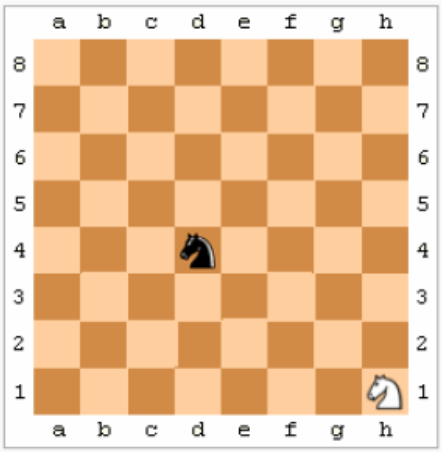
BAB II
LANDASAN TEORI

2.1 Permainan Catur

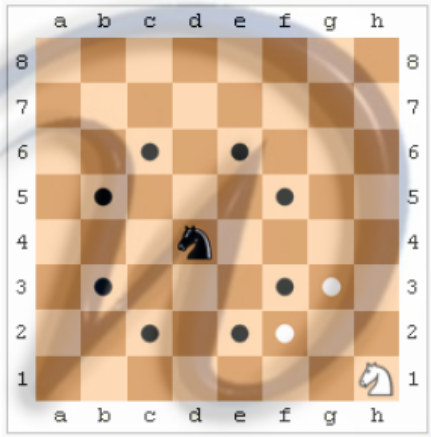
Permainan catur adalah permainan kuno yang telah dimainkan berabad-abad lamanya. Permainan catur dimainkan di atas papan yang memiliki 64 kotak (blok). Terdapat 2 kubu yang saling berhadapan (putih dan hitam), masing-masing kubu memiliki jumlah pasukan yang sama. Permainan catur berakhir ketika salah satu raja terbunuh. Kubu yang rajanya terbunuh dianggap sebagai pihak yang kalah dan yang rajanya masih hidup dianggap sebagai pemenangnya. Papan catur terdiri atas 8 baris dan 8 kolom dengan warna berselang-seling antara putih dan hitam, dengan dimulai warna putih pada baris 1 kolom 1. Setiap kotak pada papan catur memiliki nama tersendiri. Kotak-kotak dengan arah *horizontal* diberi nama a, b, c, d, e, f, g dan h sedangkan kotak-kotak dengan arah *vertical* diberi nomor urut yaitu: 1, 2, 3, 4, 5, 6, 7 dan 8.

2.1.1 *Knight Logic Problem's*

Knight adalah turunan dari permainan catur dimana pada permainan catur biasa terdapat lebih dari 32 prajurit, pada catur kuda hanya terdapat 2 buah pion, yakni kuda dipihak putih dan hitam. Perbedaan antara catur biasa dengan catur kuda adalah jumlah pionnya dan kondisi berakhirnya permainan. Pada catur biasa permainan berakhir ketika salah satu raja terbunuh, kubu yang rajanya masih hidup dinyatakan sebagai pemenang. Pada catur kuda kondisi berakhir ketika salah satu kuda tidak dapat melangkah lagi (pada catur kuda, bidang yang telah ditempati tidak boleh ditempati lagi oleh kedua belah pihak). Kuda catur adalah salah satu buah catur yang memiliki gerak atau langkah unik dengan membentuk huruf L. Kuda yang digunakan dalam catur ini hanya satu buah bidak kuda. Setiap kotak pada papan catur dinyatakan sebagai titik dan untuk langkah kuda yang mungkin dinyatakan sebagai sisi.



Gambar 2.1 Awal Posisi Kuda pada Papan Catur 8 x 8[2]



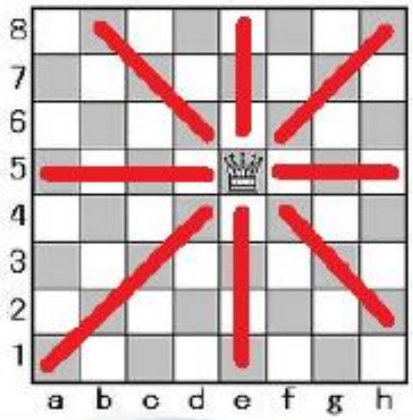
Gambar 2.2 Langkah Kuda[2]

Langkah kuda hitam dapat bergerak ke arah delapan kotak dengan titik hitam. Kuda putih hanya dapat bergerak terbatas kearah dua kotak dengan titik putih.

2.1.2 *8-Queen Problem's*

8-Queen Problem merupakan suatu permasalahan dimana ada terdapat 8 buah bidak Ratu dalam permainan catur dan dengan papan catur berukuran 8 x 8. Bidak Ratu sebanyak 8 ditempatkan sedemikian rupa dalam papan catur dengan syarat Bidak Ratu tidak boleh saling memakan satu sama lain. Arah gerak makan bidak Ratu pada *8-Queen Problem* sama dengan arah gerak bidak Ratu pada permainan catur secara umum yaitu bidak Ratu dapat bergerak ke arah horizontal kiri, horizontal kanan, vertikal atas, vertikal bawah dan juga ke arah diagonal, sehingga tidak boleh ada bidak Ratu yang saling berada di dalam satu garis

horizontal, vertikal, dan diagonal. Gambar arah gerak bidak Ratu seperti Gambar 2.3 di bawah ini.

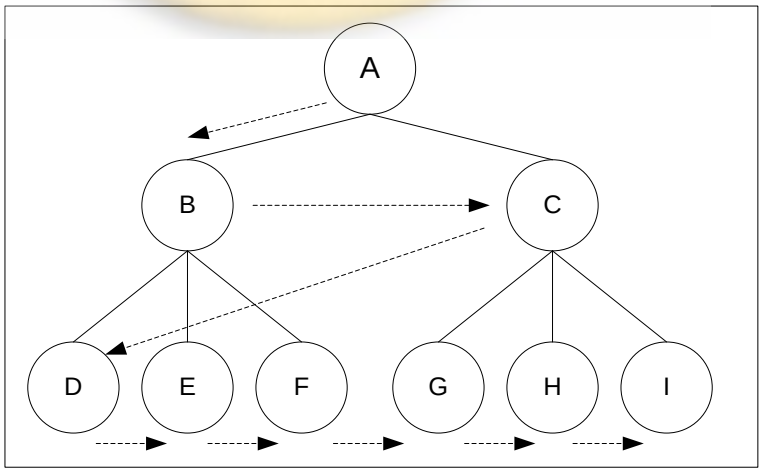


Gambar 2.3 Langkah *Queen*[3]

Gambar 2.3 di atas merupakan gambar arah gerak bidak Ratu sehingga pada jalur dengan warna merah tidak boleh ada bidak Ratu yang lain.

2.2 Pencarian Melebar Pertama *(Breadth-First Search)*

Pada metode *Breadth-First Search*, semua *node* pada level n akan dikunjungi terlebih dahulu sebelum mengunjungi *node-node* pada level n+1. Pencarian dimulai dari *node* akar terus ke level ke-1 dari kiri ke kanan, kemudian berpindah ke level berikutnya dan seterusnya hingga ditemukannya solusi. Penelusuran pohon dengan metode *Breadth-First Search* dapat dilihat pada Gambar 2.4 di bawah ini.

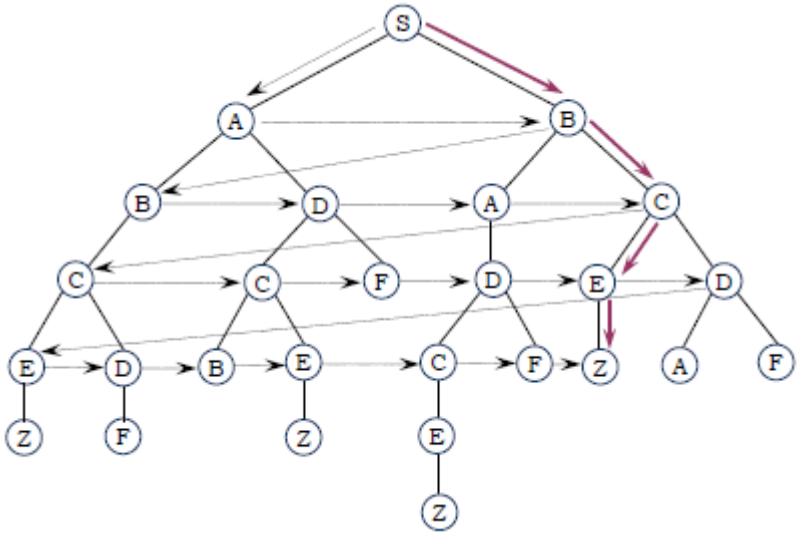


Gambar 2.4 Penelusuran Pohon dengan Metode *Breadth-First Search*[4]

Untuk memperjelas cara kerja algoritma BFS beserta antrian yang digunakannya,berikut langkah-langkah algoritma BFS :

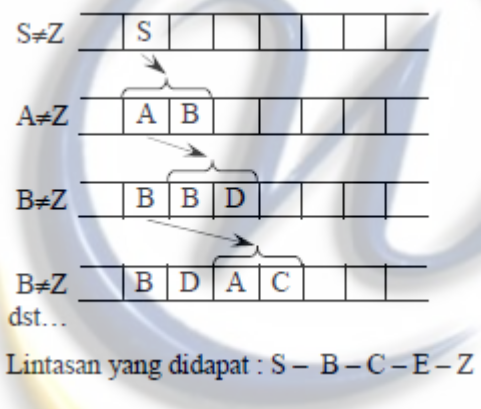
1. Masukkan *node* ujung (akar) ke dalam antrian.
2. Ambil *node* dari awal antrian, lalu cek apakah *node* merupakan solusi.
3. Jika *node* merupakan solusi, pencarian selesai dan hasil dikembalikan.
4. Jika *node* bukan solusi, masukkan seluruh *node* yang bertetangga dengan *node* tersebut (*node* anak) ke dalam antrian.
5. Jika antrian kosong dan setiap *node* sudah dicek, pencarian selesai dan mengembalikan hasil solusi tidak ditemukan.
6. Ulangi pencarian dari langkah kedua.

Kompleksitas ruang algoritma BFS adalah O (jumlah *node* +jumlah sisi). Hal ini menunjukkan bahwa algoritma BFS memerlukan ruang besar dan tidak cocok untuk pemecahan masalah besar. Angka yang sama menunjukkan kompleksitas waktu terburuk, di mana setiap simpul harus diperiksa. Kompleksitas waktu terbaik, O(1), didapat jika solusi ditemukan pada pencarian pertama. Walaupun memiliki kompleksitas ruang dan waktu yang kurang baik, algoritma BFS dapat mencari solusi secara tuntas, berarti algoritma ini akan menemukan solusi terbaik dan terdekat bagaimanapun bentuk graf tersebut. Keuntungan dari metode ini adalah tidak akan menemui jalan buntu dan Jika terdapat lebih dari satu solusi, maka BFS akan menemukannya. 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). Contoh Kasus dengan menggunakan *breadth-first search* dapat dilihat pada Gambar 2.5 halaman II-5



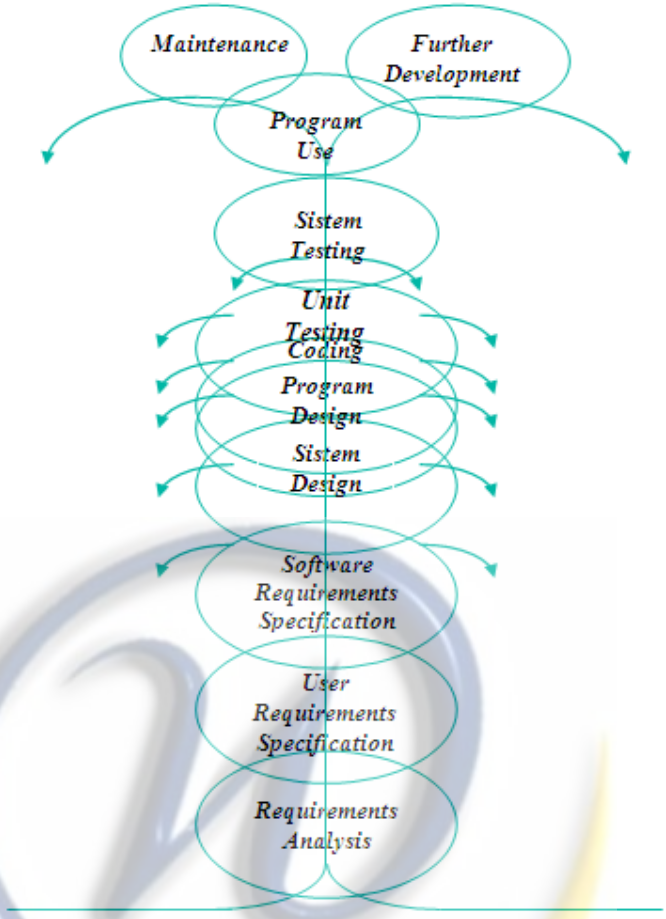
Gambar 2.5 Contoh Kasus *Breadth-First Search* pada *Tree*

Penyelesaian kasus berdasarkan Gambar 2.5 di atas :



2.3 Metode Rekayasa Perangkat Lunak dengan *Fountain Model*

Fountain model merupakan model rekayasa perangkat lunak yang diterapkan untuk pembangunan sistem yang bersifat *object oriented*. Tahapan-tahapan pengembangan yang dilakukan adalah Analisa kebutuhan, spesifikasi kebutuhan *user*, perancangan sistem, perancangan program, *coding*, unit *testing*, sistem *testing*, implementasi, pemeliharaan dan pengembangan lebih lanjut. Tahapan-tahapan tersebut saling terkait dan saling mempengaruhi (*life cycle*). Keterkaitan tahapan-tahapan dapat dilihat pada Gambar 2.6 halaman II-6.



Gambar 2.6 *Fountain Model*[5]

Berikut ini merupakan penjelasan dari tahapan-tahapan yang ada pada *fountain model* adalah sebagai berikut :

1. **Tahap *requirement analysis***: menganalisa kebutuhan sistem dan semua yang berkaitan dengan proses yang terjadi pada sistem yang diperlukan untuk membangun sistem sehingga dapat berjalan dengan baik dan maksimal.
2. **Tahap *user requirement specification***: menjelaskan tentang kebutuhan dari sisi pengguna (*user*), sehingga *user* dapat menjalankan sistem dengan baik.
3. **Tahap *software requirements specification***: menjelaskan tentang kebutuhan sistem dari sisi *software* sehingga sistem yang ada dapat berjalan dengan baik dan maksimal.
4. **Tahap sistem *design***: Pada tahap perancangan ini diberikan gambaran umum yang jelas kepada pengguna dan rancang bangun yang lengkap tentang sistem yang akan dikembangkan kepada pihak-pihak yang terlibat dalam

pengembangan sistem. Perancangan disini dilakuan dengan permodelan menggunakan metode *Object Oriented* dengan *tools* UML (*Unified Modeling Language*). Tahapan perancangan sistem disini dibagi menjadi dua bagian, yaitu perancangan global dan perancangan rinci. Perancangan global dilakukan untuk memberikan gambaran umum kepada pengguna tentang sistem yang dirancang dan sebagai persiapan untuk tahap perancangan rinci. Perancangan rinci dilakukan untuk memberikan gambaran rancang bangun yang lengkap kepada pemrogram dan pihak-pihak lain yang terlibat dalam pengembangan sistem sebagai persiapan untuk tahap implementasi.

5. **Tahap program *design***: proses perancangan dan pembuatan kerangka program yang mengacu pada tahap sebelumnya, sehingga program yang dibuat dapat mememenuhi kebutuhan dari sistem dan mencakup penyelesaian masalah yang ada.
6. **Tahap *coding*:** proses penterjemahan tahap perancangan ke dalam bahasa pemrograman. Proses *coding* meliputi unit-unit yang ada pada keseluruhan sistem.
7. **Tahap *unit testing*** : proses pengujian terhadap setiap unit yang ada pada sistem sehingga dapat unit tersebut dapat melakukan kerja sesuai dengan yang diharapkan.
8. **Tahap *system testing***: Pengujan perangkat lunak adalah elemen kritis dari jaminan kualitas perangkat lunak dan merepresentasikan kajian pokok dari spesifikasi, desain, dan pengkodean. Sejumlah aturan yang berfungsi sebagai sasaran pengujian adalah:
   a. Pengujian adalah proses eksekusi suatu program dengan maksud menemukan kesalahan.
   b. *Test case* yang baik adalah *test case* yang memiliki *probabilitas* yang tinggi untuk menemukan kesalahan yang belum pernah ditemukan sebelumnya.
   c. Pengujian yang sukses adalah pengujian yang mengungkap semua kesalaham yang sebelumnya.

Pengujian perangkat lunak tidak pernah berakhir, pengujian hanya akan berpindah dari pengembang ke pelanggan. Setiap kali pelanggan menggunakan perangkat lunak tersebut berarti pengujian sedang dilakukan.

9. **Tahap *program use***: proses pengaplikasian dan pemakaian program yang telah dibuat untuk memastikan apakah tujuan yang semula telah terpenuhi.
10. **Tahap *maintenance***: pada tahap *maintenance*, setelah dilakukan pengujian dan sistem diyakini telah *valid*, selanjutnya sistem tersebut didistribusikan kepada pengguna. Banyak *software* tidak dapat bertahan 10 sampai 15 tahun, meskipun *software* tersebut dibuat dengan desain dan teknik pengkodean terbaik saat dibuat. *Software* akan memburuk dengan semestinya ketika pindah *platform* sistem operasi yang berbeda, kebutuhan baru dari pengguna atau *software* tidak cukup bisa menjawab kebutuhan fungsional. Pemeliharaan lebih dari sekedar "memperbaiki kesalahan", ada empat perbedaan aktifitas dalam pemeliharaan yaitu:
    a. *Corrective Maintenance*, adalah sama dengan garansi untuk *software*, atau memperbaiki kesalahan *software*.
    b. *Adaptive Maintenance*, adalah memodifikasi *software* untuk *platform* sistem operasi yang berbeda, atau memodifikasi *software* untuk mengakomodasi lingkungan eksternal.
    c. *Pervective Maintenence* atau *Enhancement*, adalah menambah dan mengenali fungsi tambahan yang bermanfaat diluar kebutuhan fungsional aslinya.
    d. *Preventive Maintenance* atau *Reengineering*, adalah pembangunan kembali *software* yang sudah memburuk kinerjanya. Aktifitas ini lebih mudah disamping aktifitas pemeliharaan yang lain.

    Hanya 20% dari semua pekerjaan pemeliharaan digunakan untuk melakukan perbaikan kesalahan, sedangkan sisanya 80% digunakan untuk memodifikasi *software* untuk mengakomodasi lingkungan eksternal, menambah dan mengenali fungsi tambahan dan pembangunan kembali *software* untuk digunakan di masa depan.
11. **Tahap *further development***: pada tahap ini dijelaskan kemungkinan yang ada untuk proses pengembangan sistem.

2.4 Perancangan *System* dengan UML (*Unified Modeling Language*)

2.4.1 *Use Case Diagram*

Use case diagram merupakan diagram yang bersifat statis, diagram ini memperlihatkan himpunan *use case* dan *actor* (suatu jenis khusus dari kelas) diagram ini sangat penting untuk mengorganisasikan dan memodelkan prilaku dari suatu sistem yang dibutuhkan serta diharapkan pengguna. Adapun simbol-simbol yang sering digunakan dalam *use case diagram* terdapat pada Tabel 2.1 di bawah ini.

Tabel 2.1 Simbol-simbol pada *Use Case* Diagram[5]

No.	Simbol	Keterangan
1.		Simbol *Actor*, menggambarkan aktor pada diagram.
2.		Simbol *UseCase*, menggambarkan *UseCase* pada diagram
3.		Simbol *Unidirectional Association*, menggambarkan relasi antar aktor dan use case

2.4.2 *Class* Analisis

Elemen model yang terdapat dalam model analisis disebut kelas analisis (*analysis class*). Kelas analisis adalah kelas ber-*stereotype* "*Boundary*", "*Control*", atau "*Entity*" yang menggambarkan sebuah konsep awal mengenai "benda" dalam sistem aplikasi yang memiliki tanggung jawab dan perilaku. Kelas analisi akhirnya berkembang menjadi kelas didalam model desain. Adapun simbol - simbol yang sering digunakan dalam *Class* analisis terdapat pada tabel 2.2 halaman II-10 :

Tabel 2.2 Simbol-simbol pada *Class* Analisis[5]

No.	Simbol	Keterangan
1.		Simbol *Boundary*, menggambarkan batasan kelas pada diagram. Dimana kelas yang memodelkan interaksi antara satu atau lebih *actor* dengan system
2.		Simbol *Control*, menggambarkan unsur kendali pada diagram.
3.		*Entity* menggambarkan kelas entitas pada diagram.

2.4.3 *Class* Perancangan

Class diagram merupakan diagram yang bersifat statis, diagram ini memperlihatkan himpunan kelas-kelas, antarmuka-antarmuka, kolaborasi-kolaborasi, serta relasi-relasi. Diagram ini umum dijumpai pada pemodelan berorientasi objek, meskipun bersifat statis sering pula diagram kelas ini memuat kelas-kelas aktif. Adapun simbol-simbol yang sering digunakan dalam *Class* Perancangan terdapat pada tabel 2.3 di bawah ini :

Tabel 2.3 Simbol-simbol pada *Class* Perancangan[5]

No.	Simbol	Keterangan
1.		Simbol *Actor*, menggambarkan aktor pada diagram.
2.		Simbol *Agregation*, menggambarkan relasi agregasi
3.		Simbol *Association*, menggambarkan relasi asosiasi.

2.4.4 *Sequence Diagram*

Sequence diagram merupakan diagram yang bersifat dinamis, diagram urutan adalah diagram interaksi yang menekankan pada pengiriman pesan dalam suatu waktu tertentu. Adapun simbol-simbol yang sering digunakan dalam *Sequence Diagram* terdapat pada Tabel 2.4 di bawah ini :

Tabel 2.4 Simbol-simbol pada *Sequence Diagram*[5]

No.	Simbol	Keterangan
1.		Simbol *Actor*, menggambarkan aktor pada diagram.
2.		Simbol *Boundary*, menggambarkan batasan kelas pada diagram.
3.		Simbol *Control*, menggambarkan unsure kendali pada diagram.
4.		*Entity* menggambarkan kelas entitas pada diagram.
5.		*Object Message*, menggambarkan pesan antar dua objek.
6.		*Message to Self*, menggambarkan pesan yang menuju dirinya sendiri.

2.4.5 *Collaboration Diagram*

Collaboration diagram merupakan diagram yang bersifat dinamis, diagram kolaborasi adalah diagram interaksi yang menekankan organisasi struktural dari objek-objek yang menerima serta mengirim pesan. Adapun simbol-simbol yang sering digunakan dalam *Collaboration Diagram* terdapat pada Tabel 2.5 di bawah ini :

Tabel 2.5 Simbol-simbol pada *Collaboration Diagram*[5]

No.	Simbol	Keterangan
1.		Simbol *Actor*, menggambarkan aktor pada diagram.
2.		Simbol *Boundary*, menggambarkan batasan kelas pada diagram.

No.	Simbol	Keterangan
3.		Simbol *Control*, menggambarkan unsure kendali pada diagram.
4.		*Entity* menggambarkan kelas entitas pada diagram.
5.		*Link to Self*, menggambarkan bahwa suatu objek memanggail operasinya sendiri
6.		*Object Link*, menggambarkan lintasan komunikasi antar dua objek.
7.		*Link Message*, menggambarkan pesan antar dua objek, atau dari suatu objek ke dirinya sendiri.
8.		*Reverse Link Message*, menggambarkan pesan dalam arah berlawanan antar dua objek. atau dari suatu objek ke dirinya sendiri.

2.4.6 *Activity Diagram*

Activity diagram merupakan diagram yang bersifat dinamis, diagram aktivitas adalah tipe khusus dalam diagram *state* yang memperlihatkan aliran dari sesuatu aktifitas ke aktfitas lainnya dalam suatu sistem. Diagram ini terutama penting dalam pemodelan fungsi-fungsi dalam suatu sistem dan memberi tekanan pada aliran kendali antar objek. Adapun simbol-simbol yang sering digunakan dalam *Activity Diagram* terdapat pada Tabel 2.6 di bawah ini :

Tabel 2.6 Simbol-simbol pada *Activity Diagram*[5]

No.	Simbol	Keterangan
1.		Simbol *Start state*, menggambarkan aliran kerja berawal.
2.		Simbol *End state*, menggambarkan aliran kerja berakhbair.

No.	Simbol	Keterangan
3.		Simbol *Decision*, menggambarkan Titik keputusan pada aliran kerja.
4.		*State Transition*, menggambarkan transisi dari suatu aktivitas ke aktivitas yang lain.
5.		*State*, menggambarkan state untuk suatu objek.

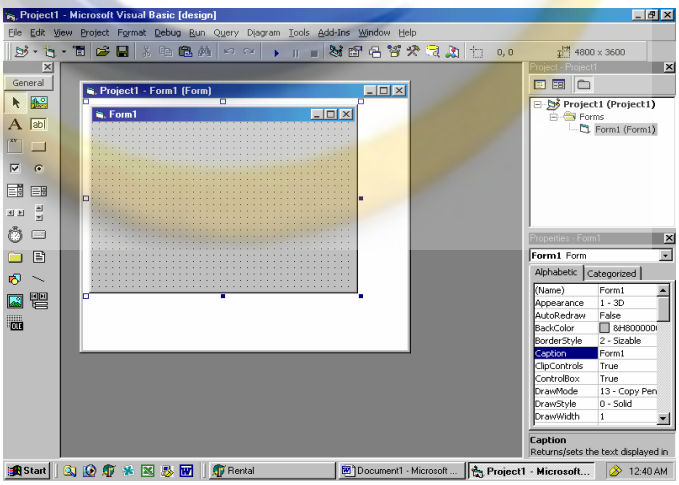
2.5 Lingkungan Implementasi

2.5.1 Microsoft Visual Basic 6.0[8]

Visual basic merupakan bahasa pemrograman yang paling handal digunakan sehingga banyak yang menggunakan. Dibandingkan dengan bahasa pemrograman lain semisal pascal maka lebih mudah bagi kita menggunakan visual basic dimana dalam penulisan kode program mampu menambahkan sendiri.

2.5.2 Lingkungan Visual Basic 6.0

Berikut ini merupakan tampilan dari lingkungan visual basic 6.0. Tampilan dari lingkungan visual basic dapat dilihat pada Gambar 2.7 di bawah ini



Gambar 2.7 Lingkungan Visual Basic[6]

Pada lingkungan Visual Basic terdiri beberapa menu khusus diantaranya :

1. *Toolbar*

   Toolbar merupakan sebuah batang yang berisi kumpulan tombol yang terletak dibagian bawah menu bar yang dapat digunakan untuk menjalankan suatu perintah. Pada kondisi *default* program visual basic hanya menampilkan *toolbar* standar. Adapun *toolbar* standar pada visual basic dapat dilihat pada Gambar 2.8 di bawah ini.

Gambar 2.8 *Toolbar*[6]

2. *Toolbox*

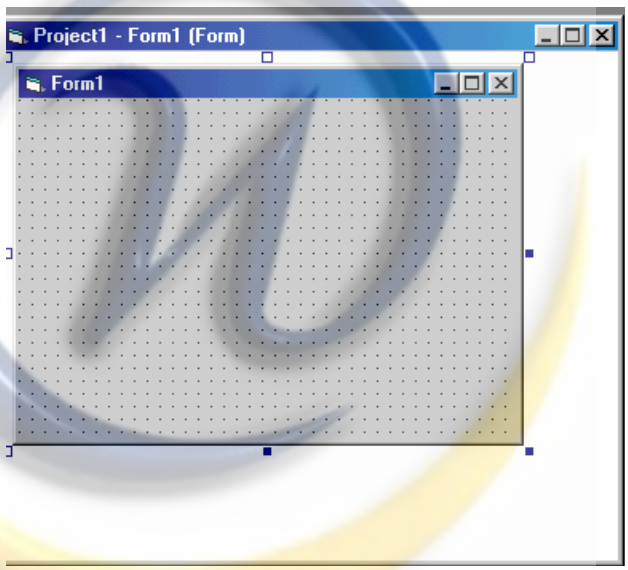
   Toolbox adalah suatu objek yang akan menjadi penghubung antara program aplikasi dan menggunakannya, dan kesemuanya harus diletakkan di dalam jendela *form*. Pada kondisi *default*, *toolbox* menampilkan tabulasi *general* dengan 21 tombol kontrol yang dapat ditampilkan.

   Bentuk *Toolbox* Visual Basic dapat dilihat pada Gambar 2.9 di bawah ini

Gambar 2.9 *Toolbox*[6]

3. *Form Window*

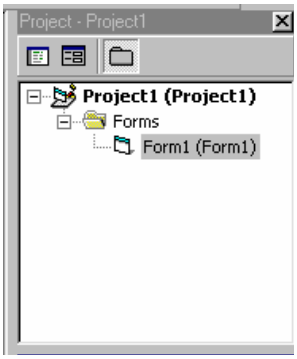
   Form Window adalah daerah kerja utama, dimana dapat digunakan untuk membuat program-program aplikasi Visual Basic. Pada *form* ini dapat ditempatkan berbagai macam objek interaktif misalnya teks, gambar, tombol-tombol perintah, *database*, *combo box* dan lain-lain. Jendela *form* ini pada awalnya berukuran kecil, tetapi ukurannya dapat diubah-ubah sesuai dengan kebutuhan tampilan yang diperlukan. Apabila program aplikasi yang sudah dijalankan, maka semua yang terdapat didalam jendela ini menjadi latar belakang dari aplikasi program. Bentuk jendela *form* yang masih kosong seperti Gambar 2.10 di bawah ini



Gambar 2.10 *Form Window*[6]

4. *Project Explorer*

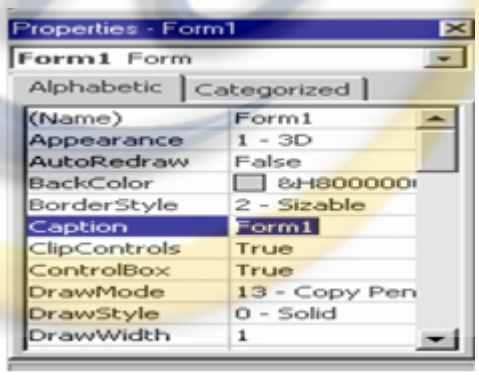
   Jendela *Project Explorer* adalah jendela yang mengandung semua *file* di dalam apliksi Visual Basic. Setiap aplikasi dalam Visual Basic disebut dengan istilah *project* (proyek), dan setiap proyek dapat mengandung lebih dari satu file. Tampilan dari *project explorer* dapat dilihat pada Gambar 2.11 halaman II-16.



Gambar 2.11 *Project Explorer*[6]

5. Jendela *Properties*

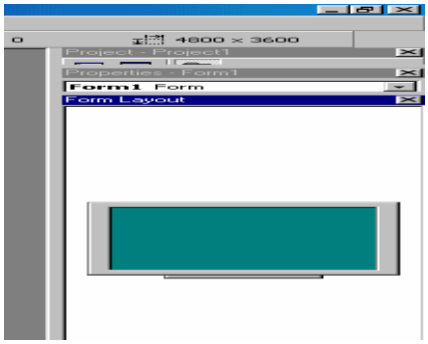
   Jendela *properties* adalah jendela yang mengandung semua informasi mengenai objek yang terdapat pada aplikasi Visual Basic. *Property* adalah sifat sebuah objek, misalnya sifat tampilan, warna, ukuran huruf, dan sebagainya. Setiap objek sebagian besar memiliki jenis *property* yang sama, tetapi tidak menutup kemungkinan untuk berbeda. Melalui jendela properties ini dapat diatur bentuk dan karakteristik dari setiap objek. *Properties* ini dapat ditampilkan urut berdasarkan abjad ataupun diurutkan berdasarkan *categories*. Tampilan dari jendela *properties* dapat dilihat pada Gambar 2.12 di bawah ini.



Gambar 2.12 Jendela *Properties*[6]

6. *Form Layout Window*

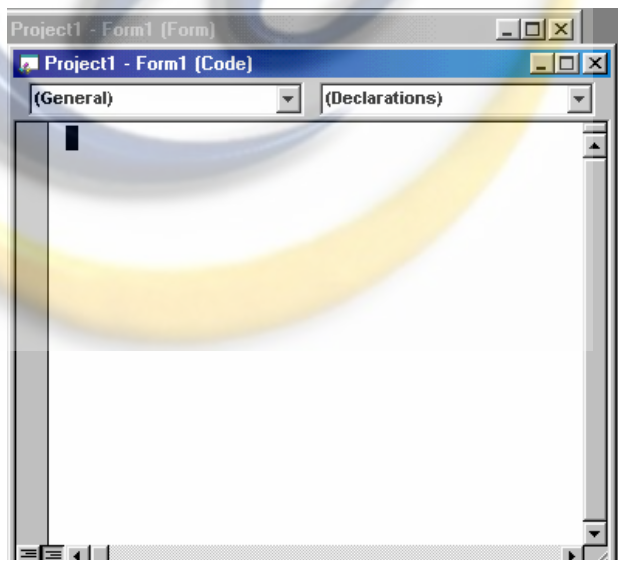
   Form layout window merupakan sebuah jendela yang dipergunakan untuk mengatur posisi dari *form* pada *form* saat program dijalankan. Pada saat mengarahkan pointer *mouse* ke bagian *form*, maka *pointer mouse* akan berubah menjadi anak panah empat arah (*pointer* pengatur posisi) untuk memindah posisi *form* pada layar monitor dapat dilakukan dengan proses *drag* atau *drop form layout window*. Tampilan dari *form layout window* dapat dilihat pada Gambar 2.13 di halaman II-17.



Gambar 2.13 *Form Layout Window*[6]

7. Jendela Kode

   Jendela kode adalah salah satu jendela yang paling penting di dalam Visual Basic. Jendela ini berisi kode-kode program yang merupakan intruksi-intruksi untuk aplikasi Visual Basic. Setiap objek pada Visual Basic dapat ditambahi kode-kode program untuk melaksanakan tugas-tugas tertentu, misalnya membatalkan perintah, menutup aplikasi, mengontrol keadaan, dan sebagainya. Tampilan dari jendela kode dapat dilihat pada Gambar 2.14 di bawah ini



Gambar 2.14 Jendela Kode[6]