BAB II

LANDASAN TEORI

Didalam bab ini akan dibahas mengenai teori-teori yang akan digunakan untuk membantu dalam perancangan serta implementasi dari permainan Halma yang meliputi pengertian game, kecerdasan buatan, rekayasa perangkat lunak, pendekatan pembangunan perangkat lunak, pengembangan perangkat lunak, *Data Flow Diagram* (DFD), perancangan antar muka, tinjauan perangkat lunak.

2.1 Pengertian Game

Game adalah suatu sistem yang memiliki aturan dengan suatu variabel dan hasil yang dapat dihitung, di mana hasil yang berbeda ditugaskan dengan nilai-nilai yang berbeda, dimana pemain melakukan usaha dalam mempengaruhi hasil perhitungan tersebut.

Game secara umum dibagi menjadi lima bagian diantaranya:

1. Arcade/ Action, game yang memiliki interaksi dengan manusia yang tinggi dan menggunakan graphis yang beragam, seperti: game Quake dan Freelancer.
2. Puzzle, game yang didalamnya terdapat unsur logika, seperti : game Tetris.
3. Card, game seperti Poker dan BlackJack.

4. Strategy, game yang memiliki suatu teknik dan pemikiran serta manajemen didalamnya seperti : game Command and Conquer dan Warcraft.
5. Sport, game seperti permainan olah raga yang sesungguhnya seperti game FIFA 2000.

Dalam game terdapat banyak unsur grafika komputer. Grafika komputer adalah seperangkat alat yang terdiri dari hardware dan software untuk membuat gambar, grafik atau citra realistik untuk seni, game komputer, foto dan film animasi.

Grafika tiga dimensi dapat menggambarkan sebuah benda dalam dunia nyata menjadi sebuah objek dari berbagai sudut pandang. Tidak seperti grafika dua dimensi, dalam grafika tiga dimensi banyak sekali hal yang harus dipertimbangkan.

2.2 Rekayasa Perangkat Lunak

Perangkat lunak merupakan suatu program yang berisi sejumlah instruksi yang dibuat untuk melakukan sejumlah proses dengan tujuan sebagai alat pengendalian sistem komputer atau untuk menyelesaikan aplikasi tertentu.

Dalam membangun suatu perangkat lunak, selalu timbul beberapa kesulitan baik dari lingkungan intern maupun lingkungan ekstern. Untuk mengatasi kesulitan tersebut disusunlah suatu tata cara atau metodologi yang terangkum dalam Rekayasa Perangkat Lunak.

Rekayasa Perangkat Lunak adalah suatu disiplin ilmu untuk membangun perangkat lunak, yang menggabungkan metode-metode untuk seluruh tahap-tahap pengembangan perangkat lunak, penentuan alat bantu yang baik untuk mengotomatisasikan metode-metode ini, pembuatan model perangkat lunak yang sangat berpengaruh untuk penerapan perangkat lunak, serta teknik-teknik yang baik untuk menjamin kualitas perangkat lunak. Untuk menghasilkan perangkat lunak yang baik maka :

1. Perangkat lunak yang dibuat harus mempunyai sifat yang mudah diperbaiki atau dikembangkan, sehingga dapat terus disesuaikan dengan perkembangan, baik dari teknologi maupun kebutuhan pengguna.
2. Perangkat lunak harus dapat diandalkan, yang mana kemampuan dari perangkat lunak diharapkan bekerja tanpa kesalahan.
3. Perangkat lunak harus bersifat efisien.

Perangkat lunak harus memberikan antarmuka yang tepat bagi pengguna.

2.3 Kecerdasan Buatan

Kecerdasan adalah sesuatu yang hanya dimiliki oleh manusia dan tidak dimiliki oleh benda mati lainnya termasuk komputer. Ini adalah suatu bentuk anugerah yang tak terhingga dari Sang Maha Pencipta. Tetapi bukan berarti kecerdasan yang dimiliki manusia tidak bisa diadopsi pada yang lain (dalam hal ini komputer), meskipun hanya sedikit meniru. Itulah sebabnya kecerdasan yang

dimiliki komputer seringkali dinamakan dengan kecerdasan buatan *(artificial intelligence).*

Kecerdasan komputer adalah suatu bentuk adopsi kecerdasan manusia yang dilakukan oleh komputer. Dengan menggunakan kecerdasan komputer itu dimungkinkan sebuah komputer mampu berpikir seperti layaknya manusia. Dalam perkembangannya sangatlah sulit menyamai kecerdasan manusia, tetapi setidak-tidaknya komputer mampu melakukan sesuatu dengan cara yang bagi manusia adalah cara yang cerdas.

2.3.1 Permainan Dengan Kecerdasan Buatan

Ada dua alasan mengapa permainan muncul menjadi hal yang baik untuk menyelidiki kecerdasan sebuah mesin (komputer).

a. Permainan menyediakan struktur perintah-perintah yang sangat mudah untuk mengetahui langkah yang benar atau salah.

b. Permainan tidak terlalu membutuhkan kuantitas pengetahuan yang nyata. Permainan hanya memikirkan bagaimana untuk mencari jawaban dengan mencari secara terus menerus dari posisi awal sampai posisi kemenangan.

Dari dua alasan diatas yang pertama cenderung benar dan nilai untuk meneruskan minat pada area permainan yang dilakukan oleh mesin. Tapi pada bagian kedua tidak selamanya benar untuk sebagian permainan yang sangat sederhana.

Pada program yang sederhana tidak mencari jawaban dengan mencari secara terus menerus pada rangkaian permainan untuk melakukan langkah pertama. Tapi untuk sebagian program memerlukannya. Untuk menambah efektifitas proses pencarian memecahkan masalah ada dua hal yang dapat dilakukan, yaitu :

a. Tambahkan prosedur pembangkit langkah jadi hanya langkah yang baik yang dibangkitkan.
b. Tambahkan prosedur pengujian sehingga langkah terbaik akan diketahui dan dilacak terlebih dahulu.

Tetapi tentu saja proses pencarian bukan satu-satunya teknik. Pada beberapa permainan setidaknya ada saat ketika lebih memerlukan teknik langsung secara tepat.

Cara yang tepat menggunakan prosedur pencarian adalah mencari solusi masalah untuk membangkitkan langkah sampai tujuan akhir terpenuhi. Konteks program permainan, tujuan akhir adalah untuk memenangkan permainan.

2.3.2 Metode Pencarian

Hal terpenting dalam menentukan keberhasilan sistem yang berdasarkan AI adalah kesuksesan dalam pencarian dan pencocokan. Pada dasarnya, terdapat 2 (dua) teknik pencarian dan pelacakan yang digunakan, yaitu:

1. Pencarian buta (blind search), dapat dibagi menjadi:

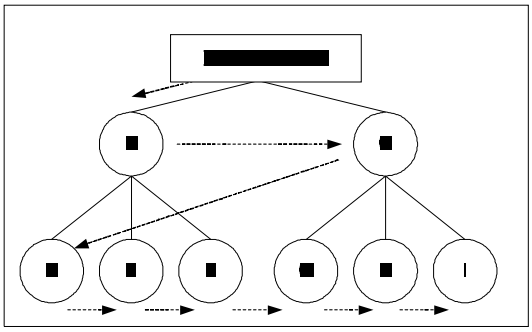
a. Pencarian Melebar Pertama (*Breadth-First Search*)

b. Pencarian Mendalam Pertama (*Depth-First Search*)

2. Pencarian Terbimbing (*Heuristic Search*)

2.3.2.1 *Breadth-First Search* (BFS)

Pada metode ini, semua *node* pada level *n* akan dikunjungi terlebih dahulu sebelum mengunjungi *node-node* pada level *n*+1 (level berikutnya). Pencarian dimulai dari *node* akar terus ke level ke-1 dari kiri ke kanan, kemudian berpindah ke level-2 dan seterusnya. Pohon dikembangkan hingga ditemukannya solusi.



Gambar 2.6 Prosedur Pencarian Melebar Pertama

Keuntungan dari metode ini adalah tidak akan menemui jalan buntu. Jika terdapat lebih dari satu solusi, maka *breadth-first search* akan menemukannya. Dan jika ada lebih dari satu solusi, maka solusi minimum akan ditemukan.

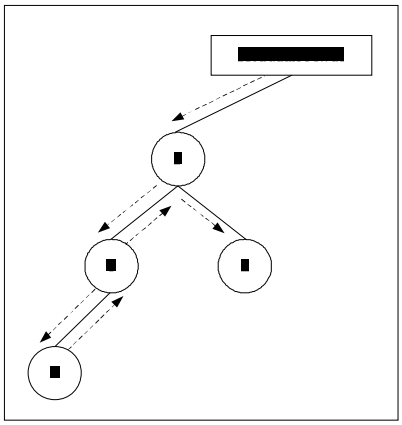
Sedangkan kelemahan dari metode ini adalah membutuhkan memori yang cukup banyak, karena menyimpan semua *node* dalam satu pohon dan membutuhkan waktu yang cukup lama, karena akan menguji semua node pada level-*n* untuk mendapatkan solusi pada level ke- (*n* + 1).

2.3.2.2 *Depth-First Search* (DFS)

Pencarian mendalam pertama merupakan teknik penelusuran data pada *node-node* secara vertikal dan sudah terdefinisikan secara mendalam. Proses pemeriksaan akan bergerak turun jika *node* yang diperiksa saat ini tidak sesuai dengan tujuan. *Node* tidak akan dikembangkan ke samping walaupun *node* masih mempunyai beberapa *node* anak yang dapat dikembangkan. *Node* anak baru dikembangkan lagi setelah selesainya pencarian mendalam terhadap *node* anak yang sebelumnya telah dikembangkan. Proses ini diulangi terus hingga didapatkan solusi. Perhatikan gambar 2.7 sebagai prosedur pencarian mendalam pertama.

Keuntungan dari metode ini adalah membutuhkan memori yang relatif kecil, karena hanya *node-node* pada lintasan yang aktif saja yang disimpan. Secara kebetulan, metode *depth-first search* akan menemukan solusi tanpa harus menguji lebih banyak lagi dalam ruang keadaan.

Sedangkan kelemahan dari metode ini adalah memungkinkan tidak ditemukannya tujuan yang diharapkan dan hanya akan mendapatkan 1 solusi pada setiap pencarian.



Gambar 2.7 Prosedur Pencarian Mendalam Pertama

Pada gambar 2.7, pencarian dimulai dari keadaan awal kemudian dilanjutkan dengan mengembangkan *node* B, *node* C dan *node* D. Ketika diketahui *node* D bukan merupakan solusi dan tidak dapat mengembangkan node anak lagi, pencarian dikembalikan ke *node parent*, dan mengecek apakah ada *node* anak yang dapat dikembangkan lagi. Apabila ada, maka pencarian dilakukan ke level bawah lagi. Pencarian dilakukan terus hingga ditemukan solusi.

2.3.2.3 Pencarian Heuristik (Heuristic Search)

Heuristic search adalah suatu istilah yang berasal dari bahasa Yunani yang berarti menemukan / menyingkap. *Heuristic* merupakan suatu perbuatan yang membantu kita menemukan jalan dalam pohon pelacakan yang menuntut kita kepada suatu solusi masalah. *Heuristic* dapat diartikan juga sebagai suatu kaidah yang merupakan metoda / prosedur yang didasarkan kepada pengalaman dan praktek, syarat, trik atau bantuan lainnya yang membantu mempersempit dan memfokuskan proses pelacakan kepada suatu tujuan tertentu.

Heuristic dapat digunakan pada beberapa kondisi berikut ini :

1. Mengatasi *combinatorial explosion*.

   Ada masalah yang kemungkinan arah penyelesaiannya berkembang pesat (bersifat faktorial) sehingga menimbulkan *combinatorial explosion*. *Heuristic* merupakan cara untuk menentukan kemungkinan arah penyelesaian masalah secara efisien.

2. Solusi paling optimal mungkin tidak diperlukan.

Dalam suatu keadaan, mungkin lebih baik mendapatkan solusi yang mendekati optimal dalam waktu yang singkat daripada solusi yang paling optimal dalam waktu yang lama.

3. Pada umumnya hasilnya cukup baik.

   Sekalipun tidak optimal biasanya mendekati optimal.

4. Membantu pemahaman bagi orang yang menyelesaikan persoalan.

Banyak alternatif *heuristic* yang dapat diterapkan dalam suatu percobaan. Orang yang menyelesaikan persoalan tersebut akan lebih mengerti persoalannya jika mencoba *heuristic* yang diterapkannya.

Pencarian buta (*uninformed* atau *blind search*) umumnya kurang efisien. Hal ini disebabkan oleh waktu akses yang cukup lama dan besarnya memori yang diperlukan. Untuk mengatasi kelemahan ini maka penambahan domain pengetahuan akan menghasilkan suatu proses pencarian dan investigasi baru. Pencarian tersebut biasanya diistilahkan dengan *informed search*, pencarian heuristik atau pencarian yang berdasarkan panduan.

George Poyla mendefinisikan heuristik sebagai studi metode dan kaidah penemuan. Dalam pencarian ruang keadaan, heuristik dinyatakan sebagai aturan untuk melakukan pemilihan cabang-cabang dalam ruang keadaan yang paling tepat untuk mencapai solusi permasalahan yang dapat diterima.

Pemecahan masalah dalam AI menggunakan heuristik dalam dua situasi dasar, yaitu:

1. Permasalahan yang mungkin tidak mempunyai solusi yang pasti disebabkan oleh ambiguitas (keraguan / ketidakpastian) mendasar dalam pernyataan permasalahan atau data yang tersedia. Diagnosa kedokteran merupakan salah satu contohnya, dimana sejumlah gejala mungkin dapat ditimbulkan oleh berbagai macam penyebab yang mungkin. Dokter menggunakan heuristik untuk memilih atau menentukan diagnosa yang paling dapat diharapkan dan memutuskan rencana penanganannya. *Vision* merupakan masalah lainnya untuk permasalahan ketidakpastian yang mendasar.
2. Permasalahan yang boleh jadi memiliki solusi pasti, tetapi biaya komputasinya untuk mendapatkan solusi tersebut mungkin sangat tinggi. Dalam banyak masalah seperti catur, ruang keadaan bertambah secara luar biasa seiring dengan pertambahan jumlah keadaan yang dimungkinkan. Dalam kasus ini, teknik pencarian yang mendalam seperti prosedur DFS dan BFS mungkin gagal memperoleh solusi sehingga heuristik akan menangani kerumitan permasalahan ini dengan panduan pencarian pada sepanjang lintasan yang memberi harapan melewati *state*. Dengan mengeliminasi *state* yang tidak memberi harapan dan turunannya dari ruang tersebut maka algoritma heuristik dapat menemukan solusi yang dapat diterima dengan lebih cepat.

Sayangnya, seperti semua kaidah penemuan lainnya, heuristik juga dapat salah. Heuristik hanyalah panduan informasi untuk menebak langkah berikutnya yang harus diambil dalam menyelesaikan suatu permasalahan, dan sering dilakukan berdasarkan eksperimen/percobaan atau secara intuisi. Oleh karena, menggunakan informasi yang terbatas, heuristik jarang dapat memprediksi

tingkah laku yang eksak dari ruang keadaan saat dilakukan pencarian. Heuristik dapat membimbing algoritma pencarian untuk mendapatkan solusi suboptimal atau gagal menemukan solusi apapun.

Heuristik dan perancangan algoritma untuk mengimplementasikan pencarian heuristik telah menjadi inti permasalahan penelitian I. *Game playing* dan pemecahan teorema (*tehorem solving*) adalah dua aplikasi paling tua dari AI, kedua-duanya memerlukan heuristik untuk memangkas ruang dari solusi yang mungkin.

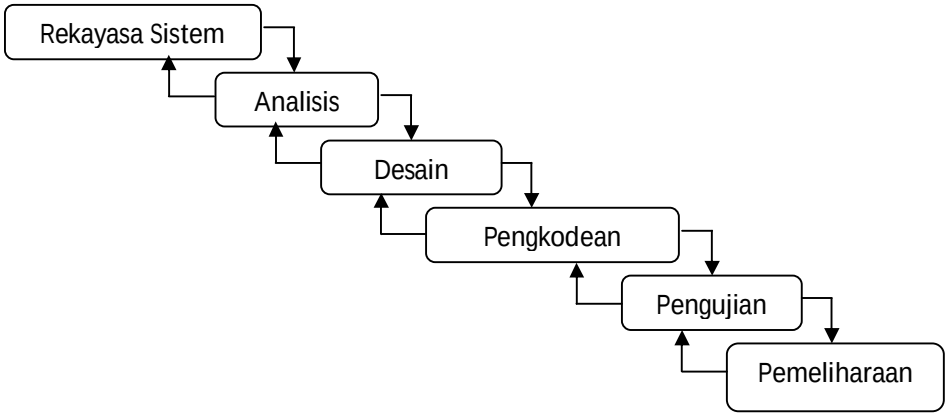
Saat ini, penelitian mengenai sistem pakar telah menyadari arti penting heuristik sebagai komponen yang esensial dalam pemecahan permasalahan. Bila seorang pakar memecahkan suatu permasalahan maka dia akan menguji informasi yang tersedia dan membuat suatu keputusan.

2.4 Pendekatan Pembangunan Perangkat Lunak

Metodologi pembangunan perangkat lunak dikembangkan dan diterapkan agar dapat menghasilkan suatu perangkat lunak yang baik dan sesuai dengan kebutuhan pengguna.

Metodologi yang dipakai dalam pembangunan perangkat lunak Lelempengan adalah pendekatan Siklus Kerja Sederhana (*Classic life Cycle*) yang biasa disebut juga model *Waterfall* [bab 1], siklus ini bersifat sistematik, dengan pendekatan sekuensial untuk membangun perangkat lunak, dimulai pada tingkat

sistem dan berkembang sampai dengan pengujian (*testing*) dan perbaikan kesalahan (*maintenance*).



Gambar 2.1

Siklus kerja Rekayasa Perangkat Lunak dengan Metode *Waterfall*

Keterangan mengenai tiap-tiap langkah pembuatan perangkat lunak adalah sebagai berikut :

1. Rekayasa Sistem

   Karena perangkat lunak selalu merupakan bagian dari sistem, maka harus diawali dengan menentukan kebutuhan-kebutuhan bagi seluruh elemen-elemen sistem, dan kemudian mengalokasikan beberapa subset dari kebutuhan-kebutuhan tersebut bagi perangkat lunak. Rekayasa sistem mencakup kumpulan kebutuhan-kebutuhan pada setiap tingkat sistem dengan jumlah yang kecil pada tingkat teratas perancangan dan analisis.

2. Analisis

   Merupakan proses pengumpulan kebutuhan perangkat lunak. Untuk memahami program yang akan dibangun, harus dipahami wawasan informasi dari perangkat lunak, fungsi yang dibutuhkan, performansi dan antarmuka. Beberapa target yang ingin dicapai antara lain :

   1. Mengetahui keinginan pengguna
   2. Efisiensi biaya, sumber daya dan waktu pembangunan perangkat lunak
   3. Ketersediaan dukungan teknik

3. Desain

   Perancangan perangkat lunak sebenarnya merupakan kumpulan proses yang difokuskan pada 4 (empat) atribut yang berbeda, yaitu struktur data, arsitektur perangkat lunak, rincian prosedur dan karakteristik antarmuka.

   Tahap ini merupakan tahap penerjemahan dari keperluan atau data yang telah dianalisa ke dalam bentuk yang mudah dimengerti oleh pemakai.

   Beberapa target dari tahap ini adalah :

   1. Spesifikasi perangkat lunak harus dapat dimengerti oleh pembangun perangkat lunak.
   2. Perancangan modul-modul memungkinkan digunakan kembali baik secara parsial maupun total.
   3. Pembentukan dokumentasi perancangan perangkat lunak.

4. Pengkodean

   Pada tahap ini perancangan harus diterjemahkan kedalam bentuk yang dapat dibaca oleh mesin. Yang menjadi target dari tahap ini adalah menuliskan program secara rinci pada setiap modul.

5. Pengujian

   Proses pengujian difokuskan pada *logical internal* dari perangkat lunak untuk memastikan bahwa semua pernyataan-pernyataan telah diuji, dan pada *functional external* untuk menemukan kesalahan-kesalahan dan memastikan bahwa masukan sesuai dengan kebutuhan.

   1. Yang menjadi target pada tahap ini adalah :
   2. Mendeteksi setiap kesalahan yang terjadi
   3. Memperbaiki setiap kesalahan yang terjadi
   4. Mencegah kesalahan baru selama perangkat lunak dimodifikasi
   5. Perangkat lunak dapat dipercaya/diandalkan.

6. Pemeliharaan

   Merupakan tahap perbaikan atau penyempurnaan perangkat lunak yang telah selesai akibat perubahan-perubahan atau penambahan yang terjadi sesuai kebutuhan atau permintaan pemakai. Untuk melakukannya, dipergunakan kembali setiap langkah pada siklus kerja terdahulu.

2.5 Pengembangan Perangkat Lunak

Secara umum, pengembangan perangkat lunak [MAH 96] terdiri dari 3 (tiga) tahap, yaitu :

1. Definisi

   Pada tahap ini dirumuskan apa yang akan diproses, fungsi dan performansi yang diinginkan, interface yang akan digunakan, serta prediksi kendala-kendala, batasan-batasan dan kriteria validasi yang diperlukan untuk menunjang keberhasilan sistem.

2. Pengembangan

   Pada tahap ini dijabarkan mengenai struktur data dan arsitektur perangkat lunak, prosedur rinci untuk penerapan, proses penerjemahan ke dalam pemrograman, serta perlakuan uji coba.

3. Perubahan

   Ada 3 (tiga) jenis perubahan atas perangkat lunak, yaitu :

   1. Perbaikan, yakni perubahan karena kekurangan.
   2. Adaptasi, yaitu perubahan yang mengacu terhadap lingkungan.
   3. Perluasan, yaitu perubahan karena permintaan pemakai.

2.6 *Data Flow Diagram (DFD)*

Data Flow Diagram (DFD) adalah teknik grafik yang digunakan untuk menjelaskan aliran informasi dan transformasi data yang bergerak dari masukan ke keluaran. DFD merupakan alat yang digunakan pada metodologi pengembangan sistem yang terstruktur. Teknik ini sering digunakan untuk menggambarkan suatu sistem yang telah ada atau sistem baru yang akan dikembangkan secara logika tanpa mempertimbangkan lingkungan fisik dimana data tersebut mengalir atau akan disimpan.

DFD menjelaskan bagaimana data masukan diubah menjadi keluaran, dimana setiap bagian pada diagram menjelaskan proses transformasi yang berbeda. DFD dapat digunakan untuk menggambarkan sistem atau perangkat lunak pada banyak tingkatan pada dari suatu abstraksi. DFD juga dapat digunakan untuk menggambarkan sistem atau perangkat lunak pada setiap tingkat dan dapat dipecah-pecah.

DFD dapat dibagi ke dalam tingkatan-tingkatan yang menggambarkan bertambahnya aliran data dan rincian fungsional. Level 1 pada DFD menyatakan prosedur-prosedur dari keseluruhan sistem yang digambarkan pada model konteks (*Context Model*).

Model konteks pada DFD merupakan model sistem pokok yang menggambarkan keseluruhan seluruh elemen perangkat lunak sebagai sebuah lingkaran dengan data masukan dan data keluaran dinyatakan dengan anak panah masuk atau keluar, berturut-turut. Model konteks menjelaskan bagaimana

gambaran sistem dan hubungan dengan pihak luar. Setiap proses dan jalur-jalur aliran data digambarkan pada tingkat yang lebih tinggi (misalnya level 2) yang menyatakan proses-proses dari keseluruhan prosedur yang digambarkan pada DFD level 1.

Untuk membuat DFD, digunakan simbol-simbol. Simbol-simbol ini dirancang untuk mempermudah penggambaran kepada pemakai. Berikut ini adalah simbol-simbol menurut *Yourdan–DeMarco.*

Persegi empat : Menyatakan entitas luar (*external entity*), yaitu sebuah sistem atau sistem lain yang menerima / menghasilkan informasi dari / bagi perangkat lunak.

Lingkaran : Menyatakan proses yang menggunakan dan mengubahnya informasi ke dalam bentuk lain.

Anak panah : Menyatakan satu atau banyak item data. Seluruh anak panah harus diberi nama.

Garis ganda : Menyatakan tempat penyimpanan data (*data storage*), yang menyimpan isi dari aliran data yang digunakan oleh sistem atau perangkat lunak.

2.7 Perancangan Antar Muka

Antarmuka suatu perangkat lunak menggambarkan bagaimana cara sebuah program akan dilihat dan digambarkan oleh pemakai. Perancangan sebuah antarmuka meliputi perumusan tampilan layar dan struktur menu.

Secara umum, fungsi antar muka bagi pemakai adalah sebagai berikut :

1. Penyampaian pesan dari pemakai ke sistem dan sebaliknya.
2. Validasi oleh sistem terhadap pesan dari pemakai dan pengiriman pesan kesalahan dari sistem ke pemakai.

Beberapa pertimbangan dari pedoman dalam melakukan perancangan antar muka, antara lain :

1. Sederhana
2. Konsisten
3. Membatasi jumlah parameter dan perintah
4. Menyediakan nilai dasar (default value)
5. Memberikan kemudahan penanganan kesalahan
6. Dapat membatalkan suatu aksi
7. Dukungan kontrol oleh pemakai
8. Memiliki fasilitas pembatalan (undo)

Berikut ini adalah penjelasan dari beberapa komponen antar muka yang penting dan harus dirancang :

2.7.1 Perancangan Menu

Beberapa pedoman perancangan menu adalah sebagai berikut :

1. Konsisten
2. Terorganisir

3. Terurut
4. Dapat kembali ke menu awal
5. Tersedia alternatif cara pemilihan item
6. Mempertimbangkan luas layar
7. Mempertimbangkan visual menu dalam hal warna, bentuk maupun letaknya
8. Memberikan keterangan tentang masukan
9. Menyediakan bantuan langsung
10. Mempertimbangkan waktu tanggap dan kecepatan tampilan

2.7.2 Perancangan Pengisian Masukan

1. Penggunaan istilah yang konsisten.
2. Memperhatikan frekuensi pemakaian.

2.7.3 Perancangan Fasilitas Bantuan Langsung Dan Pesan Kesalahan

Beberapa pedoman dalam perancangan fasilitas bantuan adalah sebagai berikut.

1. Singkat, jelas, konsisten dan mudah dipahami.
2. Pemanggilan bantuan dapat dilakukan dari semua titik pada sistem setiap saat.
3. Penayangan bantuan tidak mengganggu pekerjaan.

Beberapa pedoman dalam perancangan pesan kesalahan adalah sebagai berikut.

1. Dimunculkan jika pemakai membuat kesalahan saat interaksi.
2. Menjelaskan tentang jenis kesalahan dan cara mengatasinya.
3. Pesan tidak terlalu umum.
4. Lokasi dalam penayangannya harus konsisten.

2.8 Tinjauan Perangkat Lunak

2.8.1 ASP.net

ASP.NET adalah teknologi baru dalam pemrograman web yang merupakan kelanjutan dari teknologi ASP 3.0. Perbedaan utama disbanding ASP klasik adalah penggunaan .NET Framework sebagai fondasi pemrograman.

ASP.NET memiliki beberapa kelebihan dibandingkan teknologi terdahulu, antara lain:

1. Kemudahan mengakses berbagai library .NET Framework secara konsisten dan powerful, yang mempercepat pengembangan aplikasi.
2. Penggunaan berbagai bahasa pemrograman secara penuh, misalnya VB.NET, C#, J#, dan C++. Dalam ASP.NET bahasa-bahasa ini dapat digunakan secara penuh sebagaimana layaknya bekerja di Windows Application. Hal ini jauh berbeda dibandingkan ASP klasik yang mengunakan scripting language. Misalnya VBScript agak berbeda syntaxnya dengan VB, dalam ASP.NET kita dapat menggunakan

VB.NET dengan syntax yang 100% sama dengan ketika membangun Windows Application. Demikian juga dengan C#, tidak ada perbedaan syntax antara C# untuk membuat kode ASP.NET dengan Windows Form. Ini sangat memudahkan memprogram aplikasi Windows dan Web secara konsisten.

3. Tersedia berbagai Web Control yang dapat digunakan membangun aplikasi secara cepat.Programmer dapat dengan mudah mengkaitkan data ke web control sebagaimana layaknyamemprogram windows application. Hal ini sangat mempercepat pembuatan aplikasi dibandingkan harus menyusun kode-kode HTML secara manual.
4. Code Behind, artinya kode-kode pemrograman yang menjadi logic aplikasi ditempatkan terpisah dengan kode user interface yang berbentuk HTML. Ini sangat memudahkan dalam debugging, karena kode untuk presentation layer tidak tercampur dengan kode application logic

2.8.2 Visual C#.net

2.8.2.1 Sejarah C#.net

Pada akhir dekade 1990-an, Microsoft membuat program Microsoft Visual J++ sebagai sebuah langkah percobaan untuk menggunakan Java di dalam sistem operasi Windows untuk meningkatkan antarmuka dari Microsoft Component Object Model (COM). Akan tetapi, akibat masalah dengan pemegang hak cipta bahasa pemrograman Java, Sun Microsystems, Microsoft pun menghentikan

pengembangan J++, dan beralih untuk membuat pengganti J++, kompilernya dan mesin virtualnya sendiri dengan menggunakan sebuah bahasa pemrograman yang bersifat general-purpose. Untuk menangani proyek ini, Microsoft merekrut Anders Helsberg, yang merupakan mantan karyawan Borland yang membuat bahasa Turbo Pascal, dan Borland Delphi, yang juga mendesain Windows Foundation Classes (WFC) yang digunakan di dalam J++. Sebagai hasil dari usaha tersebut, C# pun pertama kali diperkenalkan pada bulan Juli 2000 sebagai sebuah bahasa pemrograman modern berorientasi objek yang menjadi sebuah bahasa pemrograman utama di dalam pengembangan di dalam platform Microsoft .NET Framework.

Pengalaman Helsberg sebelumnya dalam pendesain bahasa pemrograman seperti Visual J++, Delphi, Turbo Pascal) dengan mudah dilihat dalam sintaksis bahasa C#, begitu pula halnya pada inti Common Language Runtime (CLR). Dari kutipan atas interview dan makalah-makalah teknisnya ia menyebutkan kelemahan-kelemahan yang terdapat pada bahasa pemrograman yang umum digunakan saat ini, misalnya C++, Java, Delphi, ataupun Smalltalk. Kelemahan-kelemahan yang dikemukakannya itu yang menjadi basis CLR sebagai bentukan baru yang menutupi kelemahan-kelemahan tersebut, dan pada akhirnya mempengaruhi desain pada bahasa C# itu sendiri. Ada kritik yang menyatakan C# sebagai bahasa yang berbagi akar dari bahasa-bahasa pemrograman lain. [1] Fitur-fitur yang diambilnya dari bahasa C++ dan Java adalah desain berorientasi objek, seperti garbage collection, reflection, akar kelas (root class), dan juga penyederhanaan terhadap pewarisan jamak (multiple inheritance). Fitur-fitur tersebut di dalam C#

kini telah diaplikasikan terhadap iterasi, properti, kejadian (event), metadata, dan konversi antara tipe-tipe sederhana dan juga objek. C# didisain untuk memenuhi kebutuhan akan sintaksis C++ yang lebih ringkas dan Rapid Application Development yang 'tanpa batas' (dibandingkan dengan RAD yang 'terbatas' seperti yang terdapat pada Delphi dan Visual Basic).

Agar mampu mempromosikan penggunaan besar-besaran dari bahasa C#, Microsoft, dengan dukungan dari Intel Corporation dan Hewlett-Packard, mencoba mengajukan standardisasi terhadap bahasa C#. Akhirnya, pada bulan Desember 2001, standar pertama pun diterima oleh European Computer Manufacturer Association (ECMA), dengan nomor standar ECMA-334. Pada Desember 2002, standar kedua pun diadopsi oleh ECMA, dan tiga bulan kemudian diterima oleh International Organizaion for Standardization (ISO), dengan nomor standar ISO/IEC 23270:2006.

2.8.2.2 Pengertian C#.net

C# (dibaca: C sharp) merupakan sebuah bahasa pemrograman yang berorientasi objek yang dikembangkan oleh Microsoft sebagai bagian dari inisiatif kerangka .NET Framework. Bahasa pemrograman ini dibuat berbasiskan bahasa C++ yang telah dipengaruhi oleh aspek-aspek ataupun fitur bahasa yang terdapat pada bahasa-bahasa pemrograman lainnya seperti Java, Delphi, Visual Basic, dan lain-lain) dengan beberapa penyederhanaan. Menurut standar ECMA-334 C# Language Specification, nama C# terdiri atas sebuah huruf Latin C (U+0043) yang diikuti oleh tanda pagar yang menandakan angka # (U+0023). Tanda pagar # yang digunakan memang bukan tanda kres dalam seni musik (U+266F), dan tanda

pagar # (U+0023) tersebut digunakan karena karakter kres dalam seni musik tidak terdapat di dalam keyboard standar.

2.8.2.3 Tujuan Desain

Standar European Computer Manufacturer Association (ECMA) mendaftarkan beberapa tujuan desain dari bahasa pemrograman C#, sebagai berikut:

1. Bahasa pemrograman C# dibuat sebagai bahasa pemrograman yang bersifat general-purpose (untuk tujuan jamak), berorientasi objek, modern, dan sederhana.
2. Bahasa pemrograman C# ditujukan untuk digunakan dalam mengembangkan komponen perangkat lunak yang mampu mengambil keuntungan dari lingkungan terdistribusi.
3. Portabilitas programmer sangatlah penting, khususnya bagi programmer yang telah lama menggunakan bahasa pemrograman C dan C++.
4. Dukungan untuk internasionalisasi (multi-language) juga sangat penting.
5. C# ditujukan agar cocok digunakan untuk menulis program aplikasi baik dalam sistem klien-server (hosted system) maupun sistem embedded (embedded system), mulai dari program aplikasi yang sangat besar yang menggunakan sistem operasi yang canggih hingga kepada program aplikasi yang sangat kecil yang memiliki fungsi-fungsi terdedikasi.
6. Meskipun aplikasi C# ditujukan agar bersifat 'ekonomis' dalam hal kebutuhan pemrosesan dan memori komputer, bahasa C# tidak ditujukan untuk bersaing secara langsung dengan kinerja dan ukuran program

aplikasi yang dibuat dengan menggunakan bahasa pemrograman C dan bahasa rakitan.

7. Bahasa C# harus mencakup pengecekan jenis (type checking) yang kuat, pengecekan larik (array), pendeteksian terhadap percobaan terhadap penggunaan variabel-variabel yang belum diinisialisasikan, portabilitas kode sumber, dan pengumpulan sampah (garbage collection) secara otomatis.

Fitur bahasa yang digunakan C# terkadang dapat disebutkan sebagai bahasa pemrograman yang paling mencerminkan dasar dari CLR dimana semua program-program .NET berjalan, dan bahasa ini sangat bergantung pada kerangka tersebut sebab ia secara spesifik didisain untuk mengambil manfaat dari fitur-fitur yang tersedia pada CLR.

2.9 Halma

Halma ditemukan oleh seorang profesor berkebangsaan Amerika dari Boston, Dr. George Howard Monks (1853 – 1933) diantara tahun 1883 dan 1884. Monk adalah seorang ahli bedah spesifikasi bagian dada di *Harvard Medical School*. Saudara George, Robert Monks berada di Inggris pada tahun 1883 atau 1884 dan Robert menulis surat kepada saudaranya dan mendeskripsikan permainan Inggris '*Hoppity*'. G.W. Monks mengambil beberapa ketentuan dari *Hoppity* dan membangun *Halma*.

Dr. Thomas Hill (1818 – 1891), seorang ahli matematika, guru dan pendeta, membantu merancang permainan ini. Dr. Thomas Hill menamakan

permainan ini '*Halma*' yang berasal bahasa Yunani yang berarti melompat. Hill adalah presiden dari *Harvard College* antara tahun 1862 – 1868 dan merupakan ayah mertua dari Robert Monks.

Halma pertama kali dipublikasi di Amerika Serikat (AS) pada tahun 1885 oleh *E.I. Horsman Company*. Terdapat kontroversi pada *Halma* di AS, dimana *Milton Bradley Company* juga mengklaim hak atas permainan ini. Tidak jelas apakah terjadi perang hak legal, tetapi kemudian *Milton Bradley* memproduksi dan memasarkan versi modifikasi dari *Halma* dengan nama *Eckha* pada tahun 1889.

Parker Bros mengklaim bahwa George H. Monks menjual hak paten atas *Halma* kepada mereka, namun hal itu tidak diverifikasi.

Di Inggris, *Spears Co*. memproduksi permainan *Halma* pada bulan Juli 1893. Halma adalah satu-satunya permainan klasik pada abad 19 yang berasal dari Amerika Serikat dan dikenal secara internasional.