BAB II
LANDASAN TEORI

2.1 Kecerdasan Buatan (*Artificial Intelligence*)

Artificial Intelligence (AI) atau Intelegensi Buatan atau Kecerdasan Buatan dapat didefinisikan sebagai perlakuan terhadap komputer sehingga komputer dapat berpikir dan melakukan hal-hal yang manusia dapat lakukan. Menurut Rich dan Knight (1991) dalam bukunya yang berjudul *Artificial Intelligence Second Edition*, kecerdasan buatan adalah "sebuah studi tentang bagaimana membuat komputer melakukan hal-hal yang pada saat ini dapat dilakukan dengan lebih baik oleh manusia".

Tujuan umum dari kecerdasan buatan antara lain:

1. Membuat mesin menjadi lebih pintar (tujuan utama)
2. Memahami apa itu kecerdasan (tujuan ilmiah)
3. Membuat mesin menjadi lebih bermanfaat (tujuan entrepreneurial)

Masa sekarang, kecerdasan buatan umum dipakai dalam dua kategori besar, yaitu *Academic AI* dan *Game AI* (Millington, 2009). Sesuai dengan namanya, *Academic AI* adalah kecerdasan buatan yang dikembangkan dan diimplementasikan untuk keperluan penelitian akademik. Tujuan dari *Academic AI* adalah membuat sesuatu menjadi inteligen (dapat dibandingkan dengan tingkat kognitif manusia), atau mempelajari dan menanggapinya lewat aksi yang ditentukan dengan model komputasi sehingga tercapai hasil yang diinginkan yaitu melakukan aksi seperti yang manusia lakukan, dengan lebih baik dan efisien. *Academic AI* yang populer adalah *Machine Learning* dan *Deep Learning*. *Machine Learning* adalah tipe

kecerdasan buatan yang memberikan komputer kemampuan untuk belajar tanpa deprogram secara eksplisit. *Machine Learning* berfokus pada pengembangan program komputer yang dapat berubah bila diberikan data-data baru. Data-data baru tersebut digunakan untuk mendeteksi pola pada data dan aksi program akan disesuaikan berdasarkan analisa pola data baru tadi. Contoh dari *Machine Learning* adalah *timeline* (lini masa) pada Facebook. Facebook akan mempelajari perilaku pengguna lewat interaksinya. Saat pengguna menyukai pesan status dari teman, maka Facebook akan mulai memunculkan pesan status dari teman tersebut lebih banyak. Saat pengguna sudah jarang menyukai pesan status dari teman itu, maka Facebook akan menyesuaikan *timeline* (lini masa) untuk mengurangi pesan status teman itu. Berbeda dengan *Machine Learning*, *Deep Learning* lebih menjurus pada meniru pendekatan pembelajaran yang dilakukan manusia untuk memperoleh pengetahuan tertentu. *Deep Learning* lebih kompleks dan abstrak. Program *Deep Learning* akan dilatih dengan sekumpulan data tertentu hingga program dapat dengan akurat mendapatkan hasil yang diinginkan. *Deep Learning* dapat dilihat sebagai cara untuk mengotomasikan analisis prediktif. Deep Learning banyak digunakan untuk kebutuhan penelitian di bidang medis. Contoh dari *Deep Learning* adalah diagnosa kanker usus buntu yang dilakukan oleh Nvidia. Selain di bidang medis, *Deep Learning* juga digunakan untuk *Autonomous Vehicle* yang sedang diteliti dan dikembangkan oleh Google dan Tesla.

Game AI adalah kecerdasan buatan yang mencakup semua teknik yang digunakan dalam pembuatan lawan bermain dalam suatu permainan. Teknik tersebut diantaranya adalah *pathfinding* (pencarian jalan), *planning* (perencanaan)

dan *decision making* (penentuan keputusan), *learning* (pembelajaran), dan sebagainya.

Gambar 2.1 Peta Navigasi Untuk Musuh Pada *Game* F.E.A.R. 2
(Sumber: https://www.bit-tech.net/gaming/2009/03/05/how-ai-in-games-works/4)

Gambar 2.2 *Navmesh* Kompleks Pada *Game* F.E.A.R. 2
(Sumber: https://www.bit-tech.net/gaming/2009/03/05/how-ai-in-games-works/5)

Berbeda dengan *Academic AI*, *Game AI* biasanya tidak melakukan pembelajaran apapun dikarenakan siklus komputasi dan waktu yang terbatas bagi *Game AI* untuk cukup 'berpikir'. Selain itu *Game AI* lebih berorientasi pada hasil, dengan program yang umumnya tidak sekompleks *Academic AI* dan tidak berubah-ubah. *Game AI* akan menentukan jalannya permainan lewat aksi lawan bermain atau reaksi lingkungan dalam permainan.

2.2 *Tic Tac Toe*

Tic Tac Toe adalah permainan yang masuk kategori *genre paper-and-pencil-game* atau dengan kata lain permainan yang dapat dimainkan hanya dengan kertas dan pensil (atau alat tulis lainnya). Permainan ini mempunyai nama lain *Nought & Cross*. Ada juga yang menyebutnya X dan O karena umumnya dimainkan dengan menggunakan bidak/simbol huruf X dan huruf O.

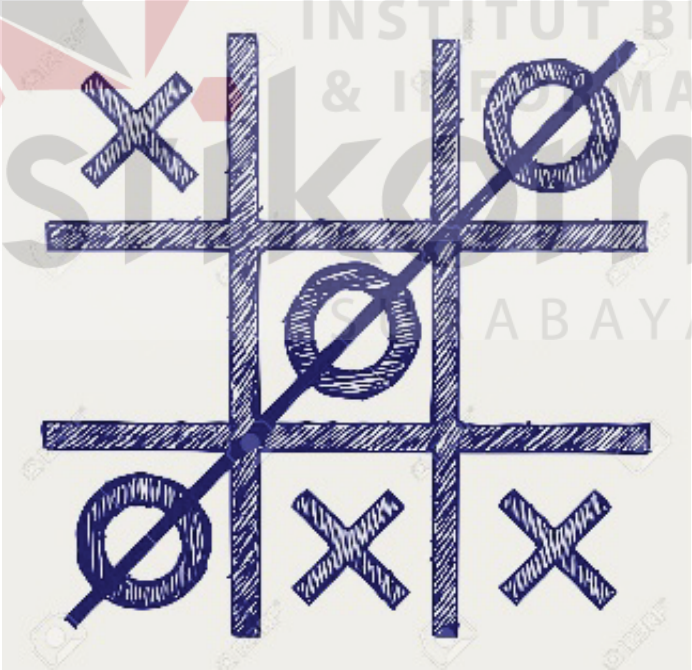
Sama dengan semua permainan papan, *Tic Tac Toe* mempunyai aturan dalam bermain. Peraturan yang sudah ditentukan adalah berikut:

a) Permainan dimainkan oleh dua orang pemain
b) Permainan dimulai dengan papan permainan dalam kosong
c) Pemain secara bergiliran menempatkan atau menandai ruang dalam kotak berukuran 3 × 3 dengan simbol yang ditentukan.
d) Pemain pertama dapat berupa simbol huruf X, dan pemain kedua dapat berupa simbol huruf O.
e) Dengan asumsi simbol huruf X bermain pertama, pemain dengan simbol huruf X mempunyai satu lebih banyak atau sama dengan jumlah dari simbol huruf O

f) Kondisi permainan berakhir adalah, saat salah satu pemain berhasil menempatkan simbolnya dalam satu baris (horisontal, vertikal, atau diagonal) atau saat papan telah penuh (tidak ada ruang kosong), manapun yang terjadi lebih dahulu

g) Bila kondisi permainan berakhir terpenuhi, kedua pemain berhenti (tidak ada yang bergerak lagi)

Terdapat tiga kemungkinan hasil akhir dari permainan ini. Pemain yang berhasil menempatkan tiga simbol mereka dalam baris horisontal, vertikal, atau diagonal dianggap memenangkan permainan. Sebaliknya, pemain yang tidak berhasil, dianggap kalah. Bila kedua pemain tidak dapat menempatkan tiga simbol yang segaris, maka permainan dianggap seri.



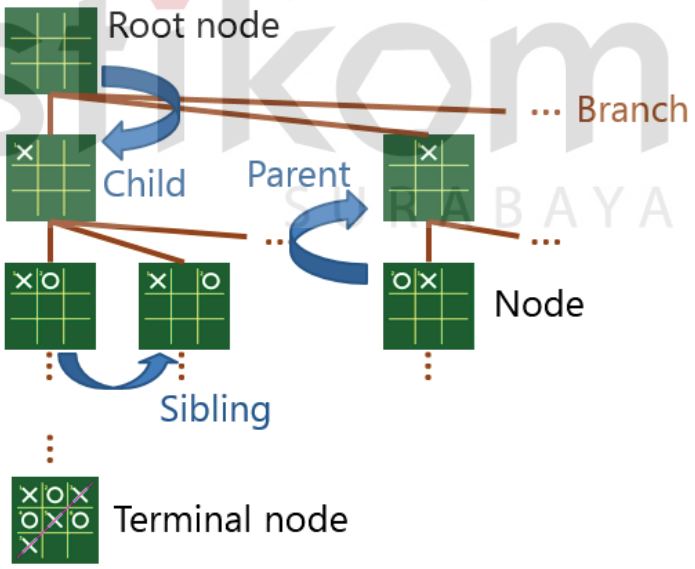
Gambar 2.3 Papan Permainan *Tic Tac Toe*

(Sumber: https://www.123rf.com/photo_16516172_tic-tac-toe-doodle-style.html)

2.3 Algoritma Pencarian

Ruang keadaan dalam permainan *Tic Tac Toe* dapat dipresentasikan dengan pohon pencarian (*tree search*). Tiap-tiap *node* pada pohon berhubungan dengan keadaan (*state*) yang mungkin dalam permainan tersebut. Setiap giliran/gerakan (*move*) akan menyebabkan perubahan dari keadaan sekarang (*current state*) ke keadaan selanjutnya (*child state*). Permasalahan yang dihadapi adalah menentukan *child state* mana yang terbaik.

Pohon (*tree*) merupakan *graph* yang masing-masing *node*-nya (kecuali *root*) hanya memiliki satu induk (*parent*), dengan kata lain tidak memiliki *cycle*. *Node* yang tidak memiliki *child* disebut *terminal node*. Metode pencarian yang umumnya digunakan pada pohon pencarian adalah *Breadth First Search* (BFS) atau *Depth First Search* (DFS).



Gambar 2.4 Contoh *Game Tree*

(Sumber: https://blogs.msdn.microsoft.com/smallbasic/2014/11/10/small-basic-game-programming-game-ai/)

2.3.1 *Minimax*

Minimax merupakan salah satu algoritma yang sering digunakan untuk game kecerdasan buatan seperti catur, yang menggunakan teknik *Depth First Search*. Algoritma *Minimax* akan melakukan pengecekan pada seluruh kemungkinan yang ada, sehingga akan menghasilkan pohon permainan yang berisi semua kemungkinan tersebut. Keuntungan penggunaan algoritma *Minimax* adalah mampu menganalisis semua kemungkinan posisi permainan untuk menghasilkan keputusan terbaik dengan mencari langkah yang akan membuat lawan mengalami kerugian. Fungsi evaluasi yang digunakan adalah fungsi evaluasi statis dengan asumsi lawan akan melakukan langkah terbaik yang mungkin. Pada *Minimax* dikenal adanya istilah *ply* yaitu gerakan pemain max dan lawan min.

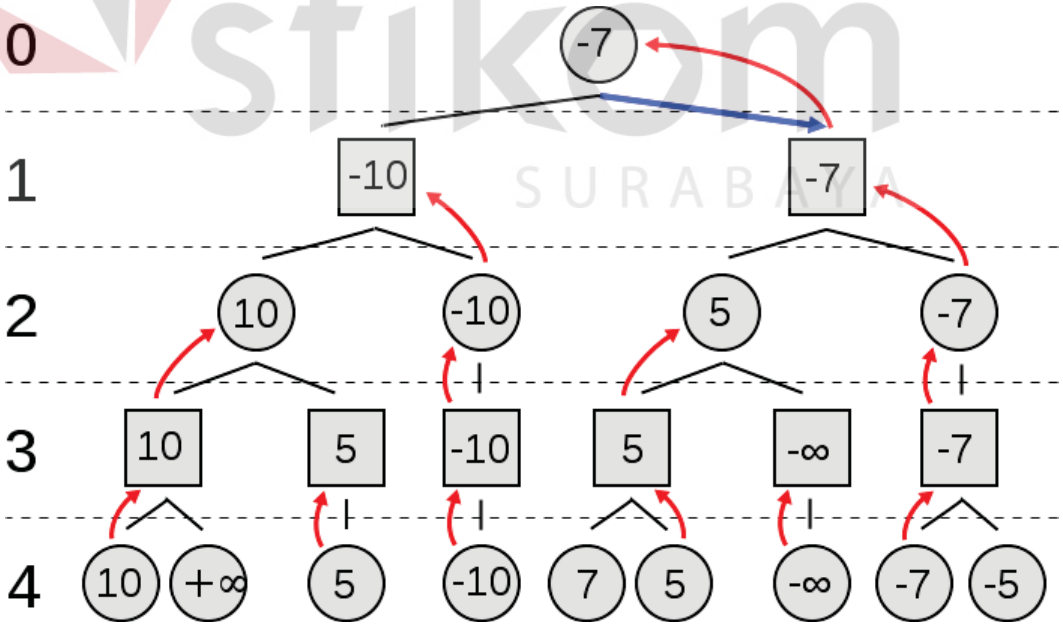
Di bawah ini adalah *pseudocode* untuk algoritma *Minimax*.

```
MinMax (GamePosition game)
{
     return MaxMove (game);
}

MaxMove (GamePosition game)
{
     if (GameEnded(game))
     {
          return EvalGameState(game);
     }
     else
     {
          best_move < - {};
          moves <- GenerateMoves(game);
          ForEach moves
          {
               move <- MinMove(ApplyMove(game));
               if (Value(move) > Value(best_move))
               {
                    best_move < - move;
               }
          }
          return best_move;
     }
}
```

```
MinMove (GamePosition game)
{
      best_move <- {};
      moves <- GenerateMoves(game);
      ForEach moves
      {
            move <- MaxMove(ApplyMove(game));
            if (Value(move) > Value(best_move))
            {
                  best_move < - move;
            }
      }
      return best_move;
}
```

Algoritma *Minimax* memiliki kelemahan yang cukup signifikan, yaitu masalah waktu. Algoritma ini menelusuri seluruh node yang ada pada pohon pencarian mulai dari kedalaman awal hingga kedalaman akhir. Perluasan pencarian setiap node ke bawah inilah yang menyebabkan pencarian Minimax membutuhkan waktu relatif lama untuk kasus-kasus tertentu.



Gambar 2.5 Contoh *Tree Minimax*

(Sumber: https://en.wikipedia.org/wiki/Minimax)

2.3.2 *Alpha-Beta Pruning*

Algoritma ini merupakan pengembangan dari algoritma sebelumnya yaitu algoritma *Minimax* yang telah dikembangkan oleh John McCarthy tahun 1956 karena beliau beranggapan bahwa pencarian *Minimax* kurang efektif. Masalah utama dengan pencarian *Minimax* adalah kita memperluas setiap node kebawah untuk kedalaman tertentu, dan untuk beberapa kasus pencarian ini akan membuang banyak sekali waktu. Kita dapat menggunakan teknik *branch and bound* atau batas cabang untuk mengurangi jumlah keadaan yang harus diuji untuk menentukan nilai dari pohon pelacakan.

Prinsip dasar seperti yang dipaparkan oleh Russell and Norvig (1995) adalah bahwa algoritma *Alpha-Beta Pruning* selalu memelihara 2 variabel yaitu α (alfa) dan β (beta). α diasosiasikan dengan MAX yang tidak pernah turun dan β diasosiasikan dengan MIN yang tidak pernah naik. Adapun bila kita definisikankan pengertian tersebut dapat dijabarkan sebagai berikut:

- *Alpha*, atau dalam simbol (α) adalah pilihan nilai terbaik dari nilai tertinggi yang kita dapatkan dari banyak pilihan sepanjang lintasan untuk MAX.
- *Beta*, atau dalam simbol (β) adalah pilihan nilai terbaik dari nilai terendah yang kita dapatkan dari banyak pilihan sepanjang lintasan untuk MIN.
- Nilai *Alpha* dan *Beta* diperbarui selama pencarian.

Salah satu metode yang digunakan untuk memodifikasi algoritma *Minimax* adalah dengan *Alpha-Beta Pruning*. *Alpha-Beta Pruning* atau pemangkas alfa-beta adalah suatu peningkatan efisiensi yang dapat menghapuskan pencarian banyak anak dari pohon pelacakan. Mengapa dipangkas? Pemangkasan dilakukan untuk

mengeliminasi node pencarian yang berpotensi tidak dapat dicapai dan untuk mempercepat proses pencarian.

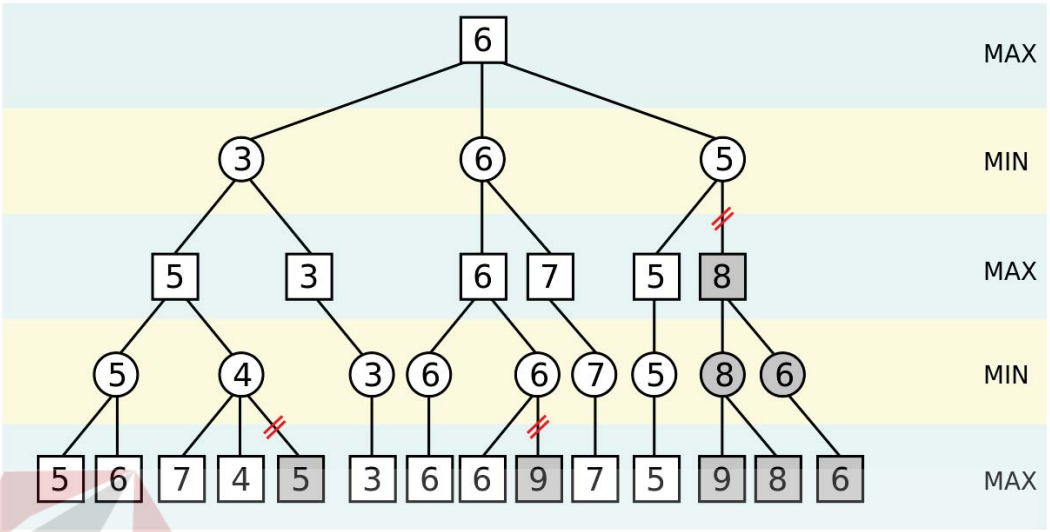
Berikut adalah pseudocode untuk algoritma *Alpha-Beta Pruning*.

```
minimaxAB(level, player, alpha, beta)
{
      if (gameover || level == 0)
      {
            return score
      }
      children = all valid moves for this "player"
      if (player is computer, i.e., max's turn)
      {
            for each child
            {
                  score = minimaxAB(level - 1, opponent, alpha,
                  beta)
                  if (score > alpha) alpha = score
                  if (alpha >= beta) break;  // beta cut-off
            }
            return alpha
      }
      else
      {
            for each child
            {
                  score = minimaxAB(level - 1, computer, alpha,
                  beta)
                  if (score < beta) beta = score
                  if (alpha >= beta) break;  // alpha cut-off
            }
            return beta
      }
}

minimaxAB(2, computer, -inf, +inf)
```

Variabel *alpha* (α) digunakan sebagai batas bawah *node* yang akan melakukan maksimisasi. Sedangkan variabel *beta* (β) digunakan sebagai batas atas untuk *node* yang akan melakukan minimisasi. Pada *node-node* yang melakukan minimasi, evaluasi akan dihentikan jika sudah didapat *node* anak yang memiliki nilai lebih kecil dibanding dengan batas bawah (α). Sedangkan pada *node-node*

yang melakukan maksimasi, evaluasi akan dihentikan jika sudah didapat *node* yang memiliki nilai lebih besar dibanding dengan batas atas (β).



Gambar 2.6 Contoh *Tree Alpha-Beta Pruning*
(Sumber: https://en.wikipedia.org/wiki/Alpha-beta_pruning)

2.4 Bahasa C++

Pada masa-masa permulaan komputer, bahasa komputer digunakan sebagai alat bantu untuk melakukan perhitungan-perhitungan telemetri. Pada saat itu, bahasa yang digunakan masih sangat primitif sekali karena masih berupa bahasa mesin yang hanya mengenal angka 0 dan 1. Bahasa mesin tersebut selanjutnya disederhanakan menjadi bahasa yang agak mudah dipahami. Hasil penyederhanaan bahasa ini disebut dengan bahasa *assembly* karena menghadirkan statemen-statemen khusus yang disebut dengan istilah *mnemonic* seperti ADD, MOV, JMP dan lain-lain. *Assembly* tergolong bahasa tingkat rendah (*low level language*).

Pada tahun 1969, laboratorium Bell AT&T di Muray Hill, New Jersey menggunakan bahasa *assembly* ini untuk mengembangkan sistem operasi UNIX.

Maksudnya adalah untuk membuat sistem operasi yang dapat bersifat '*programmer-friendly*'. Setelah UNIX berjalan, Ken Thompson, seorang pengembang sistem di laboratorium tersebut mengembangkan bahasa baru dengan nama bahasa B. Huruf B ini diambil dari BCPL (*Basic Combined Programming Language*). Bahasa B ini kemudian digunakan untuk menulis ulang atau merevisi sistem operasi UNIX. Oleh karena bahasa B ini masih bersifat interpret dan lambat, maka pada tahun 1971, sistem operasi UNIX kemudian ditulis ulang dengan menggunakan bahasa C, yaitu bahasa pemrograman yang dikembangkan oleh Dennis Ritchie, seorang pengembang sistem di laboratorium yang sama.

Sampai sekarang bahasa C masih digunakan untuk melakukan pengembangan-pengembangan program dan sistem-sistem operasi, diantaranya adalah sistem operasi Windows milik Microsoft. Alasan itulah yang menjadikan bahasa C sangat popular di dunia pemrograman, khususnya untuk industri perangkat lunak. Namun sayangnya, bahasa C merupakan bahasa yang masih tergolong susah untuk dipelajari karena masih bersifat prosedural murni. Untuk membentuk satu objek, kita harus melakukan banyak sekali penulisan kode. Hal ini tentu dapat dikatakan sebagai sebuah kelemahan. Untuk mengatasi masalah ini, pada tahun 1983, seorang doktor yang bernama Bjarne Stroustrup yang juga bekerja di laboratorium yang sama menciptakan bahasa baru yaitu bahasa C++ yang menjadi bahasa *hybrid* dari bahasa C.

Bahasa C++ didasarkan atas bahasa C sehingga kita dapat melakukan kompilasi program-program yang ditulis dalam bahasa C dengan menggunakan kompiler C++. Keistimewaan dari bahasa C++ adalah bahasa ini mendukung

pemrograman berarah objek atau yang lebih sering dikenal dengan istilah *Object Oriented Programming* (OOP).

Seperti telah dikemukakan sebelumnya bahwa kedua bahasa ini (C dan C++) merupakan bahasa yang sangat popular dalam dunia pengembangan perangkat lunak. Kedua bahasa ini digolongkan ke dalam bahasa tingkat menengah (*middle level language*). Di sisi lain, yaitu di sisi akademis, seorang professor yang bernama Niklaus Wirth di Politeknik Zurich, Swiss, mengembangkan bahasa tingkat tinggi (high level language) yag disebut dengan bahasa Pascal untuk mengajarkan algoritma kepada para mahasiswanya. Maka dari itu, di lingkungan akademis bahasa Pascal lebih popular dibandingkan dengan bahasa C atau C++. Berikut adalah pengelompokan tingkatan dari bahasa pemrograman (Raharjo, 2011).

Tabel 2.1 Pengelompokan Tingkatan Bahasa Pemrograman

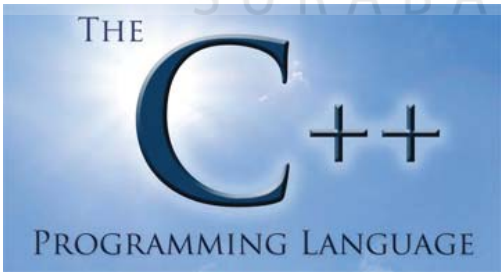
Tingkat	Bahasa Pemrograman
Tinggi	Ada
	Modula-2
	Pascal
	COBOL
	FORTRAN
	BASIC
Menengah	Java
	C++
	C
	FORTH
Rendah	Macro-Assembler
	Assembler

Dari tabel tersebut dapat dilihat bahwa bahasa pemrograman yang terdapat pada bagian paling atas merupakan bahasa pemrograman yang paling mudah untuk dipahami. Sebagai contoh, C adalah bahasa yang lebih sulit dipahami daripada bahasa C++ dan bahasa C++ lebih sulit dipahami daripada bahasa Java, dan seterusnya.

Semenjak dikembangkan, bahasa C dan C++ banyak dikembangkan untuk mengembangkan program-program aplikasi di bidang telekomunikasi, finansial atau bisnis dan sistem operasi. Bahkan sampai saat ini, pembuatan program-program untuk permainan komputer (*game*) sebagian besar masih menggunakan bahasa C/C++.

Menurut Bjarne Stroustrup, alasan mengapa bahasa C diambil sebagai bahasa dasar dari pembentukan bahasa C++ adalah sebagai berikut:

- Dapat dihubungkan dengan bahasa tingkat rendah
- Dapat berjalan di manapun dan untuk masalah apapun
- Dapat berjalan mulus dalam sistem operasi UNIX



Gambar 2.7 Logo Bahasa C++

(Sumber: https://isocpp.org/)

Perbedaan antara bahasa pemrograman C dan C++ meskipun bahasa-bahasa tersebut menggunakan sintaks yang sama tetapi mereka memiliki perbedaan, C merupakan bahasa pemrograman prosedural, dimana penyelesaian suatu masalah

dilakukan dengan membagi-bagi masalah tersebut ke dalam sub-sub masalah yang lebih kecil. Selain itu, C++ merupakan bahasa pemrograman yang memiliki sifat pemrograman berorientasi objek. Untuk menyelesaikan masalah, C++ melakukan langkah pertama dengan menjelaskan *class-class* yang merupakan anak *class* yang dibuat sebelumnya sebagai abstraksi dari objek-objek fisik. *Class* tersebut berisi keadaan *object*, anggota-anggotanya dan kemampuan dari *object*-nya. Setelah beberapa *Class* dibuat kemudian masalah dipecahkan dengan *Class*.

2.5 SDL (*Simple DirectMedia Layer*)

Gambar 2.8 Logo SDL
(Sumber: http://www.libsdl.org)

SDL (*Simple DirectMedia Layer*) adalah pustaka (*library)* pemrograman untuk membuat aplikasi multimedia di berbagai sistem operasi. Dengan menggunakan SDL, seorang programmer dapat mengakses layar, suara, *keyboard*, *joystick*, *hardware* 3D dan 2D *framebuffer* dengan menggunakan cara yang sama di berbagai sistem operasi. Kelebihan lain dari SDL adalah dapat digunakan dengan berbagai bahasa pemrograman.

Sesuai namanya, SDL didesain oleh Sam Latinga pada tahun 1998, sebagai pustaka pemrograman grafik yang sederhana. Pustaka inti SDL hanya menyediakan antarmuka pemrograman aplikasi (API, *application programming interface*) untuk:

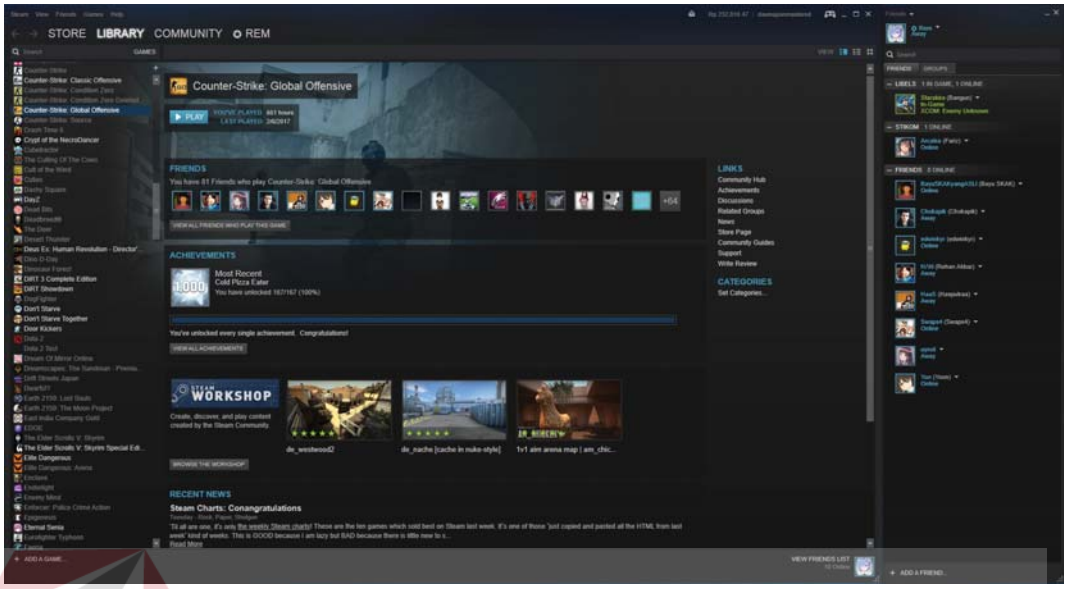
- Manipulasi piksel
- Operasi warna
- Suara
- Penanganan kejadian (*event*)
- Pewaktuan (*timing*)
- *Multithreading*
- Akses file

Pustaka tersebut dibuat multi-platform dengan membungkus API asli pada sistem operasi (*wrapper*). Selanjutnya di atas SDL, ada pustaka-pustaka tambahan seperti:

- SDL_image : untuk menangani berbagai format image.
- SDL_mixer : menyediakan operasi suara.
- SDL_net : dukungan *networking*.
- SDL_ttf : menyediakan operasi menulis teks dengan true type font.
- SDL_rtf : memanipulasi dokumen rich text format

SDL sendiri digunakan secara ekstensif di industri *game* baik proyek besar maupun kecil. Lebih dari 700 game, 180 aplikasi, dan 120 *demo* telah dicatatkan di situs pustaka SDL.

Contoh aplikasi populer yang menggunakan pustaka SDL sebagai penjembatan antara aplikasi tersebut dan permainan (*game*) itu sendiri adalah Steam. Steam adalah distributor permainan (*game*) digital milik Valve.



Gambar 2.9 Aplikasi Steam Milik Valve
(Sumber: http://www.libsdl.org)

Selain Steam, *game engine* milik Valve yaitu Source Engine, yang berkontribusi membangun *game-game* populer seperti Counter Strike, Dota, dan Team Fortress, juga menggunakan SDL.

Dalam Tugas Akhir ini, pustaka SDL versi 1.2.15 digunakan sebagai antarmuka aplikasi permainan *Tic Tac Toe*.

2.6 Microsoft Visual Studio 2015

Visual Studio merupakan salah satu *Integrated Development Environment* (IDE) yang paling banyak digunakan khususnya untuk mengembangkan aplikasi *mobile*, *desktop*, maupun *website*. Kompiler yang dimasukkan ke dalam paket Visual Studio antara lain Visual C++, Visual C#, Visual Basic, Visual Basic .NET, Visual InterDev, Visual J++, Visual J#, Visual FoxPro, dan Visual SourceSafe.

Microsoft Visual Studio dapat digunakan untuk mengembangkan aplikasi dalam *native code* (dalam bentuk bahasa mesin yang berjalan di atas Windows) ataupun *managed code* (dalam bentuk *Microsoft Intermediate Language* di atas .NET Framework).

Gambar 2.10 Logo Microsoft Visual Studio
(Sumber: https://en.wikipedia.org/wiki/Visual_studio/)

Selain itu, Visual Studio juga dapat digunakan untuk mengembangkan aplikasi Silverlight, aplikasi Windows Mobile (yang berjalan di atas .NET Compact Framework). Versi teranyar dari Visual Studio telah mampu untuk mengembangkan aplikasi pada *platform* Android.

Dalam Tugas Akhir ini, Visual Studio 2015 digunakan sebagai IDE untuk mengembangkan aplikasi.