebenarnya, perhatikan gambar 2.8.



Sumber gambar : Coppin [9]

Gambar 2.8 Pohon Permainan dengan Algoritma Negascout

2.6 Pemrograman Berorientasi Objek atau Object Oriented Programming

Metodologi berorientasi objek adalah suatu strategi pembangunan perangkat lunak yang mengorganisasikan perangkat lunak sebagai kumpulan objek yang berisi data dan operasi yang diberlakukan terhadapnya. Metodologi berorientasi objek merupakan suatu cara bagaimana sistem perangkat lunak

dibangun melalui pendekatan objek secara sistematis. Metode berorientasi objek didasarkan pada penerapan prinsip-prinsip pengelolaan kompleksitas. Metode berorientasi objek meliputi rangkaian aktivitas analisis berorientasi objek, perancangan berorientasi objek, pemrograman berorientasi objek, dan pengujian berorientasi objek [10].

Pada saat ini, metode berorientasi objek banyak dipilih karena metodologi lama banyak menimbulkan masalah seperti adanya kesulitan pada saat mentransformasi hasil dari satu tahap pengembangan ke tahap berikutnya, misalnya pada metode pendekatan terstruktur jenis aplikasi yang dikembangkan saat ini berbeda dengan masa lalu. Aplikasi yang dikembangkan pada saat ini sangat beragam (aplikasi bisnis, real-time, utility, dan sebagainya) dengan platform yang berbeda-beda, sehingga membutuhkan tuntutan kebutuhan metodologi pengembangan yang dapat mengakomodasi ke semua jenis aplikasi tersebut.

Keuntungan menggunakan metodologi berorientasi objek adalah sebagai berikut :

1. Meningkatkan Produktivitas

   Karena kelas dan objek yang ditemukan dalam suatu masalah dapat dipakai ulang untuk masalah lainnya yang melibatkan objek tersebut (reusable).

2. Kecepatan Pengembangan

   Karena sistem yang dibangun dengan baik dan benar pada saat analisis dan perancangan akan menyebabkan berkurangnya kesalahan pada saat pengkodean.

3. Kemudahan Pemeliharaan

   Karena dengan model objek, pola-pola yang cenderung tetap dan stabil dapat dipisahkan dengan pola-pola yang mungkin sering berubah-ubah.

4. Adanya Konsistensi

   Karena sifat pewarisan dan penggunaan notasi yang sama pada saat analisis, perancangan maupun pengkodean.

5. Meningkatkan Kualitas Perangkat Lunak

   Karena pendekatan pengembangan lebih dekat dengan dunia nyatadan adanya konsistensi pada saat pengembangannya, perangkat lunak yang dihasilkan akan mampu memenuhi kebutuhan pemakai serta mempunyai sedikit kesalahan.

Saat ini sudah banyak bahasa pemrograman berorientasi objek. Banyak orang berfikir bahwa pemrograman berorientasi objek identik dengan bahasa java. Memang bahasa java merupakan bahasa yang paling konsisten dalam mengimplementasikan paradigma pemrograman berorientasi objek. Namun sebenarnya bahasa pemrograman yang mendukung pemrograman berorientasi objek tidak hanya bahasa java [10].

2.7 UML (Unified Modeling Language)

Pada perkembangan teknik pemrograman berorientasi objek, muncul sebuah standarisasi bahasa pemodelan untuk pembangunan perangkat lunak yang dibangun dengan menggunakan teknik pemrograman berorientasi objek, yaitu *Unified Modeling Language* (UML). UML muncul karena adanya kebutuhan pemodelan visual untuk menspesifikasikan, menggambarkan, membangun, dan dokumentasi dari sistem perangkat lunak. UML merupakan bahasa visual untuk pemodelan dan komunikasi mengenai sebuah sistem dengan menggunakan diagram dan teks-teks pendukung [10].

UML hanya berfungsi untuk melakukan pemodelan. Jadi penggunaan UML tidak terbatas pada metodologi tertentu, meskipun pada kenyataanya UML paling banyak digunakan pada metodologi berorientasi objek. Seperti banyak kita ketahui bahwa banyak hal di dunia informasi yang tidak dapat dibakukan, semua tergantung kebutuhan, lingkungan, dan konteksnya.

Begitu juga dengan perkembangan penggunaan UML bergantung pada level abstraksi penggunanya. Jadi, belum tentu pandangan yang berbeda dalam penggunaan UML adalah suatu yang salah, tapi perlu ditelaah dimanakah UML digunakan dan hal apa yang ingin divisualkan. Secara analogi jika dengan bahasa

yang kita gunakan sehari-hari, belum tentu penyampaian bahasa dengan puisi adalah hal yang salah. Sistem informasi bukanlah ilmu pasti, maka jika ada banyak perbedaan dan interpretasi di dalam bidang sistem informasi merupakan hal yang sangat wajar.

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 Diagram*

   Secara grafik menggambarkan interaksi antara sistem, hal-hal diluar sistem, dan *user*. Dengan kata lain, mereka mengambarkan siapa yang akan memakai sistem dan dengan cara itu seorang *user* akan berinteraksi dengan sistem. *Use-case* merupakan urutan langkah yang secara tindakan saling terkait (Skenario), baik termotivasi maupun secara manual, untuk tujuan melengkapi satu tugas bisnis tunggal [10].

   Notasi yang digunakan dalam *use-case diagram* antara lain :

   1. *Actor*

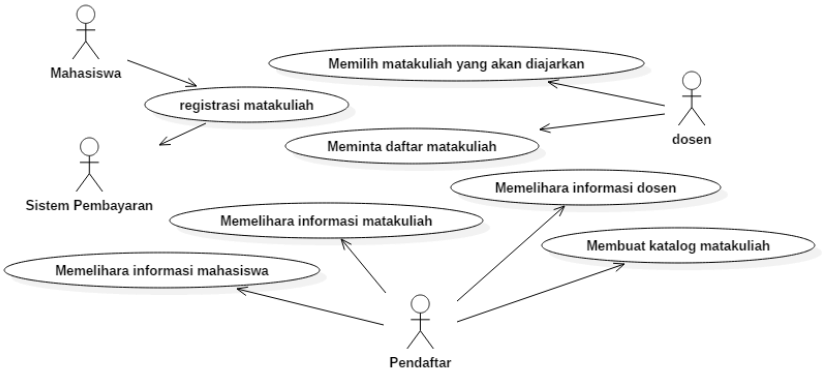
      Adalah posisi yang dimiliki user terhadap sistem. Yang menjadi *actor* dapat berupa manusia, hardware atau sistem lain yang berhubungan dengan sistem yang berjalan atau *actor* adalah sesuatu yang memerlukan interaksi dengan sistem untuk bertukar informasi.

   2. *Use – case* simbol

      Menggambarkan sekumpulan urutan, dimana setiap urutan mewakili intervensi antara *actor* dengan sistem. *Use-case* simbol mempresentasikan fungsionalitas sistem secara keseluruhan.

   Bagian umum kejadian *use-case* adalah deskripsi *use-case,* yaitu langkah demi langkah mulai dengan pelaku menginisialisasi *use-case* dan melanjutkan hingga akhir kejadian bisnis. Dimana dalam deskripsi *use-case* terdapat :

a. Nama *use-case* (*use-case*), merupakan nama dari suatu *use-case*.
b. Aktor (*actor*), pelaku yang berpartisipasi didalam suatu *use-case*.
c. Tujuan (*goal*), tujuan dari *use-case*.
d. Prakondisi (*precondition*), batasan pada keadaan sistem sebelum *use-case* dapat dieksekusi.
e. Ringkasan (*summary*), deskripsi ringkasan pendek yang berisi jumlah kalimat yang menunjukan secara garis besar tujuan *use-case* dan berbagai kegiatanya.
f. *Related use-case*, *use-case* yang saling berhubungan.
g. Langkah – langkah (*steps*), menjelaskan setiap langkah dari *use-case* dengan mengunakan dua kolom, dimana kolom sebelah kiri menunjukan aksi dari pelaku dan kolom sebelah kanan merupakan respon yang diberikan oleh sistem.



Gambar 2.9 Contoh *Use Case Diagram*

2. *Activity Diagram*

Activity diagram atau diagram aktivitas adalah sebuah diagram yang dapat digunakan untuk menggambarkan secara grafis aliran proses bisnis, langkah langkah sebuah *use-case* atau logika behavior object. Diagram ini serupa dengan *flowchart*, dimana secara grafis diagram ini menggambarkan aliran sekuensial dengan *flowchart* dimana diagram ini menggambarkan mekanisme kegiatan parallel [10].

Oleh karena itu, diagram ini sangat berguna untuk memodelkan kegiatan yang akan dilakukan saat sebuah operasi dieksekusi dan untuk

memodelkan hasil – hasil dari kegiatan ini, seperti memodelkan *event* yang menyebabkan *window* akan ditampilkan atau ditutup.

Dibawah ini merupakan simbol – simbol yang ada pada diagram aktivitas :

a. *Initial node*

Lingkaran yang diisi penuh merupakan awal mulainya dari diagram ini. *Initial node* tidak harus ada, tetapi dengan mengunakanya membuat diagramnya lebih mudah untuk dibaca.

b. *Activity*

Segi empat bersudut tumpul menggambarkan kegiatan yang perlu dilakukan bisa secara *activity,* bisa juga secara fisik, seperti *inspect Forms,* atau secara elektronik, seperti *Display Create Student Screen*.

c. *Control Flow*

Panah yang ada di diagram, menggambarkan sasaran yang mengawas kegiatan.

d. *Fork*

Bar hitam dengan satu flow yang ke dalamnya dan beberapa flow lainnya meninggalkannya, merupakan bar sinkronissi dimana kegiatan dapat dilakukan secara paralel.

e. *Join*

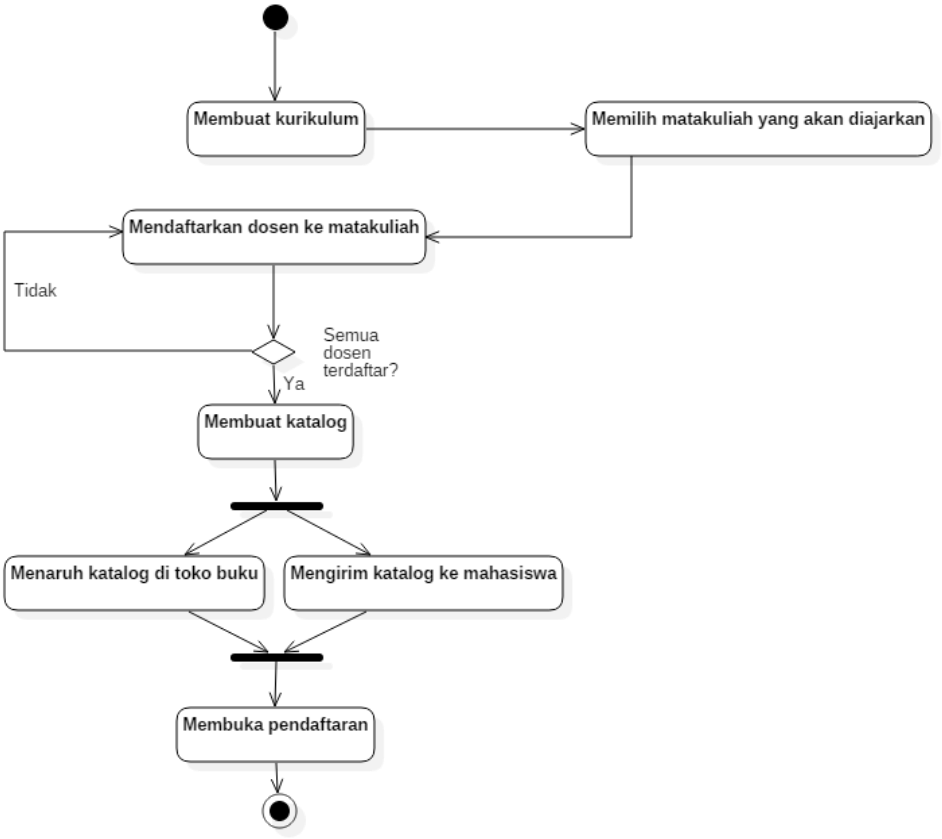
Bar hitam dengan satu *flow* mengarah ke dalamnya dan ada satu *flow* yang meninggalkannya, merupakan bar sinkronisasi dimana beberapa aktivitas yang mempunyai tujuan yang sama dari keduannya digabung menjadi satu.

f. *Decision*

Gambar sebuah belah ketupat, menggambarkan sebuah kegiatan keputusan.

g. *Final Node*

Gambar lingkaran yang diisi penuh dan ada batas lagi di luarnya, menggambarkan akhir dari sebuah proses.

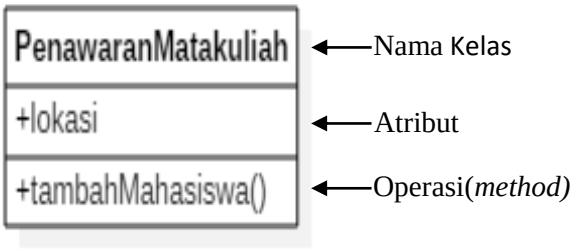


Gambar 2.10 Contoh *Activity Diagram*

3. *Class Diagram*

Class diagram merupakan salah satu diagram struktur statis yang menggambarkan struktur hubungan antar kelas. *Class diagram* digunakan untuk mensimulasikan objek-objek dalam dunia nyata ke dalam sistem yang akan dibangun. Notasi UML pada class diagram adalah sebuah persegi yang dibagi menjadi 3 area, yaitu nama kelas, atribut, dan operasi (*method*) [10].

Dalam sistem yang akan dibangun, kelas-kelas akan dihubungkan secara konseptual. Hubungan ini disebut asosiasi. *Class diagram* dapat juga menggambarkan keanekaragaman (*multiplicity*), yaitu jumlah objek dari suatu kelas yang berhubungan dengan sebuah objek dari kelas yang berasosiasi.



Gambar 2.11 Contoh Notasi UML untuk *Class Diagram*

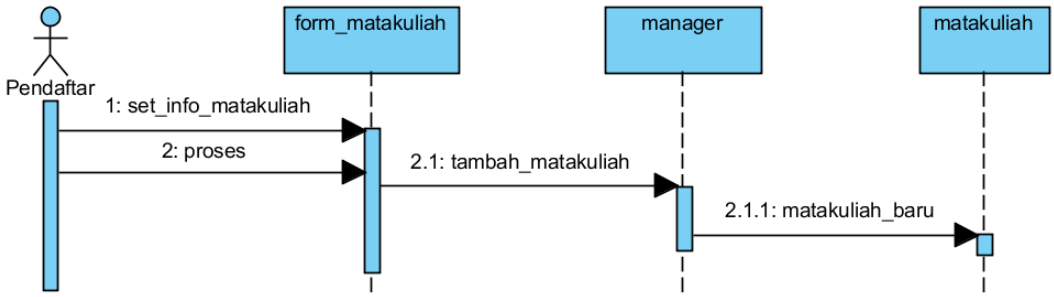
4. *Sequence Diagram*

Sequance diagram digunakan terutama untuk menunjukan interaksi antar objek dalam urutan sekuensial. *Sequance diagram* sangat berguna untuk mengkomunikasikan bagaimana objek-objek berinteraksi dalam suatu proses bisnis. Analisis sistem umumnya mengunakan *sequance diagram* untuk memperjelas *use case*.

Sequance diagram terdiri dari objek-objek yang digambarkan dengan sebuah persegi yang memiliki nama. Objek-objek tersebut diletakan diatas garis hidup tersebut, ada kotak kecil memanjang yang dinamakan aktivasi. Aktivasi mempresentasikan eksekusi dari operasi yang objek lakukan. Suatu objek dapat bertukar pesan dengan objek lain. Pesan tersebut digambarkan sebagai sebuah panah dari garis hidup suatu objek ke objek lain.

Ada empat jenis pesan yang bisa digunakan, yakni pesan sederhana (*simple message*), pesan sinkron (*synchronous message*), pesan asinkron (*asynchronous message*), dan pesan nilai pengembalian (*return value*). Pesan sederhana digunakan untuk memindahkan kontrol dari satu objek ke objek lain. Pesan sinkron digunakan apabila suatu objek harus menunggu jawaban dari objek lain (yang dipanggil) terhadap pesan tersebut sebelum melanjutkan proses lainnya.

Pesan asinkron digunakan apabila suatu objek tidak perlu menunggu jawaban dari objek lain sebelum melanjutkan proses lainnya. Sequance diagram merepresentasikan waktu dan arah vertikal, dari atas ke bawah. Pesan yang lebih atas menandakan bahwa pesan tersebut terjadi terlebih dahulu [10].



Gambar 2.12 Contoh *Sequence Diagram*

2.8 Metode Pengujian Sistem

Metode pengujian sistem untuk mengetahui efektifitas dari *software* yang digunakan selain memberikan kesempatan kepada pengguna untuk mengoperasikan dan melakukan pengecekan terhadap laporan yang dihasilkan melalui *software*. Metode pengujian sistem terdiri dari Pengujian alpha yang menerapkan pengujian *Whitebox*, Pengujian *Blackbox* dan pengujian beta [4].

2.8.1 Skenario pengujian alpha

Skenario pengujian alpha menjelaskan pengujian terhadap sistem yang dibangun, dimana dilakukan pengujian white box dan pengujian black box. Metode yang digunakan dalam pengujian white box ini adalah metode Basis Path. Metode Basis Path mengijinkan pendesain kasus uji untuk membuat perkiraan lojik yang kompleks dari desain prosedural dan menggunakan perkiraan ini untuk mendefinisikan aliran eksekusi.

2.8.1.1 Whitebox

Pengujian whitebox dilakukan untuk menguji prosedur-prosedur yang ada. Lintasan lojik yang dilalui oleh setiap bagian prosedur diuji dengan memberikan kondisi/loop spesifik. Pengujian whitebox menjamin pengujian terhadap semua lintasan yang tidak bergantungan minimal satu kali, mencoba semua keputusan lojik dari sisi “true” dan “false”, eksekusi semua loop dalam batasan kondisi dan batasan operasionalnya dan pengujian validasi data internal. Konsep Pengujian Basis Path sebagai berikut :

1. Merupakan bagian dari pengujian *whitebox* dalam hal pengujian prosedur - prosedur.

2. Mempergunakan notasi aliran *graph* (*node*, *link* untuk merepresentasikan *sequence*, *if, while, until* dan sebagainya).
3. Konsep kompleksitas *cyclomatic* antara lain cara perhitungan daerah tertutup pada graph planar dimana dapat menghubungkan batas atas jumlah pengujian yang harus direncanakan dan dieksekusi untuk menjamin pengujian seluruh *statement* program.
4. Memunculkan kasus-kasus yang akan diuji dengan membuat daftar lintasan kasus pengujian berdasarkan kompleksitas dan *cyclomatic* yang didapat.
5. Membuat alat bantu graph matriks yang membantu pengawasan pengujian.

Untuk menentukan banyaknya jalur independen pada *Basis Path* dapat digunakan *cyclomatic complexity*. Jalur independen adalah jalur yang melalui program yang menghasilkan sedikitnya satu rangkaian statemen proses baru atau suatu kondisi baru. Untuk mencari *cyclomatic complexity*, menggunakan rumus sebagai berikut:

$$\mathrm{V(G)} = edge - node + 2$$

Dimana: $edge$: Jumlah baris pada program

node: Jumlah baris pada diagram alir

Dan untuk mencari *Predicate note* suatu *graph matriks* menggunakan rumus sebagai berikut:

$$\mathrm{V(G)} = Jumlah\ Graph\ matriks + 1$$

2.8.1.2 Pengujian *Blackbox*

Black Box Testing adalah tipe pengujian yang memperlakukan perangkat lunak yang tidak diketahui kinerjanya pada bagian *internal* [11]. Para *tester* memandang perangkat lunak sebagai "kotak hitam" yang tidak penting di lihat isinya, tapi cukup dikenali proses *testing* di bagian luar, dimana *testing* ini banyak berkonsentrasi pada bagaimana proses berjalannya perangkat lunak. Pada *black box testing* atau pengujian fungsional, kondisi pengujian dikembangkan berdasarkan fungsionalitas dari program atau sistem yang akan diuji, oleh karena itu, penguji membutuhkan informasi mengenai data input dan output yang diamati, tetapi tidak

mengetahui bagaimana program atau sistem tersebut bekerja atau tidak perlu mengetahui bagaimana struktur internal dari program tersebut melakukan eksekusi.

Penguji berfokus pada pengujian fungsionalitas dari program terhadap spesifikasi, juga memfokuskan pada *requirement* fungsi dari perangkat lunak, pengujian ini merupakan komplenetari dari pengujian *whitebox*. Pengujian *whitebox* dilakukan terlebih dahulu pada proses pengujian, sedangkan pengujian *blackbox* dilakukan pada tahap akhir dari pengujian perangkat lunak.

Proses yang terdapat dalam proses pengujian *blackbox* antara lain sebagai berikut:

1. Pembagian kelas data untuk pengujian setiap kasus yang muncul pada pengujian *whitebox*.
2. Analisis batasan nilai yang berlaku untuk setiap data.

2.8.2 Pengujian Beta

Pengujian beta merupakan, pengujian perangkat lunak yang dilakukan oleh *user* dalam lingkungan yang sebenarnya. Pengembangan biasanya tidak ada pada pengujian ini. *User* merekam semua masalah (*real* atau *imajiner*) yang ditemukan selama pengujian dan melaporkan pada pengembang pada interval waktu tertentu.

2.9 Penarikan kesimpulan

Penarikan kesimpulan merupakan tahapan terakhir dari proses pengujian sistem. Pada tahapan ini, hasil dari pengujian sistem akan diambil dan ditarik kesimpulan yang menjadi tolak ukur dari penelitian ini.