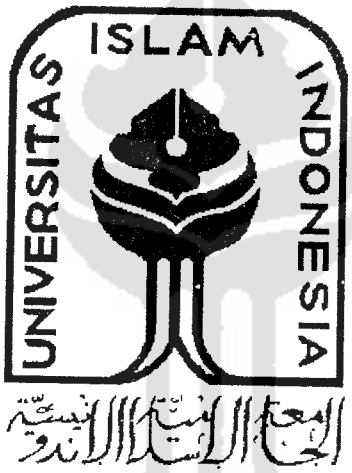
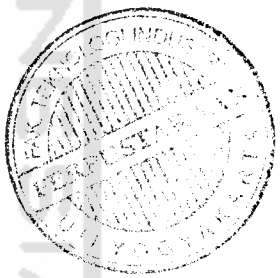
APLIKASI LOGICAL AGENT DENGAN STUDI KASUS PERMAINAN MINESWEEPER

TUGAS AKHIR

Diajukan Sebagai Salah Satu Syarat
Untuk Memperoleh Gelar Sarjana Jurusan Teknik Informatika





oleh:

Nama : Reza Ardhianto
No. Mahasiswa : 01 523 118

JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INDUSTRI
UNIVERSITAS ISLAM INDONESIA
YOGYAKARTA
2007

LEMBAR PENGESAHAN PEMBIMBING

APLIKASI LOGICAL AGENT DENGAN STUDI KASUS PERMAINAN MINESWEEPER

TUGAS AKHIR

oleh:

Nama : Reza Ardhianto

No. Mahasiswa : 01 523 118

Yogyakarta, 15 Februari 2007

Pembimbing,

Taufiq Hidayat, ST, MCS

LEMBAR PENGESAHAN PENGUJI

APLIKASI LOGICAL AGENT DENGAN STUDI KASUS PERMAINAN MINESWEEPER

TUGAS AKHIR

oleh:

Nama : Reza Ardhianto

No. Mahasiswa : 01 523 118

Telah Dipertahankan di Depan Sidang Penguji sebagai Salah Satu Syarat
untuk Memperoleh Gelar Sarjana Jurusan Teknik Informatika
Fakultas Teknologi Industri Universitas Islam Indonesia

Yogyakarta, 6 Maret 2007

Tim Penguji

Taufiq Hidayat, ST, MCS
Ketua

Sri Kusumadewi , S.Si, MT
Anggota I

Syarif Hidayat, S.Kom
Anggota II

Mengetahui,
Ketua Jurusan Teknik Informatika
Fakultas Teknologi Industri
Universitas Islam Indonesia

Yudi Prayudi, S.Si, M.Kom

Persembahanku...

Aku Bersembah sujud hanya kepada-Mu

Ya Allah Hanya Engkau yang patut disembah

Ya Allah hanya dengan rahmat-Mu dan ridho-Mu hamba mampu Ya Allah

Tiada yang dapat meraih ilmu tanpa ridho dan petunjuk-Mu

Untukmu...

Ibuku , kakak dan adik - adikku yang selalu menyayangiku

Alamarhum Ayahku Ir. H. Sukamas semoga engkau selalu mendapatkan limpahan rahmat dan ampunan serta bahagia disisi-Nya

Untuk agamaku, insan TI, bangsaku ...

Semoga apa yang telah kuselesaikan ini bermanfaat bagimu

بسم الله الرحمن الرحيم

Tuntutlah ilmu sampai ke negeri cina

Tiada berguna ilmu tanpa diamalkan

Aja emban sinde emban siladan

Durung menang yen durung wani kalah. Durung unggul yen durung wani asor.

Durung gedhe yen durung ngaku cilik

KATA PENGANTAR

Assalamu'alaikum Warohmatullahi Wabarokatuh.

Puji syukur kami panjatkan kehadirat Allah SWT. Karena berkat Rahmat dan hidayahNya sehingga Tugas Akhir yang berjudul " APLIKASI LOGICAL AGENT DENGAN STUDI KASUS PERMAINAN MINESWEEPER" dapat terselesaikan.

Tugas Akhir ini diajukan sebagai salah satu syarat untuk memperoleh gelar sarjana Teknik Informatika. Penulis dalam kesempatan ini mengucapkan terima kasih kepada :

1. Ibunda tercinta Hj. Sumarni, yang selalu mencurahkan kasih sayang, dukungan, dan kesabarannya selama ini.
2. Kakakku Anang, adikku Ayu dan Dhany yang selalu menjadi penyemangat dan sumber inspirasiku.
3. Bapak Taufik Hidayat, ST, MCS, Dosen Pembimbing yang selalu dengan sabar membantu dalam penyelesaian Tugas Akhir ini.
4. Teman – teman seperjuangan Nugroho, Yudi, Nanda, Krisna terima kasih atas dukungan dan semangat kalian, tetap semangat dan terus maju pantang mundur.
5. Teman – teman Informatika '01, terimakasih atas semangat dan persahabatan kalian.
6. Adik – adik kosku yang selalu membuat hari – hariku selalu ceria Deddy, Aji, Harry, Rendra, Uut, Bos "Baroto", dan seluruh penghuni Kos Muslim "Ceria". Kalian semua memang tiada duanya.

7. Semua teman dan kerabat yang tidak bisa disebutkan satu persatu, terima kasih atas segala bantuan dan dukungan kalian semua.

Dalam penyelesaian Tugas Akhir ini penulis menyadari bahwa masih banyak terdapat kesalahan dan kekurangannya, untuk itu penulis mengharapkan kritik dan saran yang membangun agar bisa berguna untuk masa mendatang.

Akhir kata, Semoga Tugas Akhir ini dapat berguna bagi semua insan IT untuk kepentingan agama, nusa dan bangsa.

Amien

Assalamu'alaikum Warohmatullahi Wabarokatuh

Yogyakarta, 15 Februari 2007

Penyusun

SARI

Kecerdasan buatan adalah salah satu bagian dari ilmu komputer yang bertujuan untuk membuat komputer dapat berpikir dan bertindak seperti manusia. Dengan berkembangnya bidang ini, kini manusia dapat memecahkan berbagai masalah dengan menggunakan bantuan komputer.

Logical Agent merupakan salah satu aplikasi di bidang kecerdasan buatan. *Agent* adalah sebuah sistem atau program komputer yang dikondisikan berada dalam suatu lingkungan dan mampu melakukan suatu analisa serta tindakan terhadap masalah yang dihadapi.

Permainan *Minesweeper* adalah salah satu masalah dimana *Logical Agent* dapat diaplikasikan untuk menyelesaikannya. Informasi yang bisa didapatkan agen adalah nilai angka petunjuk dari kotak yang telah terbuka. Angka tersebut menunjukkan jumlah ranjau yang berada di sekitarnya. Tujuan agen adalah menyelesaikan permainan dengan membuka semua kotak yang bukan merupakan lokasi ranjau. Metode yang digunakan dalam pemecahan masalah permainan *Minesweeper* adalah *Constraint Propagation*.

Aplikasi dibuat menggunakan bahasa pemrograman *Pascal* dengan *Compiler* menggunakan Borland Delphi 7.

Kata kunci : *Logical Agent*, *Constraint Propagation*.

TAKARIR

agent	perwakilan
artificial intelligence	kecerdasan buatan
array	larik
autoplay	bermain secara otomatis
beginner	pemula
compiler	aplikasi untuk mengompile suatu file
constraint	batas
constraint collector	pengumpul constraint
constraint propagation	penelusuran batas
constraint satisfaction problem	permasalahan bagaimana memenuhi suatu constraint
digital assistant	asisten dalam bentuk digital (program)
domain	kumpulan nilai yang dapat diambil oleh variabel
expert	ahli
freeware	alat / program yang dapat digunakan tanpa harus membeli
interface	antarmuka
intermediate	pertengahan
logical ability	kemampuan melakukan proses logika
logical agent	agen yang mempunyai kemampuan logika

minesweeper	permainan mencari ranjau pada suatu papan permainan
propagator	penelusur
propagator engine	mesin penelusur
shortcut	jalan pintas
solver engine	mesin pemecahan untuk membantu agen
update	memperbaharui
variabel	faktor tidak tetap
zero propagator	penelusur angka nol

DAFTAR ISI

DAFTAR GAMBAR

BAB I

PENDAHULUAN

1.1 Latar Belakang

Perkembangan teknologi informasi yang sangat pesat telah membawa perubahan dalam kehidupan manusia. Kini manusia dapat memecahkan berbagai masalah yang dihadapi dengan bantuan komputer. Hal ini dimungkinkan karena berkembangnya salah satu bagian dari ilmu komputer yaitu *Artificial Intelligence* atau kecerdasan buatan yang dapat membuat komputer berpikir dan bertindak seperti yang dilakukan oleh manusia. *Logical agent* merupakan salah satu aplikasi di bidang kecerdasan buatan.

Agent adalah sebuah sistem atau program komputer yang dikondisikan berada dalam suatu lingkungan dan mampu melakukan suatu analisa serta tindakan terhadap masalah yang dihadapi. Agen dapat memecahkan masalah dengan mengolah data yang sudah ada baik dari data internal yang terdapat pada agen itu sendiri maupun data eksternal yang terdapat pada masalah yang akan dipecahkan sehingga tidak memerlukan intervensi langsung dari manusia untuk melakukan suatu analisa dan tindakan. Teknologi *Agent* saat ini sudah banyak dipakai oleh beberapa pengembang perangkat lunak, seperti Microsoft yang menggunakan teknologi *Microsoft Agent* dalam beberapa produknya untuk membantu para pemakainya.

Salah satu masalah dimana *logical agent* dapat diaplikasikan untuk memecahkannya yaitu penyelesaian permainan *Minesweeper*. Masalah yang

dihadapi oleh agen pada permainan *Minesweeper* adalah bagaimana menemukan lokasi semua ranjau yang terdapat pada papan permainan tanpa membuat ranjau – ranjau tersebut meledak. Petunjuk yang dapat digunakan oleh agen adalah angka yang terdapat pada kotak aman yang telah dibuka. Angka tersebut menunjukkan jumlah ranjau yang terdapat di sekitarnya. Permainan akan berakhir apabila semua lokasi ranjau telah ditemukan atau salah satu kotak yang berisi ranjau dibuka.

Logical agent juga dapat diaplikasikan dalam permainan *Minesweeper* untuk membantu pemain pemula dalam menyelesaikan permainan dengan memberikan prediksi lokasi ranjau dan kotak aman yang boleh dibuka. Agen akan memberikan tanda apakah kotak tersebut aman atau terdapat ranjau di dalamnya.

1.2 Rumusan Masalah

Berdasarkan latar belakang masalah yang telah disebutkan maka dapat ditarik rumusan masalah yaitu membuat dan mengaplikasikan *logical agent* untuk menyelesaikan permainan *Minesweeper*.

1.3 Batasan Masalah

Untuk menghindari meluasnya materi yang dibahas pada penelitian ini, maka diberikan batasan masalah sebagai berikut :

1. Pembuatan permainan *Minesweeper* dengan fitur yang sama dengan permainan *Minesweeper* yang terdapat pada Ms. Windows.
2. Penambahan fitur / alat bantu yang merupakan aplikasi dari *logical agent* yang bisa digunakan untuk menyelesaikan permainan dan membantu pemain.

3. Hanya ada satu agen dan mempunyai tujuan menyelesaikan permainan dengan membuka semua kotak aman dan menemukan lokasi ranjau.
4. Metode yang digunakan agen untuk memecahkan masalah adalah metode *Constraint Propagation*.

1.4 Tujuan Penelitian

Tujuan akhir yang akan dicapai dalam penelitian ini adalah sebagai berikut':

1. Untuk memberikan penjelasan tentang *logical agent* dan aplikasinya.
2. Memberikan solusi terhadap masalah yang dihadapi dalam penyelesaian permainan *Minesweeper*.

1.5 Manfaat Penelitian

Manfaat yang didapat dari penelitian ini baik untuk penulis maupun pembaca adalah sebagai berikut :

1. Membantu dalam menyelesaikan permainan *Minesweeper* bagi pemain pemula.
2. Memberikan pengalaman dan pengetahuan bagi penulis dalam bidang kecerdasan buatan khususnya tentang *logical agent* serta langkah – langkah dalam pembuatannya untuk menyelesaikan permainan *Minesweeper*.

1.6 Metodologi Penelitian

Metode – metode yang digunakan untuk menyelesaikan penelitian ini adalah metode pengumpulan data, metode analisis, dan tahapan pengembangan perangkat lunak.

1.6.1 Metode Pengumpulan Data

Metode pengumpulan data adalah metode yang digunakan untuk mencari dan mengumpulkan data atau teori yang berhubungan dengan penelitian yang dilakukan. Metode yang dilakukan dengan mempelajari teori – teori yang berhubungan dengan penelitian serta literatur – literatur lain yang dapat membantu untuk memecahkan masalah yang berkaitan dengan penelitian serta dengan melakukan konsultasi secara berkesinambungan dengan dosen pembimbing.

1.6.2 Metode Analisis

Untuk melakukan analisis digunakan metode analisis yang berkaitan dengan penelitian yang dilakukan, dan menggunakan analisis yang digunakan dalam penelitian sejenis sehingga yang dilakukan tidak bertentangan dengan penelitian sejenis lainnya.

1.6.3 Tahapan Pengembangan Perangkat Lunak

Metode pengembangan perangkat lunak disusun berdasarkan hasil dari yang sudah diperoleh, meliputi :

1. Analisis Kebutuhan dan Perancangan. Pada tahap ini dilakukan proses analisis terhadap berbagai kebutuhan yang mungkin diperlukan oleh sistem yang akan dibangun dan dilanjutkan dengan proses perancangan aplikasi perangkat lunak.
2. Proses Implementasi dan Pengujian. Implementasi dilakukan setelah semua bagian dalam tahap perancangan sudah layak dilanjutkan menuju

proses implementasi. Selama implementasi, pada tiap – tiap bagian tertentu dilakukan proses pengujian secara bertahap hingga pada akhirnya seluruh hasil implementasi telah mengalami pengujian dengan baik.

3. Analisis Kerja. Tahapan ini dilakukan untuk menguji dan mengevaluasi secara keseluruhan kinerja perangkat lunak yang dibuat. Dari hasil analisis kinerja dapat dilihat kesesuaian rancangan dan hasil akhir yang dihasilkan.

1.7 Sistematika Penelitian

Sistematika penulisan laporan berguna untuk memberikan gambaran umum dari keseluruhan isi laporan serta untuk mempermudah pembacaan agar lebih jelas dan akurat. Sistematika penulisan dan garis besar isi laporan ini adalah sebagai berikut :

1. BAB I Pendahuluan

   Menjelaskan tentang latar belakang masalah, gambaran umum permasalahan yang dihadapi beserta batasan masalah yang menjadi tolak ukur penulisan dalam melakukan penelitian rumusan masalah, tujuan penelitian yang ingin dicapai, manfaat dari apa yang akan didapatkan dari tugas akhir ini serta metodologi penelitian dan sistematika penulisan yang digunakan.

2. BAB II Landasan Teori

   Menjelaskan tentang teori – teori yang digunakan dan relevan dengan topik tugas akhir dimulai dengan teori mengenai *logical agent*, metode *Constraint Propagation*, sampai dengan teori yang menjelaskan tentang perangkat lunak yang akan digunakan untuk mendukung implementasi *logical agent* pada permainan *Minesweeper*.

3. BAB III Analisis Kebutuhan Perangkat Lunak

   Bab ini memuat tentang pembahasan aplikasi *Minesweeper* dan metode analisis kebutuhan perangkat lunak yang digunakan berupa masukan, keluaran, dan antar muka yang harus dipenuhi oleh sistem.

4. BAB IV Perancangan Perangkat Lunak

   Bab ini memuat uraian tentang metode perancangan perangkat lunak yang menjelaskan mengenai pembuatan bentuk dari perancangan sistem yang akan diterapkan di lapangan sehingga apa yang dirancang benar – benar sesuai dengan apa yang dibutuhkan. Meliputi metode *Constraint Propagation*, dan memuat hasil perancangan perangkat lunak yang merupakan terjemahan kebutuhan perangkat lunak.

5. BAB V Implementasi Perangkat Lunak

   Bab ini memuat batasan implementasi perangkat lunak yaitu asumsi – asumsi yang dipakai, lingkungan pengembangan, bahasa dan kompilator yang dipakai beserta alasan pemilihan, dan batasan lain yang ditemui selama pengembangan.

6. BAB VI Analisis Kinerja Perangkat Lunak

   Bab ini memuat dokumentasi hasil pengujian terhadap perangkat lunak yang dibandingkan dengan kebenaran dan kesesuaiannya serta bagaimana kelebihan dan kekurangannya dapat digunakan untuk peneliti lebih lanjut.

7. BAB VII Penutup

   Bab ini memuat kesimpulan – kesimpulan dari proses pengembangan perangkat lunak, baik pada tahap analisis kebutuhan, perancangan,

implementasi, dan analisis kinerja. Juga berisi saran yang perlu diperhatikan berdasar keterbatasan – keterbatasan yang ditemukan dan asumsi – asumsi yang dibuat selama melakukan TA.

BAB II

LANDASAN TEORI

2.1 Agent

Penelitian mengenai *Agent* sudah dimulai oleh para peneliti di bidang kecerdasan buatan sejak 10 tahun yang lalu. Belum ada definisi yang disetujui secara universal mengenai *agent*, tetapi di dalam penelitian ini digunakan definisi dari Jennings dan Wooldridge. *Agent* adalah sebuah sistem komputer yang dikondisikan berada dalam suatu lingkungan dan mampu untuk melakukan suatu analisa serta tindakan terhadap suatu masalah untuk mencapai tujuan yang telah ditetapkan. Definisi tersebut didasarkan pada beberapa konsep, yaitu *autonomy*, *situatedness*, dan *flexibility* [AD01].

Autonomy, agen mempunyai kemampuan untuk melakukan suatu aksi tanpa adanya intervensi langsung dari manusia maupun agen yang lain serta dapat mengontrol tindakan dan perintah – perintah internalnya sendiri.

Situatedneess, agen dikondisikan berada dalam suatu lingkungan dan dapat mempelajari serta memanipulasi keadaannya baik secara sebagian maupun secara menyeluruh.

Flexibility, agen mempunyai beberapa properti, yaitu *responsiveness*, *pro-activity*, dan *social ability*.

Responsiveness, agen mengamati lingkungannya dan memberikan respon setiap kali terjadi perubahan.

Pro-activity, agen tidak hanya merespon perubahan yang terjadi pada lingkungannya, akan tetapi dapat melakukan suatu inisiatif yang berorientasi pada tujuan yang akan dicapai.

Social ability, agen harus dapat berinteraksi dengan agen lain atau manusia untuk memecahkan suatu masalah yang dihadapi dan membantu tindakan mereka.

Pada perkembangannya, banyak istilah yang digunakan untuk menyebut *Agent* yang tergantung pada metode yang digunakan dan fungsinya dalam memecahkan suatu permasalahan yang lebih spesifik. Istilah – istilah tersebut diantaranya *Learning Agent*, *Reasoning Agent*, *Logical Agent*, dan lain sebagainya.

Dalam studi kasus permainan *Minesweeper*, yang dimaksud dengan *Logical Agent* adalah robot / program yang mempunyai *logical ability* dan bertugas untuk menyelesaikan permainan dengan membuka semua kotak yang aman dan menemukan semua lokasi ranjau yang terdapat pada papan permainan.

2.2 Constraint Propagation

Constraint Propagation adalah sebuah metode yang lebih menekankan pada penetapan suatu solusi yang harus dipenuhi berdasarkan sejumlah batasan atau kondisi yang telah dideskripsikan daripada melakukan sekumpulan langkah untuk mencari sebuah solusi yang belum ditetapkan [WI06].

Pemodelan masalah yang dihadapi dalam metode *Constraint Propagation* biasa disebut dengan istilah *Constraint Satisfaction Problem (CSP)*. Pemecahan dari *CSP* adalah dengan memberikan nilai kepada setiap variabel agar dapat memenuhi semua *constraint* yang ada [WI06a].

Dalam metode *Constraint Propagation* terdapat tiga elemen penting, yaitu *Domain, Constraint,* dan *Variable*.

Domain adalah sekumpulan nilai yang disediakan untuk dapat diambil oleh *Variable* dalam menyelesaikan suatu permasalahan. Dalam studi kasus permainan *Minesweeper*, *domain* yang disediakan hanya bernilai 1 atau 0. Sedangkan kotak – kotak yang terdapat pada papan permainan merupakan *Variabel* yang masing - masing akan mengambil nilai 1 apabila terdapat ranjau di dalamnya dan 0 apabila sebaliknya.

Constraint merupakan sebuah relasi logika dari sejumlah variabel yang mengambil nilainya masing – masing dari dalam sebuah *domain* yang telah ditentukan. *Constraint* akan membatasi variabel dalam mengambil nilai yang tersedia di dalam domain. Nilai tersebut akan mempresentasikan sebagian informasi tentang tujuan yang akan dicapai.

Sifat – sifat dari *constraint* antara lain :

1. Dapat memberikan informasi secara *partial*.

   Constraint tidak perlu untuk secara khusus menentukan nilai bagi variabel – variabelnya. (Contoh: *constraint* X > 2 tidak menentukan secara pasti nilai dari variabel X, jadi X dapat bernilai 3, 4, 5, ... dst.).

2. *Heterogeneous*.

   Constraint dapat mendeklarasikan relasi antar variabel dengan *domain* yang berbeda. (Contoh : X = panjang(Y)).

3. *Non-directional*.

Sebuah *constraint*, misal mempunyai dua variabel X dan Y yang masing – masing dapat digunakan untuk mendeklarasikan sebuah *constraint* untuk digunakan oleh satu dengan yang lainnya. (Contoh : X = Y + 2 dapat digunakan untuk mengkomputasi variabel X menggunakan X := Y + 2, sama seperti variabel Y menggunakan Y := X – 2).

4. *Declarative*.

   Constraint menentukan hubungan yang harus dipertahankan tanpa mendeklarasikan sebuah prosedur komputasional untuk memaksakan hubungan tersebut.

5. *Additive*.

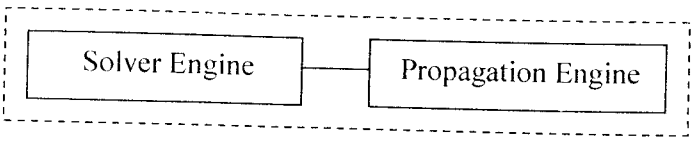
   Urutan dari *constraints* yang akan dikenakan tidak dipermasalahkan, yang penting adalah pada akhirnya konjungsi dari *constraints* tersebut dapat memberikan efek yang diinginkan.

6. *Rarely independent*.

   Kumpulan *constraint* dalam penyimpanan dapat saling berbagi variabel.

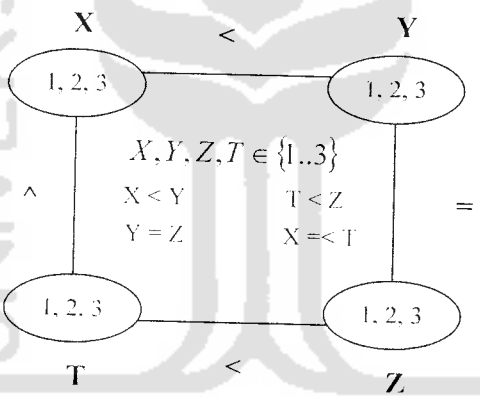
2.2.1 Constraint Propagation Engine

Constraint Propagation Engine terdiri dari 2 bagian, yaitu *Propagation Engine* dan *Solver Engine*. Fungsi dari *Propagator Engine* adalah melakukan penelusuran *constraint* untuk menentukan nilai sebuah variabel. Sedangkan fungsi dari *Solver Engine* adalah mencari asumsi – asumsi untuk membantu *Propagator Engine* dalam memecahkan masalah apabila menemui kebuntuan.



Gambar 2.1 Arsitektur *Constraint Propagation Engine*

Gambar 2.2 menjelaskan contoh penyelesaian *Constraint Satisfaction Problem* menggunakan metode *Constraint Propagation*. Pada contoh tersebut telah ditentukan *domain*, *constraint*, dan variabel – variabelnya. Persamaan yang menunjukkan nilai dari domain adalah $X, Y, Z, T \in \{1..3\}$. Variabel X, Y, Z, dan T dapat mengambil nilai antara 1 sampai dengan 3 asalkan dapat memenuhi *constraint*-nya, yaitu X < Y; T < Z; Y = Z; dan X =< T.



Gambar 2.2 Contoh penyelesaian *Constraint Satisfaction Problem*

2.3 Minesweeper

Minesweeper adalah sebuah permainan yang sangat populer terutama bagi para pengguna sistem operasi Ms. Windows. Sebuah papan permainan yang merepresentasikan sebuah ladang ranjau diberikan kepada pemain pada awal permainan. Setiap kotak yang terdapat pada papan permainan dapat menyembunyikan paling banyak 1 buah ranjau didalamnya. Jumlah total ranjau

yang terdapat pada papan permainan diketahui oleh pemain. Tujuan dari permainan ini adalah untuk membuka semua kotak yang tidak terdapat ranjau di dalamnya sehingga lokasi ranjau yang tersebar secara acak pada papan permainan dapat ditemukan. Pemain akan mendapatkan petunjuk setiap berhasil membuka kotak aman, berupa angka – angka yang menunjukkan jumlah ranjau yang terdapat di sekitarnya. Sebaliknya, jika pemain membuka kotak yang terdapat ranjau di dalamnya, ranjau akan meledak dan permainan berakhir.

BAB III

ANALISIS KEBUTUHAN

3.1 Metode Analisis

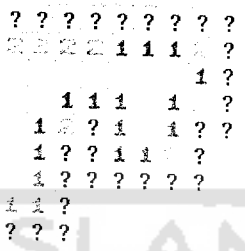
Metode analisis yang diperlukan sebagai bahan acuan adalah analisis dengan pendekatan literatur yang dilengkapi dengan bahan dan pemahaman terhadap sistem sehingga hasil analisis dapat dikembangkan dan diimplementasikan ke dalam sistem.

Metode analisis yang digunakan dalam sistem ini adalah pemodelan masalah menggunakan *Constraint Satisfaction Problem (CSP)* dan pemecahannya dengan menggunakan metode *Constraint Propagation*.

3.2 Analisis Masalah

Masalah yang dimunculkan dalam kasus ini adalah penyelesaian permainan *Minesweeper*, dimana agen harus menemukan semua lokasi ranjau yang terdapat pada papan permainan dengan membuka semua kotak aman. Persoalan penyelesaian permainan diangkat menjadi pokok permasalahan karena terbatasnya informasi yang diperoleh oleh agen tentang kotak mana saja yang harus dibuka dan menentukan lokasi ranjau tanpa membuatnya meledak, sehingga permainan dapat diselesaikan. Agen akan mendapatkan informasi yang lebih lengkap apabila telah menjalankan permainan. Informasi tersebut berupa angka – angka yang terdapat pada kotak aman yang telah dibuka, dan menunjukkan jumlah ranjau yang terdapat di sekitar kotak tersebut. Informasi lain yang bisa di dapatkan oleh

agen adalah jumlah ranjau yang sudah ditemukan dan jumlah sisanya untuk membuat suatu asumsi atau perkiraan lokasi ranjau.



Gambar 3.1 Masalah penentuan lokasi ranjau pada *Minesweeper*

Dari permasalahan tersebut, dapat ditarik poin – poin sebagai berikut :

1. Agen hanya memiliki informasi yang terbatas mengenai kotak aman yang harus dibuka dan menentukan lokasi ranjau.
2. Agen harus menyelesaikan permainan dengan membuka semua kotak aman dan menemukan lokasi ranjau tanpa membuatnya meledak.
3. Informasi tambahan dapat diperoleh agen setelah menjalankan permainan. Informasi tersebut berupa angka yang menunjukkan jumlah ranjau disekitarnya, jumlah ranjau yang telah ditemukan serta jumlah sisanya yang dapat digunakan untuk membuat suatu asumsi.

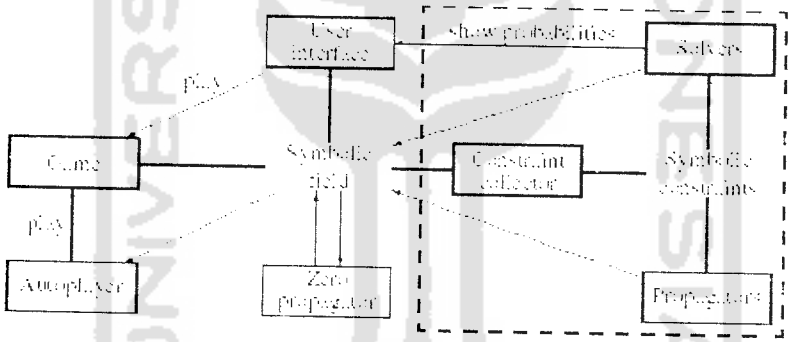
3.3 Hasil Analisis

3.3.1 Gambaran Umum Sistem

Secara garis besar, sistem yang akan diwujudkan nantinya adalah penyelesaian permainan *Minesweeper* oleh agen dengan cara membuka semua kotak aman dan menentukan lokasi ranjau yang terdapat pada papan permainan.

Agen juga dapat digunakan oleh pemain pemula untuk membantu menyelesaikan permainan.

Gambaran umum dari sistem yang diharapkan adalah agen dapat membuka semua semua kotak aman dan menentukan lokasi ranjau dengan bantuan angka – angka yang terdapat pada kotak aman yang telah dibuka sebagai informasi. Agen akan mendapatkan informasi tambahan berupa angka petunjuk setiap kali berhasil membuka kotak aman serta jumlah ranjau yang tersisa pada papan permainan. Dengan informasi tersebut, agen dapat membuat suatu asumsi untuk melakukan aksi selanjutnya untuk menyelesaikan permainan.



Gambar 3.2 Arsitektur Sistem

Gambar 3.2 menunjukkan arsitektur sistem yang akan dibangun. Simbol – simbol yang terdapat pada papan permainan akan mencerminkan kondisi permainan. Simbol tersebut akan dikumpulkan oleh *constraint collector* untuk kemudian diproses oleh *propagator* dan *solver*. Setelah *constraint* selesai diproses, kemudian akan menghasilkan kemungkinan / prediksi – prediksi yang akan ditunjukkan oleh agen pada antarmuka (*interface*) permainan. Jika fitur *autoplay* diaktifkan, maka secara otomatis kotak yang aman akan dimainkan oleh agen.

3.3.2 Analisis Kebutuhan Masukan

Sistem yang akan dibangun membutuhkan masukan – masukan sebagai berikut :

1. Informasi kondisi papan permainan. Pemain harus mengawali permainan sehingga sudah ada kotak yang terbuka, karena agen melakukan aksi berdasarkan angka petunjuk pada papan permainan.
2. Jumlah total ranjau yang terdapat pada papan permainan, jumlah yang telah ditemukan, dan jumlah sisanya diketahui oleh agen. Informasi ini juga digunakan untuk membuat suatu asumsi pengambilan langkah dalam permainan.
3. Informasi tambahan akan diperoleh agen setiap kali berhasil membuka kotak aman dan menemukan lokasi ranjau.

3.3.3 Analisis Kebutuhan Keluaran

Sistem yang akan dibangun membutuhkan keluaran – keluaran sebagai berikut :

1. Solusi dari permainan *Minesweeper*, yaitu menemukan semua lokasi ranjau yang terdapat pada papan permainan tanpa membuatnya meledak. Agen akan memberi tanda setiap lokasi ranjau yang ditemukan.
2. Pada saat agen digunakan oleh pemain pemula sebagai alat bantu, agen akan memberi tanda pada kotak yang aman untuk dibuka oleh pemain dan memberi tanda pada kotak yang di dalamnya terdapat ranjau.

3.3.4 Kinerja yang harus dipenuhi

Sistem yang akan dibangun diharapkan dapat memenuhi kebutuhan – kebutuhan antara lain :

1. Mampu menyelesaikan permainan *Minesweeper* dengan mengolah dan menggunakan informasi yang terdapat pada papan permainan.
2. Dapat membantu pemain pemula untuk menyelesaikan permainan dengan memberikan petunjuk kotak aman yang boleh dibuka dan lokasi ranjau pada papan permainan.

3.3.5 Fungsionalitas yang dikehendaki

Diharapkan agen mampu untuk mengenali dan mengolah informasi yang didapat untuk membuka semua kotak aman dan menemukan lokasi ranjau, sehingga masalah penyelesaian permainan *Minesweeper* dapat dipecahkan.

3.3.6 Antarmuka yang diinginkan

Sistem yang dibangun diharapkan dapat memberikan antarmuka yang *user friendly* dan menarik sehingga memudahkan pengguna untuk menggunakannya. Setiap angka petunjuk yang berbeda pada papan permainan akan ditampilkan dengan warna yang berbeda pula. sehingga diharapkan dapat memperkecil kemungkinan pemain untuk melakukan kesalahan dalam mengenali angka – angka tersebut. Antarmuka juga akan dilengkapi dengan beberapa menu yang dapat dipilih oleh pemain dengan menggunakan *mouse* maupun dengan kode *shortcut* menggunakan keyboard.

BAB IV

PERANCANGAN

4.1 Metode Perancangan

Metode perancangan yang digunakan pada perancangan *Logical Agent* ini adalah pemodelan masalah menggunakan *Constraint Satisfaction Problem (CSP)* dan pemecahannya menggunakan metode *Constraint Propagation*. Seperti yang telah disampaikan pada bab III subbab metode analisis bahwa masalah penyelesaian permainan *Minesweeper* dapat dimodelkan menggunakan *CSP* dengan melibatkan tiga elemen, yaitu *Domain*, *Constraint*, dan *Variable*.

4.2 Hasil Perancangan

Hasil perancangan meliputi perancangan antarmuka, serta metode *Constraint Propagation* yang digunakan untuk memecahkan masalah yang telah dimodelkan ke dalam *Constraint Satisfaction Problem*.

4.2.1 Pemodelan Constraint Satisfaction Problem

Dalam pemodelan masalah menggunakan *Constraint Satisfaction Problem*, setiap variabel akan mengambil nilai dari *domain* untuk memenuhi *constraint* / batasan yang telah ditetapkan.

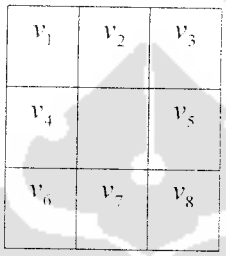
4.2.1.1 Domain

Nilai – nilai yang disediakan oleh *domain* pada kasus permainan *Minesweeper* hanya bernilai 1 atau 0. Nilai 1 akan diberikan kepada variabel

(kotak – kotak pada papan permainan) apabila di dalamnya terdapat ranjau, dan nilai 0 apabila sebaliknya.

4.2.1.2 Variabel

Variabel untuk membentuk suatu *constraint* didapatkan dari 8 kotak yang berada di sekitar angka petunjuk yang didapatkan ketika membuka kotak yang aman seperti yang dijelaskankan pada gambar 4.1.



Gambar 4.1 Delapan kotak yang mengitari angka petunjuk

4.2.1.3 Constraint

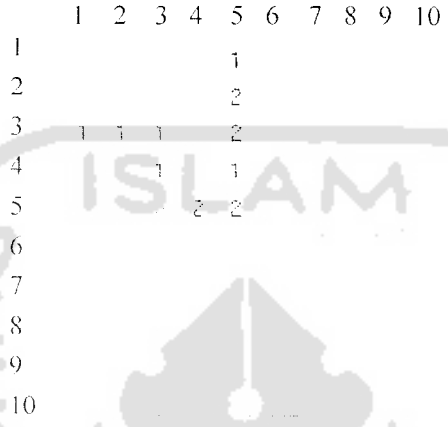
Langkah selanjutnya, *constraint collector* akan mengumpulkan petunjuk dari kondisi papan permainan untuk membentuk *constraint*. Persamaan yang dibentuk dari 8 kotak yang berada di sekitar angka petunjuk adalah :

$$(i, j) \quad v_1 + v_2 + \ldots + v_8 = K$$

(i, j) merupakan koordinat dari kotak yang mengandung angka petunjuk, dan $v_1 \ldots v_8$ adalah kotak – kotak yang berada di sekitarnya. K menunjukkan jumlah ranjau yang berada di sekitar kotak (i, j).

Pada contoh papan *Minesweeper* berukuran 10 X 10 dengan 20 ranjau seperti yang terdapat pada gambar 4.2. setiap kotak diidentifikasi berdasarkan

koordinatnya *(baris, kolom)*. Kotak (1,1), (1,2), (1,3), (1,4), (2,1), (2,2), (2,3), (2,4), (3,4), dan (4,4) telah dibuka dan tidak terdapat ranjau disekitarnya.



Gambar 4.2 Papan Minesweeper dengan 20 ranjau

Dari informasi yang telah diperoleh pada gambar 4.2, akan menghasilkan *constraint* sebagai berikut :

$$\sum i, j \in \{1,....,10\} x_{ij} = 20$$

(1,5) $x_{16} + x_{26} = 1$

(2,5) $x_{16} + x_{26} + x_{36} = 2$

(3,1) $x_{41} + x_{42} = 1$

(3,2) $x_{41} + x_{42} = 1$

(3,3) $x_{42} = 1$

(3,5) $x_{26} + x_{36} + x_{46} = 2$

(4,2) $x_{42} + x_{52} = 1$

(4,5) $x_{56} + x_{46} + x_{56} = 1$

(5,3) $x_{42} + x_{52} + x_{62} + x_{63} + x_{64} = 3$

(5,4) $x_{63} + x_{64} + x_{65} = 2$

(5,5) $x_{46} + x_{56} + x_{64} + x_{65} + x_{66} = 2$

Pada kumpulan *constraint* di atas, *constraint* $\sum i, j \in \{1,....,10\} x_{ij} = 20$ menunjukkan jumlah total ranjau yang terdapat pada papan permainan, dan

diasumsikan *"zero constraint"* (*constraint* yang menghasilkan angka 0) seperti pada kotak (1, 1) telah di propagasi.

4.2.2 Constraint Propagation

Setelah *constraint* terbentuk, langkah selanjutnya yaitu memberikan nilai pada setiap variabel agar dapat memenuhi *constraint*-nya masing – masing. Pemberian nilai kepada suatu variabel yang terdapat pada suatu *constraint* tidak boleh bertentangan dengan *constraint* yang lainnya (*arc consistency*).

Untuk mencegah agar tidak terjadi *arc inconsitency*, maka dilakukan proses penelusuran *constraint* (*constraint propagation*) untuk memastikan bahwa nilai suatu variabel sama untuk setiap *constraint* yang menggunakannya.

Penelusuran dalam metode *constraint propagation* dilakukan secara merambat. Suatu *propagator* dapat memicu *propagator* lain untuk melakukan inferensi terhadap nilai suatu variabel apabila melibatkan suatu variabel yang sama pada beberapa *constraint* yang berbeda.

Jenis *propagator* yang akan digunakan untuk melakukan proses *constraint propagation* dan menyelesaikan permainan *Minesweeper* ini adalah *Set Propagator* dan *Zero Propagator*. *Set Propagator* berfungsi untuk melakukan inferensi dalam menentukan nilai sebuah variabel dalam suatu *constraint*. Sedangkan *Zero Propagator* berfungsi untuk memberikan nilai 0 kepada semua variabel pada suatu *constraint* apabila nilai $K = 0$.

4.2.2.1 Set Propagator

Pada kumpulan *constraint* yang telah dibentuk dari kondisi papan permainan yang ditunjukkan pada gambar 4.2, *propagator* untuk (3,2) akan segera menginferensikan $x_{42} = 1$, yang berarti telah ditemukan satu lokasi pasti sebuah ranjau. Informasi ini akan digunakan oleh *propagator* (3,1) dan (3,2) untuk menginferensikan nilai variabel yang sama.

(3,3) $x_{42} = 1$

(3,1) $x_{41} + x_{42} = 1 \Leftrightarrow x_{41} + 1 = 1$

$\Leftrightarrow x_{41} = 0$

(3,2) $x_{41} + x_{42} = 1 \Leftrightarrow x_{41} = 0 \ ; \ x_{42} = 1$

sementara *propagator* (4,2) menginferensikan :

(4,2) $x_{42} + x_{52} = 1 \Leftrightarrow 1 + x_{52} = 1$

$\Leftrightarrow x_{52} = 0$

Melanjutkan proses inferensi awal yang telah dilakukan, *constraint* yang masih tersisa adalah :

(1,5) $x_{16} + x_{26} = 1$

(2,5) $x_{16} + x_{26} + x_{36} = 2$

(3,5) $x_{26} + x_{36} + x_{46} = 2$

(4,5) $x_{36} + x_{46} + x_{56} = 1$

(5,3) $x_{42} + x_{52} + x_{62} + x_{63} + x_{64} = 3$

(5,4) $x_{63} + x_{64} + x_{65} = 2$

(5,5) $x_{46} + x_{56} + x_{64} + x_{65} + x_{66} = 2$

Kelebihan dari *set propagator* adalah kemampuan untuk berbagi informasi berdasarkan suatu *set* variabel seperti $x_{16} + x_{26}$. Untuk contoh *constraint* (2,5)

dengan menggunakan *propagator* jenis ini, akan dibuat *propagator* sebanyak jumlah bagian dari kumpulan penunjuk / koordinat variabel{16, 26, 36}.

Untuk setiap *subset I* dari koordinat, "*set*" variabel x_I dapat didefinisikan dengan

$$x_I = \sum_{i \in I} x_i$$

Selanjutnya *constraint* (2,5) dapat diekspresikan sebagai berikut. Untuk setiap bagian $P = \{I_1, \ldots, I_k\}$ dari penunjuk, dapat dibuat satu *propagator* untuk *constraint*

$$x_{I_1} + \ldots + x_{I_n} = 2$$

yang ekuivalen dengan (2,5). Selanjutnya didapatkan *propagator* untuk

(2,5.*a*) $x_{\{16\}} + x_{\{26\}} + x_{\{36\}} = 2$

(2,5.*b*) $x_{\{16,26,36\}} = 2$

(2,5.*c*) $x_{\{16\}} + x_{\{26,36\}} = 2$

(2,5.*d*) $x_{\{26\}} + x_{\{16,36\}} = 2$

(2,5.*e*) $x_{\{36\}} + x_{\{16,26\}} = 2$

Hasil yang didapatkan setelah melakukan penelusuran dengan menggunakan *set propagator* adalah :

(1,5) $x_{16} + x_{26} = 1$

(2,5.e) $x_{\{36\}} + x_{\{16,26\}} = 2 \quad \Leftrightarrow \quad x_{\{36\}} + 1 = 2$

$$\Leftrightarrow \quad x_{\{36\}} = 1$$

sehingga *propagator* (4,5) akan menginferensikan :

(4,5) $x_{36} + x_{46} + x_{56} = 1 \quad \Leftrightarrow \quad 1 + x_{46} + x_{56} = 1$

$$\Leftrightarrow \quad x_{46} = x_{56} = 0$$

Penelusuran dari (3,5) dan (1,5) kemudian akan memberikan nilai x_{26} dan x_{16}.

(3,5) $x_{26} + x_{36} + x_{46} = 2 \Leftrightarrow x_{26} + 1 + 0 = 2$

$\Leftrightarrow x_{26} = 1$

(1,5) $x_{16} + x_{26} = 1 \Leftrightarrow x_{16} + 1 = 1$

$\Leftrightarrow x_{16} = 0$

Proses – proses diatas akan terus dilakukan setiap didapatkan informasi baru mengenai keadaan papan permainan. Sampai pada akhirnya semua kotak aman pada papan permainan telah terbuka dan semua lokasi ranjau telah ditemukan.

4.2.2.2 Zero Propagator

Zero Propagator bekerja dengan prinsip yang sederhana. Jika suatu kotak dibuka dan ternyata menunjukkan bahwa tidak ada ranjau di sekitarnya (angka petunjuk = 0) maka semua kotak yang berada di sekitarnya akan diberi status aman (bukan lokasi ranjau).

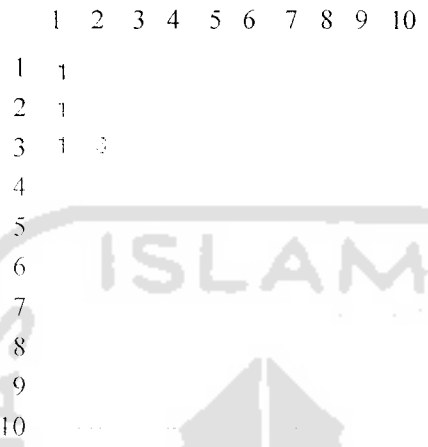
Contoh *Zero Propagator* pada kondisi permainan yang ditunjukkan pada gambar 4.2 adalah *propagator* (1,1) :

$$x_{12} + x_{21} + x_{22} = 0 \Leftrightarrow x_{12} = x_{21} = x_{22} = 0$$

4.2.3 Solver Engine

Pada suatu kondisi tertentu, metode *constraint propagation* tidak dapat menemukan solusi yang tepat untuk memecahkan masalah. Untuk itu, *constraint propagation engine* dilengkapi dengan *solver engine*. Fungsi dari *solver engine*

adalah mencoba memecahkan masalah yang dihadapi dengan cara menghitung nilai peluang suatu variabel.



Gambar 4.3 Contoh Kondisi Pemicu *Solver Engine*

Gambar 4.3 menunjukkan contoh dari kondisi papan permainan yang akan memicu *solver engine* untuk melakukan perhitungan peluang setiap variabel yang posisinya berhimpitan dengan angka – angka petunjuk.

Untuk menghitung peluang tiap variabel untuk bernilai 1 (ranjau) digunakan persamaan :

$$P(A \cap B) = P(A) \cdot P(B)$$

Sehingga pemecahan dari kondisi pada gambar 4.3 adalah :

Kotak (1, 2) = $(1/2) \times (1/2) = 0.25$

Kotak (2, 2) = $(1/2) \times (1/2) \times (3/6) \times (1/3) = 0.04125$

Kotak (2, 3) = $3/6 = 0.5$

Kotak (3, 3) = $3/6 = 0.5$

Kotak (4, 3) = $3/6 = 0.5$

Kotak (4, 1) = $(1/3) \times (3/6) = 0.165$

Kotak (4, 2) = $(1/3)\times(3/6) = 0.165$

Dari hasil perhitungan di atas, dapat disimpulkan bahwa kotak yang berpeluang sebagai lokasi ranjau adalah kotak (1, 2), (2, 3), (3, 3), (4, 3), (4, 1), dan (4, 2). Variabel – variabel yang mempunyai nilai peluang terkecil akan diberi tanda sebagai kotak yang aman.

4.2.4 Aplikasi Agent Pada Permainan

Fitur tambahan yang merupakan aplikasi dari *Agent* dan akan berinteraksi langsung dengan pemain pada permainan *Minesweeper* ini adalah fitur *Digital Assistant* dan *Autoplay*.

4.2.4.1 Digital Assistant

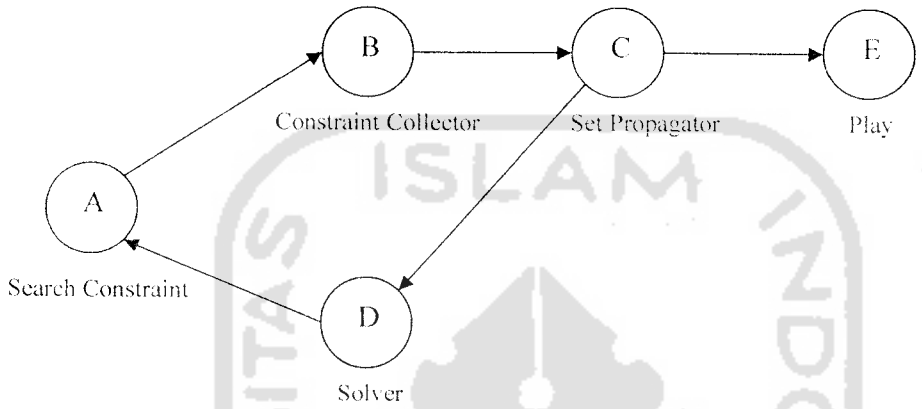
Digital Assistant akan menginformasikan hasil dari proses yang telah dilakukan oleh *constraint propagation engine* dengan memberi tanda / simbol pada kotak yang telah diketahui statusnya (aman atau lokasi ranjau), sehingga pemain dapat mengetahui kotak yang harus dimainkan / dibuka dan kotak yang merupakan lokasi ranjau.

4.2.4.2 Autoplay

Autoplay akan membuka semua kotak yang telah diberikan status aman oleh *constraint propagation engine*. *Autoplay* akan berhenti apabila semua kotak aman telah terbuka, atau menemui kondisi yang tidak dapat dipecahkan.

4.2.5 Aliran Proses Pada Agen

Proses – proses yang dijalankan oleh agen dapat digambarkan sebagai sebuah aliran proses.



Gambar 4.4 Aliran Proses Agen

4.2.5.1 Proses Utama

Berikut adalah *pseudo code* proses utama pada agen

```
ARanjauKetemu←HitungRanjauKetemu
AprevRanjauKetemu←ARanjauKetemu

if (AStatusAgen=saAssistant) then
begin
  ClickList←MoveGameClickList
  SearchConstraint←ClickList

  ARanjauKetemu←HitungRanjauKetemu

  if (ARanjauKetemu=APrevRanjauKetemu) then
    Solver
end else
if (AStatusAgen=saAutoplay) then
begin
  APrevCondition←FMinesweeper.ClickList.Count

  ClickList←MoveGameClickList
  SearchConstraint←ClickList

  if (FMinesweeper.ClickList.Count=APrevCondition) then
    Solver

end
```

4.2.5.2 Proses Pencarian Constraint (*Search Constraint)*

Proses ini akan mencari kotak yang telah terbuka / dimainkan pada papan permainan. Jika kotak tersebut mengandung angka petunjuk 0, maka akan memicu proses *ZeroPropagator*. Sedangkan jika angka petunjuk > 0, maka akan mengaktifkan proses *SetPropagator*. Berikut adalah *pseudo code* proses pencarian *constraint*

```
idx←0

while idx < AClickList.Count do
begin
  X1←ClickList[idx]^.X
  Y1←ClickList[idx]^.Y

  if (FMinesweeper.Game[X1,Y1].NearbyMines=0) then
    ZeroPropagator(X1,Y1)
  else
  if (FMinesweeper.Game[X1,Y1].NearbyMines>0) then
  begin
    AlistVar←ConstrCollector(X1,Y1)

    SetPropagator(X1,Y1)

    AListVar.Clear
  end

  AClickList.Delete(idx)
  inc(idx)
end
```

4.2.5.3 Proses Pengumpulan Constraint (*Constraint Collector*)

Proses ini akan mengumpulkan koordinat kotak yang belum terbuka di sekitar angka petunjuk dan memasukkannya ke dalam daftar variabel constraint. Berikut adalah *pseudo code* proses pengumpulan *constraint*

```
X←Kolom
Y←Baris

for I←X-1 to X+1 do
 for J←Y-1 to Y+1 do
  if (FMinesweeper.Game[I,J].Status=csUnClicked) then
  begin
```

```
      New(Variabel)

      Variabel.X←I;
      Variabel.Y←J;

      AListVar.Add(Variabel);
    end
  end for
end for
```

4.2.5.4 Proses Set Propagator

Proses ini akan membentuk *set* propagator dari variabel – variabel yang telah dikumpulkan yang kemudian dapat diproses untuk menentukan nilai suatu variabel. Berikut adalah *pseudo code* proses *set propagator*

```
if (AListVar.Count=FMinesweeper.Game[X, Y].NearbyMines) then
  begin
    idx←0

    while idx < AListVar.Count do
    begin
      X1←AListVar[idx]^.X
      Y1←AListVar[idx]^.Y

      ANilaiVar[X1,Y1].Nilai←1
      ANilaiVar[X1,Y1].Status←svRanjau

      InsertClickList(X1,Y1)

      Play

      inc(idx)
    end
  end else
  begin
    for JmlPartisi←AListVar.Count downto 1 do
    begin
      if (AListVar.Count div JmlPartisi)=AListVar.Count then
      begin
        WriteLog(X,Y,JmlPartisi)

        idx←0

        While idx < AListVar.Count do
        begin
          X1←AListVar[idx]^.X
          Y1←AListVar[idx]^.Y

          New(Variabel)
          Variabel.X←X1
```

```
      Variabel.Y←Y1

      APartisi[1].Variabels←Variabel

      inc(idx)
    end

    CekPartisi(X,Y,JmlPartisi)
    CrossCheck(X,Y,JmlPartisi)

    APartisi[1].Variabels.Clear
  end else
  if (JmlPartisi=AListVar.Count) then
  begin
    idx←0
    NoPartisi←1

    while idx < AListVar.Count do
    begin
      X1←AListVar[idx]^.X
      Y1←AListVar[idx]^.Y

      New(Variabel)
      Variabel.X←X1
      Variabel.Y←Y1

      APartisi[NoPartisi].Variabels←Variabel

      inc(NoPartisi)

      inc(idx)
    end

    CekPartisi(X,Y,JmlPartisi)

    WriteLog(X,Y,JmlPartisi)

    for NoPartisi←1 to JmlPartisi do
    begin
      APartisi[NoPartisi].Variabels.Clear
      APartisi[NoPartisi].Nilai←0
      APartisi[NoPartisi].NoneCount←0
    end for
  end else
  begin
    NoPartisi←1
    idx←0

    while idx < AListVar.Count do
    begin
      X1←AListVar[idx]^.X
      Y1←AListVar[idx]^.Y

      New(Variabel)
      Variabel.X←X1
```

```
        Variabel.Y←Y1

        if (NoPartisi <= JmlPartisi) then
          APartisi[NoPartisi].Variabels←Variabel
        else
        begin
          NoPartisi←1

          APartisi[NoPartisi].Variabels←Variabel
        end

        inc(NoPartisi)

        inc(idx)
      end

      if (JmlPartisi = AListVar.Count - 1) then
       RollVarPosition(X,Y,JmlPartisi)

       for NoPart←JmlPartisi downto 2 do
        if (APartisi[NoPart].Variabels.Count > 1) then
        begin
          idx←0

          while idx < APartisi[NoPart].Variabels.Count do
          begin
            if (idx > 0) then
            begin
              X1←APartisi[NoPart].Variabels[idx]^.X
              Y1←APartisi[NoPart].Variabels[idx]^.Y

              New(Variabel)
              Variabel.X←X1
              Variabel.Y←Y1

              APartisi[1].Variabels←Variabel

              APartisi[NoPart].Variabels.Delete idx)
            end

            RollVarPosition(X,Y,JmlPartisi)

            inc(idx)
          end
        end

      for NoPartisi←1 to JmlPartisi do
      begin
       APartisi[NoPartisi].Variabels.Clear

       APartisi[NoPartisi].Nilai←1
       APartisi[NoPartisi].NoneCount←0
      end for
    end
  end for
```

```
end
```

4.2.5.5 Proses Solver

Proses ini akan melakukan penghitungan nilai peluang jika *set propagator* menghadapi situasi yang tidak dapat dipecahkan. Berikut adalah *pseudo code* dari proses *solver*

```
Berhenti←false

for X←0 to FMinesweeper.Lebar-1 do
begin
 for←0 to FMinesweeper.Tinggi-1 do
  if (FMinesweeper.Game[X,Y].Status=csClicked) and
 (FMinesweeper.Game[X,Y].NearbyMines>0) then
  begin
    ConstrCollector(X,Y)

    idx←0
    JmlRanjau←0
    while idx < AListVar.Count do
    begin
      X1←AListVar[idx]^.X;
      Y1←AListVar[idx]^.Y;

      if (ANilaiVar[X1,Y1].Nilai=1) and (ANilaiVar[X1,
      Y1].Status=svRanjau) then
      begin
        AListVar.Delete(idx)

        inc(Jmlranjau)
      end else
      if (ANilaiVar[X1,Y1].Nilai=0) and (ANilaiVar[X1,
      Y1].Status=svAman) then
      begin
        AListVar.Delete(idx)
      end

      inc(idx)
    end

    if (AListVar.Count>0) and (AListVar.Count>
    formUtama.FMinesweeper.Game[X, Y].NearbyMines -
    JmlRanjau) then
    begin
      idx←0

      while idx < AListVar.Count do
      begin
        X1←AlistVar[idx]^.X
        Y1←AListVar[idx]^.Y
```

```
 APeluang[X1, Y1]←(AlistVar.Count -
(FMinesweeper.Game[X,Y].NearbyMines -
JmlRanjau))/AListVar.Count

for I←X1-1 to X1+1 do
 for J←Y1-1 to Y1+1 do
  if (FMinesweeper.Game[I, J].Status = csClicked) and
   (FMinesweeper.Game[I,J].NearbyMines > 0) then
   begin
     NbrConstrCollector(I,J)

     idx2←0
     JmlRanjauNeighbour←0

      while idx2<ANeighbourVarList.Count do
      begin
        X2←ANeighbourVarList[idx2]^.X
        Y2←ANeighbourVarList[idx2]^.Y

        if (ANilaiVar[X2, Y2].Nilai  = 1) and
       (ANilaiVar[X2,Y2].Status=svRanjau) then
       begin
         ANeighbourVarList.Delete(idx2)

         inc(JmlRanjauNeighbour)
       end else
       if (ANilaiVar[X2, Y2].Nilai = 0) and
       (ANilaiVar[X2,Y2].Status=svAman) then
       begin
         ANeighbourVarList.Delete(idx2)
       end

       inc(idx2)
      end

     if (ANeighbourVarList.Count > 0) then
     begin
       APeluang[X1, Y1]←APeluang[X1, Y1] +
       ((ANeighbourVarList.Count -
       (FMinesweeper.Game[I, J].NearbyMines -
        JmlRanjauNeighbour))/ANeighbourVarList.Count)
     end

     ANeighbourVarList.Clear
   end
  end

  inc(idx)
end

idx←0
NilaiMinimum←0
JmlPilihanMin←0

while idx < AlistVar.Count do
```

```
begin
  X1←AListVar[idx]^.X
  Y1←AlistVar[idx]^.Y

  if (idx=0) then
  begin
    NilaiMinimum←APeluang[X1, Y1]
    JmlPilihanMin←1
  end else
  if (idx>0) then
  begin
    if (APeluang[X1, Y1] < NilaiMinimum) then
    begin
      NilaiMinimum←APeluang[X1, Y1]
      JmlPilihanMin←1
    end else
    if (APeluang[X1, Y1]=NilaiMinimum) then
      inc(JmlPilihanMin)
  end

  inc(idx)
end;

idx←0
NilaiMaksimum←0
JmlPilihanMax←0

while idx<AlistVar.Count do
begin
  X1←AListVar[idx]^.X
  Y1←AlistVar[idx]^.Y

  if (idx = 0) then
  begin
    NilaiMaksimum←APeluang[X1, Y1]
    JmlPilihanMax←1
  end else
  if (idx>0) then
  begin
    if (APeluang[X1,Y1]>NilaiMaksimum) then
    begin
      NilaiMaksimum←APeluang[X1, Y1]
      JmlPilihanMax←1
    end else
    if (APeluang[X1,Y1]=NilaiMaksimum) then
      inc(JmlPilihanMax
  end

  inc(idx)
end

if (FMinesweeper.Game[X, Y].NearbyMines = 1) and
(NilaiMinimum <> NilaiMaksimum) then
begin
  if (JmlPilihanMin>1) then
```

```
begin
  idx2←0

  while idx2 < AlistVar.Count do
  begin
    X2←AListVar[idx2]^.X
    Y2←AListVar[idx2]^.Y

    if (APeluang[X2,Y2]=NilaiMaksimum) then
    begin
      ANilaiVar[X2,Y2].Nilai←0
      ANilaiVar[X2,Y2].Status←svAman

      InsertCLickList(X2,Y2)

      Play
    end

    inc(idx2)
  end

  for←0 to FMinesweeper.Lebar - 1 do
   for←0 to FMinesweeper.Tinggi - 1 do
    APeluang[I, J] ←0

  AListVar.Clear
  Berhenti←true
end else
if (JmlPilihanMin=1) then
begin
  idx2←0

  while idx2 < AlistVar.Count do
  begin
    X2←AListVar[idx2]^.X
    Y2←AListVar[idx2]^.Y

    if (APeluang[X2,Y2]=NilaiMinimum) then
    begin
      ANilaiVar[X2,Y2].Nilai←1
      ANilaiVar[X2,Y2].Status←svRanjau

      InsertCLickList(X2,Y2)

      Play
    end else
    if (APeluang[X2, Y2]>NilaiMinimum) then
    begin
      ANilaiVar[X2, Y2].Nilai←0
      ANilaiVar[X2, Y2].Statuss←vAman

      InsertCLickList(X2,Y2

      Play
    end
```

```
            inc(idx2)
          end

          for←0 to FMinesweeper.Lebar - 1 do
           for J←0 to FMinesweeper.Tinggi - 1 do
            APeluang[I,J] ←0

          AListVar.Clear
          Berhenti←true
        end
      end else
      if (FMinesweeper.Game[X, Y].NearbyMines > 1) then
      begin
        if (NilaiMinimum <> NilaiMaksimum) then
        begin
          idx2←0

          while idx2 < AListVar.Count do
          begin
            X2←AlistVar[idx2]^.X
            Y2←AListVar[idx2]^.Y

            if (APeluang[X2,Y2]=NilaiMaksimum) then
            begin
              ANilaiVar[X2, Y2].Status←svAman

              InsertClickList(X2,Y2)

              Play
            end

            inc(idx2)
          end

          for I←0 to FMinesweeper.Lebar - 1 do
           for J←0 to FMinesweeper.Tinggi - 1 do
            APeluang[I, J]←0

          AListVar.Clear
          Berhenti←true
        end
      end
    end

    AListVar.Clear
  end

  for I←0 to FMinesweeper.Lebar - 1 do
   for J←0 to FMinesweeper.Tinggi - 1 do
    APeluang[I, J]←0

End
```

4.2.5.6 Proses Memainkan Suatu Kotak (*Play*)

Proses ini akan memainkan kotak yang telah diketahui nilainya. Jika status aktif agen sebagai *autoplayer* dan kotak tersebut bernilai 0 (aman) maka agen akan membuka kotak tersebut. Jika status aktif agen adalah sebagai *assistant*, maka agen akan memberikan tanda pada kotak tersebut. Berikut *pseudo code* dari proses *Play*

```
for I←0 to FMinesweeper.Lebar - 1 do
  for J←0 to FMinesweeper.Tinggi - 1 do
  begin
    if (AStatusAgen=saAssistant) then
    begin
        if (FMinesweeper.Game[I, J].Status = csUnClicked) and
        (ANilaiVar[I, J].Status = svAman) then
        begin
          Draw(I * LEBAR_KOTAK, J * TINGGI_KOTAK,
          GbrKotak.Bitmap[SAVE_BMP])
        end else
        if (FMinesweeper.Game[I,J].Status=csUnClicked) and
        (ANilaiVar[I, J].Nilai = 1) then
        begin
          Draw(I * LEBAR_KOTAK, J * TINGGI_KOTAK,
          GbrKotak.Bitmap[FLAG_BMP])
        end
    end else
    if (AStatusAgen=saAutoPlay) then
    begin
      if (FMinesweeper.Game[I, J].Status = csUnClicked) and
      (ANilaiVar[I, J].Status = svAman) then
        FMinesweeper.SetStatus(I,J,csClicking)
    end
  end for
end for
```

4.2.6 Strukutur Data

4.2.6.1 Data Ranjau

Koordinat dari kotak yang menyimpan ranjau di dalamnya, disimpan dalam sebuah *array* dua dimensi yang menunjukkan koordinat dari masing – masing kotak pada papan permainan.

```
TKotak = record
    Mined : Boolean;
    NearbyMines : Integer;
    ClickingStep : Integer;
    Status : TStatusKotak;
  end;


  TGame = Array [0..GAME_LEBAR_MAX-1, 0..GAME_TINGGI_MAX-1]
  of TKotak;
```

4.2.6.2 Data Constraint

Koordinat variabel – variabel *constraint* yang didapatkan oleh *constraint collector* akan disimpan dalam suatu *array* dua dimensi sesuai dengan koordinatnya masing – masing.

```
TVarStatus = record
   Nilai : Integer;
   Status : TStatusVar;

TNilaiVar=array [0..GAME_LEBAR_MAX - 1, 0..GAME_TINGGI_MAX -
1] of TVarStatus;
```

4.2.7 Perancangan Antarmuka

4.2.7.1 Form Utama

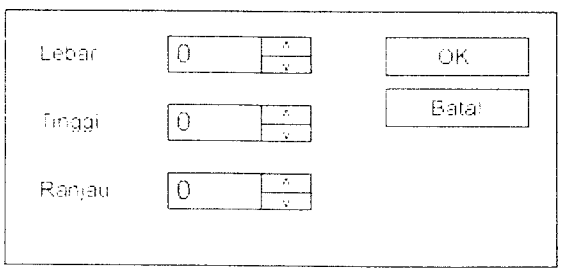
Memberikan gambaran tampilan utama yang akan diberikan pada pemain saat akan memulai permainan. *Form* ini dibagi menjadi tiga bagian utama, yaitu *Main Menu*, *Title Panel*, dan *Papan ranjau*.

Pada bagian *Main Menu* ditampilkan tiga menu utama, yaitu *Game*, *Assistant*, dan *Help* yang masing – masing mempunyai *sub menu* serta dapat dieksekusi melalui tombol *shortcut* menggunakan *keyboard*. Sedangkan pada bagian *Title Panel* terdapat informasi jumlah ranjau yang tersisa pada papan permainan, status permainan, dan penunjuk waktu. Pemain juga dapat memulai permainan baru dengan meng-*klik Title Panel*.

Gambar 4.5 Rancangan form utama

4.2.7.2 Form Custom Game

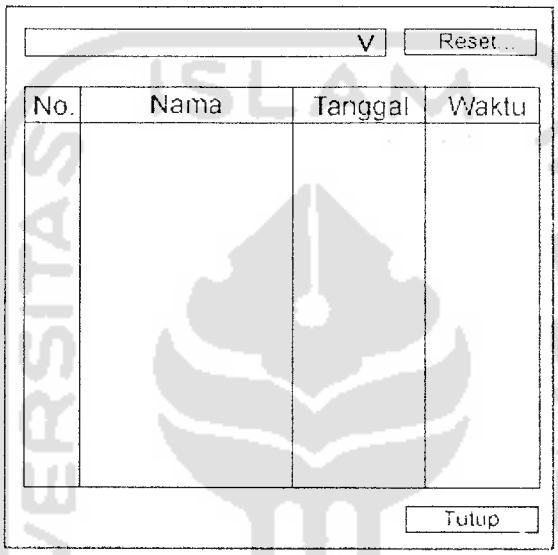
Digunakan untuk melakukan setting permainan sesuai dengan keinginan pemain. Nilai lebar dan tinggi minimal papan permainan yang dapat dipilih pemain adalah 5, dan nilai maksimalnya adalah 50. Sedangkan jumlah minimal ranjau adalah 1 dan jumlah maksimalnya adalah (Lebar * Tinggi)/2. Pembatasan nilai lebar, tinggi dilakukan karena sistem yang akan dibangun tidak dilengkapi dengan fungsi untuk menampilkan *scrollbar* secara otomatis bila ukuran papan permainan melebihi ukuran layar pengguna.



Gambar 4.6 Rancangan form custom game

4.2.7.3 Form Best Times

Pada form ini akan ditampilkan catatan waktu terbaik yang diperoleh pemain pada masing – masing ukuran papan permainan yang sudah pernah dimainkan. Satuan waktu yang digunakan adalah "detik".



Gambar 4.7 Rancangan form best times

4.2.7.4 Form About

Form ini akan menampilkan informasi mengenai pembuat software.

Gambar 4.8 Rancangan form about

BAB V

IMPLEMENTASI PERANGKAT LUNAK

5.1 Batasan Implementasi

Batasan – batasan yang digunakan dalam implementasi sistem ini adalah sebagai berikut :

Pemain hanya dapat bermain dengan menggunakan bantuan *mouse*. Sedangkan *keyboard* dapat digunakan untuk memilih menu yang disediakan melalui kode *shortcut*.

File utama dari aplikasi permainan ini adalah *Minesweeper.exe*, sedangkan file – file pendukungnya adalah *BestTimes.dat* untuk menyimpan catatan waktu terbaik, dan *Minesweeper.ini* untuk menyimpan konfigurasi papan permainan serta *Logs.txt* dan *Solver.txt* yang akan menyimpan rekaman inferensi yang dilakukan oleh *Constraint Propagation Engine*.

5.1.1 Lingkungan pengembangan

Lingkungan pengembangan aplikasi *Logical Agent* dengan studi kasus permainan *Minesweeper* ini adalah lingkungan Windows.

5.1.2 Bahasa dan Compiler yang dipakai

Bahasa pemrograman yang digunakan dalam implementasi aplikasi *Logical Agent* dengan studi kasus permainan *Minesweeper* ini adalah bahasa pemrograman *Pascal* dengan *Compiler* menggunakan Borland Delphi 7.

5.1.3 Perangkat Lunak yang digunakan

Untuk mewujudkan sistem yang akan dibangun, dibutuhkan beberapa perangkat lunak pendukung, diantaranya :

1. Sistem Operasi

   Sistem operasi yang digunakan dalam aplikasi *Logical Agent* dengan studi kasus permainan *Minesweeper* ini adalah Windows XP.

2. Komponen Tambahan

   Komponen yang digunakan untuk membantu membuat tampilan papan ranjau dan panel utama adalah *Graphics32*. Komponen ini dibuat oleh Alex A. Denisov dan diambil dari http://www.g32.org. Komponen *Graphics32* ini bersifat *freeware*. dan bebas digunakan untuk tujuan pendidikan maupun komersil.

3. Aplikasi untuk pengembangan perangkat lunak

   Aplikasi yang digunakan untuk pengembangan sistem adalah Adobe Photoshop CS dan Corel Draw 12. Aplikasi tersebut digunakan untuk membuat rancangan antarmuka dan grafis untuk tampilan pada papan permainan.

5.1.4 Perangkat Keras yang digunakan

Perangkat keras yang digunakan dalam pengembangan aplikasi ini adalah :

1. Notebook COMPAQ Centrino Duo 1.60 GHz.
2. Memory 512 DDR2
3. HDD minimal 1 Gbyte untuk menampung file instalasi Borland Delphi 7 dan aplikasi pendukung lainnya.

5.2 Implementasi

5.2.1 Pembangunan Aplikasi Permainan

5.2.1.1 Inisialisasi Permainan

Pada saat proses inisialisasi permainan, aplikasi akan membaca konfigurasi permainan yang terakhir kali dimainkan pada file *Minesweeper.ini*. File tersebut terdapat pada *folder* Data di direktori program utama.

```
TIniFile.Create(ExtractFilePath(ParamStr(0))+
'/Data/Minesweeper.ini')
```

Apabila aplikasi berhasil membaca konfigurasi pada file tersebut, maka papan permainan akan diset sesuai dengan konfigurasi pada file *Minesweeper.ini*. Jika file tersebut belum ada, maka aplikasi akan membuat dan mengisinya dengan konfigurasi level *Beginner* (9 X 9, dengan 10 ranjau).

```
SetUkuran(ReadInteger('Main', 'Lebar', 9), ReadInteger('Main',
'Tinggi', 9), ReadInteger('Main', 'Ranjau', 10));

GameMarks.Checked := ReadBool('Main', 'Marks', False);
GameSound.Checked := ReadBool('Main', 'Sound', False);
```

Selanjutnya dilakukan pengacakan posisi ranjau dan menentukan angka petunjuk pada kotak yang di sekitarnya terdapat ranjau. Koordinat – koordinat dan nilai angka petunjuk tersebut disimpan dalam sebuah *array*.

```
//Acak posisi ranjau

for I := 0 to FCountRanjau - 1 do
begin
  repeat
  X := Random(Lebar);
  Y := Random(Tinggi);
  until not FGame[X, Y].Mined;

  FGame[X, Y].Mined := true;
end;

//Set angka petunjuk
```

```
for I := 0 to FLebar - 1 do
 for J := 0 to FTinggi - 1 do
  for K := I - 1 to I + 1 do
   for L := J - 1 to J + 1 do
    if (K >= 0) and (K < FLebar) and (L >= 0) and (L < FTinggi)
      and FGame[K, L].Mined then
        inc(FGame[I, J].NearbyMines);
```

5.2.1.2 Penggambaran Papan Ranjau

Setelah proses inisialisasi dilakukan, selanjutnya adalah menggambar papan ranjau sesuai dengan kondisi yang ada pada file konfigurasi.

```
with AKotak do
  case Status of

csUnclicked :

if (FMinesweeper.StatusGame in [gsMenang, gsKalah]) and (Mined)
then
begin
  GamePb.Buffer.Draw(X*LEBAR_KOTAK,Y*TINGGI_KOTAK,GbrKotak.
  Bitmap[CLICKED_BMP]);

  GamePb.Buffer.Draw(X*LEBAR_KOTAK,Y*TINGGI_KOTAK,Gbrkotak.
  Bitmap[MINED_BMP]);

end else

  GamePb.Buffer.Draw(X*LEBAR_KOTAK,Y*TINGGI_KOTAK,GbrKotak.
  Bitmap[UNCLICKED_BMP]);

csClicked, csClicking :

if (Status = csClicking) and (ClickingStep >= NB_STEP_MAX) then
begin
  GamePb.Buffer.Draw(X*LEBAR_KOTAK,Y*TINGGI_KOTAK,GbrKotak.
  Bitmap[UNCLICKED_BMP]);

end else

begin
if Mined then
begin
  GamePb.Buffer.Draw(X*LEBAR_KOTAK,Y*TINGGI_KOTAK,GbrKotak.
  Bitmap[RED_BMP]);

  GamePb.Buffer.Draw(X*LEBAR_KOTAK,Y*TINGGI_KOTAK,GbrKotak.
  Bitmap[MINED_BMP]);

end else

if NearbyMines >= 0 then
```

```
begin
  GamePb.Buffer.Draw(X*LEBAR_KOTAK,Y*TINGGI_KOTAK,GbrKotak.
  Bitmap[CLICKED_BMP]);

if NearbyMines > 0 then
  GamePb.Buffer.Draw(X*LEBAR_KOTAK,Y*TINGGI_KOTAK,Gbrkotak.
  Bitmap[MINE1_BMP + NearbyMines - 1]);
end;
end;

csMined :
if (FMinesweeper.StatusGame in [gsMenang, gsKalah]) and not
Mined then
begin
  GamePb.Buffer.Draw(X*LEBAR_KOTAK,Y*TINGGI_KOTAK,GbrKotak.
  Bitmap[CLICKED_BMP]);

  GamePb.Buffer.Draw(X*LEBAR_KOTAK,Y*TINGGI_KOTAK,GbrKotak.
  Bitmap[MINED_BMP]);

  GamePb.Buffer.Draw(X*LEBAR_KOTAK,Y*TINGGI_KOTAK,GbrKotak.
  Bitmap[CROSS_BMP]);

end else

begin
  GamePb.Buffer.Draw(X*LEBAR_KOTAK,Y*TINGGI_KOTAK,GbrKotak.
  Bitmap[UNCLICKED_BMP]);

  GamePb.Buffer.Draw(X*LEBAR_KOTAK,Y*TINGGI_KOTAK,GbrKotak.
  Bitmap[FLAG_BMP]);
end;

csMarked :
begin
  GamePb.Buffer.Draw(X*LEBAR_KOTAK,Y*TINGGI_KOTAK,gbrKotak.
  Bitmap[UNCLICKED_BMP]);

  GamePb.Buffer.Draw(X*LEBAR_KOTAK,Y*TINGGI_KOTAK,GbrKotak.
  Bitmap[MARKED_BMP]);
end;
end;
```

Prosedur ini juga digunakan pada saat permainan telah berjalan untuk melakukan *update* terhadap papan permainan apabila pemain melakukan suatu aksi.

5.2.1.3 Set Status Permainan

Prosedur ini dijalankan saat pemain melakukan aksi klik pada papan ranjau. Jika kotak yang di-klik mengandung ranjau, maka status permainan akan diset kalah. Sebaliknya, apabila kotak yang di-klik tidak mengandung ranjau, maka kotak tersebut akan dibuka dan status permainan tetap berlanjut.

```
if (X >= 0) and (X < FLebar) and (Y >= 0) and (Y < FTinggi) then
begin
  if FStatusGame = gsMulai then
    FStatusGame := gsBermain;

  if Nilai = csMined then
  begin
    Dec(FSisaRanjau);
    if FSisaRanjau = 0 then
      CekGameSelesai;
  end else
  if ((Nilai = csUnClicked) and (FGame[X, Y].Status = csMined))
   or (Nilai = csMarked) then
    Inc(FSisaRanjau);

  if (Nilai = csClicking) and (FGame[X, Y].NearbyMines = 0) then
    ClearZone(X, Y, X, Y)
  else
  begin
    FGame[X, Y].Status := Nilai;
    if Nilai = csClicking then
    begin
      FGame[X, Y].ClickingStep := NB_STEP_MAX;
      AddClickKotak(X, Y);
      AddClickList(X, Y);
    end;

  if (Nilai in [csClicked, csClicking]) and FGame[X, Y].Mined
then
    FStatusGame := gsKalah;
  end;

  if (FClickList.Count) + SisaRanjau = (Lebar * Tinggi) then
   for I := 0 to Lebar - 1 do
    for J := 0 to Tinggi - 1 do
     if (Game[I,J].Status in [csMined, csUnClicked]) then
     begin
       SetStatus(I, J, csMined);

       formUtama.TimerKotak.Enabled := true;
     end;
```

Pada prosedur ini juga dilakukan pengecekan apakah kotak yang dibuka merupakan kotak aman yang terakhir. Jika kotak tersebut adalah kotak aman yang terakhir, maka status permainan akan diset menang.

5.2.1.4 Pencatatan Waktu Terbaik

Jika permainan dapat diselesaikan tanpa menggunakan bantuan dari agen dalam waktu , yang lebih cepat dari catatan waktu yang sudah tersimpan sebelumnya, maka pemain dapat mencatatkan waktunya ke dalam daftar waktu terbaik. Untuk setiap kategori / ukuran papan permainan, hanya dicatat 10 waktu terbaik.

```
if Kategori <> nil then
begin
  I := 1;

  while (Kategori.BestTimes[I].waktu > -1) and (Kategori.
   BestTimes[I].waktu < Waktu) and (I <= NB_BEST_TIMES) do
     Inc(I);

  if I <= NB_BEST_TIMES then
  begin
    for J := NB_BEST_TIMES downto I + 1 do
      Kategori.BestTimes[J] := Kategori.BestTimes[J - 1];

    Kategori.BestTimes[I].Nama := Nama;
    Kategori.BestTimes[I].Tanggal := Tanggal;
    Kategori.BestTimes[I].waktu := Waktu;
  end;
end;
```

Jika waktu untuk menyelesaikan permainan memenuhi syarat untuk dicatat, maka akan dilakukan penulisan ke file *Data.dat*.

```
AssignFile(F, ExtractFilePath(ParamStr(0)) +
'/Data/BestTimes.dat');
  Rewrite(F);

  while ListWaktu.Count > 0 do
  begin
    BlockWrite(F, PWaktu(ListWaktu[0])^, 1);
    Dispose(ListWaktu[0]);
    ListWaktu.Delete(0);
```

```
end;

CloseFile(F);
ListWaktu.Free;
ListWaktu := nil;
```

5.2.2 Pembangunan Agen

5.2.2.1 Constraint Collector

Pada saat agen diaktifkan oleh pemain, baik dengan status sebagai *Digital Assistant* maupun *Autoplay*, agen akan membaca setiap kotak yang telah terbuka pada papan ranjau. Jika kotak mengandung angka petunjuk, prosedur ini akan mencari kotak di sekitar yang belum terbuka dan memasukkan koordinatnya ke dalam daftar variabel.

```
for I := X - 1 to X + 1 do
 for J := Y - 1 to Y + 1 do
 if (I >= 0) and (I < formUtama.FMinesweeper.Lebar) and (J >= 0)
 and (J < formUtama.FMinesweeper.Tinggi) and
 (formUtama.FMinesweeper.Game[I, J].Status = csUnClicked) then
   begin
     New(Variabel);

     Variabel.X := I;
     Variabel.Y := J;

     AListVar.Add(Variabel);
   end;
```

Variabel – variabel yang telah dikumpulkan akan membentuk sebuah *constraint*, dimana hasil penjumlahan dari semua variabel tersebut harus sama jumlahnya dengan angka petunjuk yang dikelilingi oleh variabel – variabel itu.

5.2.2.2 Zero Propagator

Zero Propagator akan dieksekusi jika pada kotak yang dibuka tidak terdapat angka petunjuk. Hal ini menunjukkan bahwa di sekitar kotak tersebut tidak terdapat ranjau sehingga semua kotak di sekitarnya diberi status aman.

```
for I := X - 1 to X + 1 do
 for J := Y - 1 to Y + 1 do
  if (I >= 0) and (I < formUtama.FMinesweeper.Lebar) and (J >=
  0) and (J < formUtama.FMinesweeper.Tinggi) then
    ANilaiVar[I, J].Status := svAman;
```

5.2.2.3 Pencatatan Inferensi

Pada saat *propagator engine* melakukan inferensi, kegiatan tersebut akan dicatat dan ditulis ke dalam file *Logs.txt*. Pencatatan dilakukan setiap kali dibentuk *set propagator* untuk menentukan nilai suatu variabel pada *constraint* yang aktif. Hasil pencatatan pada file *Logs.txt* hanya memuat daftar *constraint* berdasarkan jumlah partisi yang dapat dibentuk terhadap *constraint* aktif, dan tidak terdapat nilai variabel yang telah diketahui.

```
idx := 0;

str := '(' + IntToStr(X) + ',' + IntToStr(Y) + ') ==> (';

while idx < AListVar.Count do
begin
  X1 := PVariabel(AListVar[idx])^.X;
  Y1 := PVariabel(AListVar[idx])^.Y;

  str := str + 'X[' + IntToStr(X1) + ',' + IntToStr(Y1) + ']';

  inc(idx);

  if (idx = AListVar.Count) then
     str := str + ') = ' + IntToStr(formUtama.FMinesweeper.
     Game[X, Y].NearbyMines)
  else
     str := str + '+';
  end;

  writeln(ALogFile, str);
  end else
  for NoPartisi := 1 to JmlPartisi do
  begin
    idx := 0;

    if (NoPartisi = 1) then
     str := '(' + IntToStr(X) + ',' + IntToStr(Y) + ') ==> ('
    else
    str := '(';

    while idx < APartisi[NoPartisi].Variabels.Count do
```

```
      begin
        X1 := PVariabel(APartisi[NoPartisi].Variabels[idx])^.X;
        Y1 := PVariabel(APartisi[NoPartisi].Variabels[idx])^.Y;

        str := str + 'X[' + IntToStr(X1) + ',' + IntToStr(Y1) +
        ']';

        inc(idx);

        if (idx = APartisi[NoPartisi].Variabels.Count) then
         str := str + ')'
        else
         str := str + '+';
      end;

      if (NoPartisi < JmlPartisi) then
       str := str + ' ; '
      else
       str := str + ' = ' + IntToStr(formUtama.FMinesweeper.
      Game[X, Y].NearbyMines);

     write(ALogFile, str);
    end;

   writeln(ALogFile, ' ');
```

5.2.2.4 Memainkan Kotak Tertentu

Apabila suatu kotak telah diketahui nilainya, agen akan segera memainkan kotak tersebut sesuai dengan status aktifnya. Apabila agen berjalan pada status *autoplay*, agen akan membuka kotak tersebut secara otomatis. Sedangkan apabila status agen adalah *digital assistant*, maka kotak aman akan diberi tanda lingkaran hijau, dan lokasi ranjau ditandai dengan bendera merah.

```
ClearMarks;

 for I := 0 to formUtama.FMinesweeper.Lebar - 1 do
  for J := 0 to formUtama.FMinesweeper.Tinggi - 1 do
  begin
    if (AStatusAgen = saAssistant) then
    begin
      if (formUtama.FMinesweeper.Game[I, J].Status =
      csUnClicked) and (ANilaiVar[I, J].Status = svAman) then
      begin
        formUtama.GamePb.Buffer.Draw(I * LEBAR_KOTAK, J *
        TINGGI_KOTAK, formUtama.GbrKotak.Bitmap[SAVE_BMP]);

        formUtama.TimerKotak.Enabled := true;
      end else
```

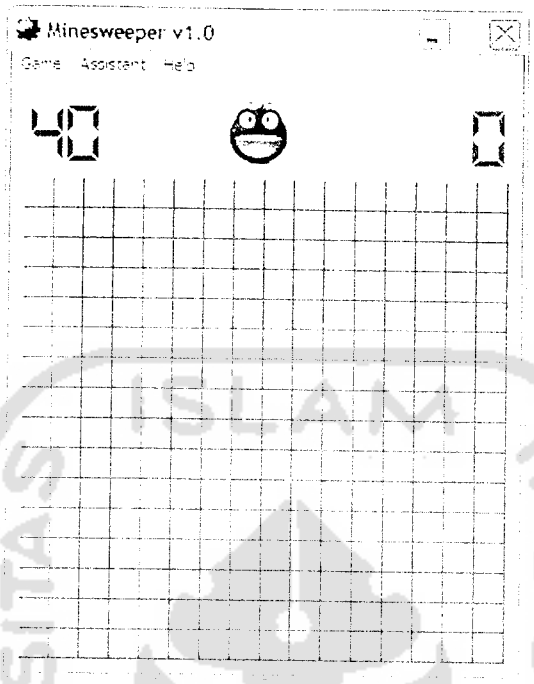
```
      if (formUtama.FMinesweeper.Game[I, J].Status =
      csUnClicked) and (ANilaiVar[I, J].Nilai = 1) then
      begin
        formUtama.GamePb.Buffer.Draw(I * LEBAR_KOTAK, J *
        TINGGI_KOTAK, formUtama.GbrKotak.Bitmap[FLAG_BMP]);

        formUtama.TimerKotak.Enabled := true;
      end;
    end else
    if (AStatusAgen = saAutoPlay) then
    begin
      if (formUtama.FMinesweeper.Game[I, J].Status =
      csUnClicked) and (ANilaiVar[I, J].Status = svAman) then
      begin
        formUtama.FMinesweeper.SetStatus(I, J, csClicking);

        formUtama.TimerKotak.Enabled := true;

        formUtama.ActionList1.Actions[13].Execute;
      end;
    end;
  end;
```

5.2.3 Implementasi Antarmuka

5.2.3.1 Form Utama

Memberikan gambaran tampilan utama yang akan diberikan pada pemain saat akan memulai permainan. *Form* ini dibagi menjadi tiga bagian utama, yaitu *Main Menu*, *Title Pane*, dan Papan ranjau (gambar 5.1).

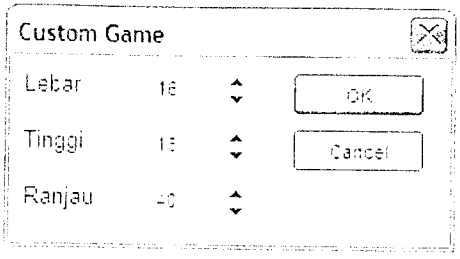
Pada bagian *Main Menu* ditampilkan tiga menu utama, yaitu *Game*, *Assistant*, dan *Help* yang masing – masing mempunyai *sub menu* serta dapat dieksekusi melalui tombol *shortcut* menggunakan *keyboard*. Sedangkan pada bagian *Title Panel* terdapat informasi jumlah ranjau yang tersisa pada papan permainan, status permainan, dan penunjuk waktu. Pemain juga dapat memulai permainan baru dengan meng-*klik Title Panel*.



Gambar 5.1 Form Utama

5.2.3.2 Form Custom Game

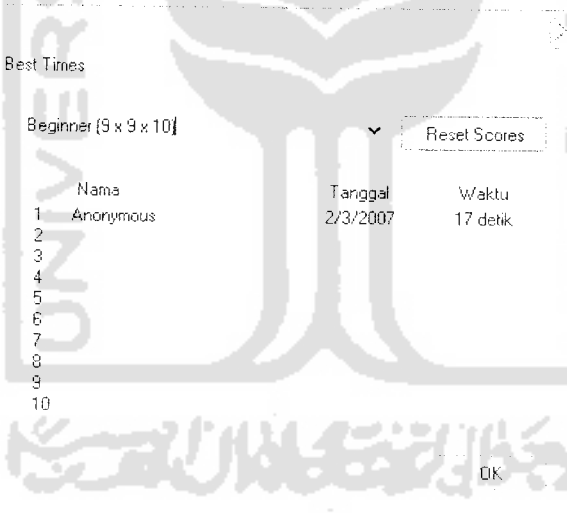
Digunakan untuk melakukan setting permainan sesuai dengan keinginan pemain (gambar 5.2). Nilai lebar dan tinggi minimal papan permainan yang dapat dipilih pemain adalah 5, dan nilai maksimalnya adalah 50. Sedangkan jumlah minimal ranjau adalah 1 dan jumlah maksimalnya adalah (Lebar * Tinggi)/2. Pembatasan nilai lebar, tinggi dilakukan karena sistem yang dibangun tidak dilengkapi dengan fungsi untuk menampilkan *scrollbar* secara otomatis bila ukuran papan permainan melebihi ukuran layar pengguna.



Gambar 5.2 Form Custom Game

5.2.3.3 Form Best Times

Form ini menampilkan catatan waktu terbaik yang diperoleh pemain pada masing – masing ukuran papan permainan yang sudah pernah dimainkan. Pemain dapat mencatatkan waktunya hanya jika dapat menyelesaikan permainan tanpa bantuan agen. Satuan waktu yang digunakan adalah "detik" (gambar 5.3).



Gambar 5.3 Form Best Times

5.2.3.4 Form About

Form ini menampilkan informasi mengenai pembuat software (gambar 5.4).

Gambar 5.4 Form About

BAB VI

ANALISIS KINERJA PERANGKAT LUNAK

6.1 Dokumentasi Penggunaan Sistem

Pengujian dilakukan dengan menggunakan data *Minesweeper* normal dan data *Minesweeper* tidak normal. Hal ini dilakukan untuk menguji agen apakah dapat menyelesaikan permainan atau berhenti karena tidak mampu menemukan solusi saat menghadapi suatu kondisi.

Pengujian yang dilakukan meliputi penerapan agen dalam memberikan solusi permainan *Minesweeper* dalam mode *Autoplay* menggunakan ukuran papan permainan pada level *beginner* dan *intermediate*. Pada masing – masing level akan dilakukan pengujian terhadap data normal dan data tidak normal. Data normal adalah kondisi papan permainan yang telah terbuka dan terdapat kotak yang sudah mempunyai nilai pasti (nilai 1 untuk ranjau atau 0 jika bukan ranjau). Data tidak normal adalah kondisi papan permainan yang telah terbuka namun belum ada kotak yang mempunyai nilai pasti.

6.2 Analisis Kinerja

Analisis kinerja dilakukan dengan berbagai pengujian sebagai berikut :

Pengujian yang dilakukan dengan menguji agen untuk menyelesaikan permainan *Minesweeper* dalam mode *Autoplayer*. Dalam pengujian ini, akan dapat diketahui kemampuan agen dalam memecahkan permasalahan permainan *Minesweeper*.

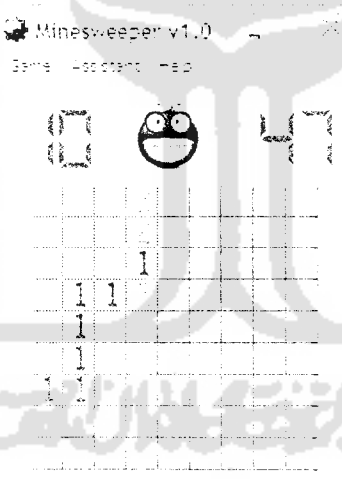
Dalam pengujian akan diamati kinerja dari *Constraint Propagation Engine* dalam memecahkan masalah yang dihadapi dalam permainan *Minesweeper*. Dalam pengujian juga akan diamati apakah *solver engine* diperlukan agen untuk membantu menyelesaikan permainan.

6.2.1 Pengujian Terhadap Aplikasi

6.2.1.1 Dengan Data Normal

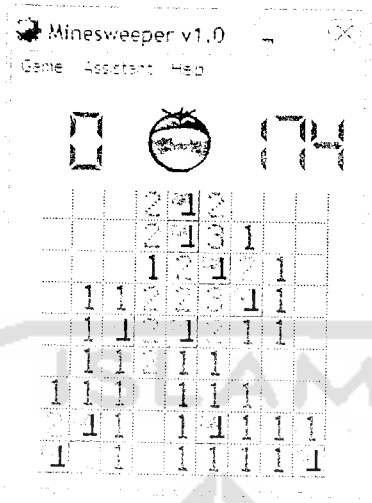
6.2.1.1.1 Pengujian Level Beginner

Pada pengujian terhadap level *beginner*, ukuran papan permainan yang dipakai adalah 9 X 9 dengan ranjau berjumlah 10 buah (gambar 6.1).



Gambar 6.1 Papan permainan ukuran 9 X 9 yang sudah terbuka

Pada Gambar 6.1, beberapa kotak pada papan permainan telah dibuka oleh pemain. Hal ini diperlukan karena agen membutuhkan data dari kondisi permainan yang telah berjalan untuk melakukan inferensi sehingga dapat memberikan solusi untuk penyelesaian permainan.



Gambar 6.2 Eksekusi Autoplay pada level beginner dengan data normal

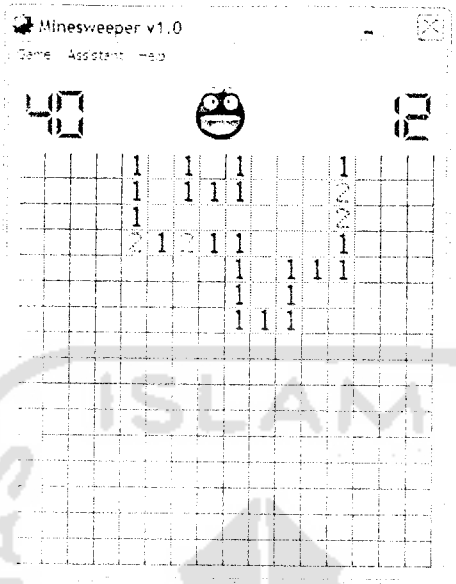
Setelah pemain mengaktifkan fitur Autoplay, agen akan menyelesaikan permainan dengan membuka semua kotak yang aman dan memberi tanda pada semua lokasi ranjau seperti yang terlihat pada gambar 6.2.

Pada gambar 6.1, kotak (0, 7) dan (1, 7) tidak dapat ditentukan nilainya oleh *propagator engine*, tetapi karena masih ada informasi yang dapat diolah, *solver engine* tidak diaktifkan.

Dari pengujian pada level *beginner* dengan menggunakan data normal, agen dapat menyelesaikan permainan dan tercatat bahwa *solver engine* tidak melakukan inferensi.

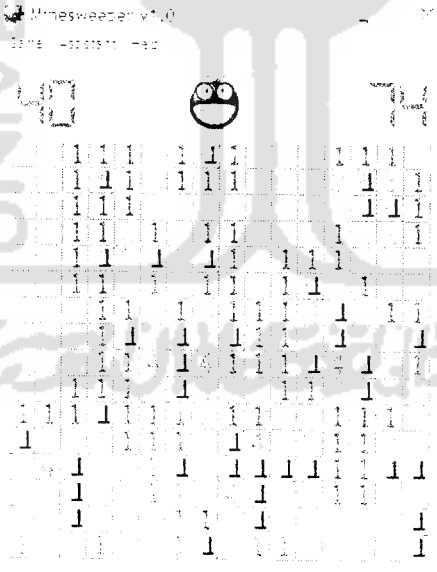
6.2.1.1.2 Pengujian Level Intermediate

Pada pengujian terhadap level *intermediate*, ukuran papan permainan yang dipakai adalah 16 X 16 dengan ranjau berjumlah 40 buah (gambar 6.3).



Gambar 6.3 Papan permainan ukuran 16 X 16 yang sudah terbuka

Pada gambar 6.3 terdapat papan permainan berukuran 16 X 16 dan terdapat 40 buah ranjau yang tersebar di dalamnya.



Gambar 6.4 Eksekusi Autoplay pada level intermediate dengan data normal

Pada gambar 6.4, agen telah diaktifkan dengan status sebagai *Autoplayer*. Dari kondisi tersebut, *solver engine* memberikan nilai peluang dari beberapa variabel yang tidak dapat di inferensikan oleh *propagator engine* (gambar 6.5).

```
CONSTRAINT (0, 12)

(0, 13) = 0.166666666666667
(1, 13) = 0.166666666666667

Nilai MAKSIMUM = 0.166666666666667
Nilai MINIMUM  = 0.166666666666667
---------------------------------------------------

CONSTRAINT (0, 14)

(0, 13) = 0.166666666666667
(1, 13) = 0.166666666666667
(1, 14) = 0.125
(1, 15) = 0.125

Nilai MAKSIMUM = 0.166666666666667
Nilai MINIMUM  = 0.125
---------------------------------------------------

CONSTRAINT (0, 15)

(1, 14) = 0.125
(1, 15) = 0.125

Nilai MAKSIMUM = 0.125
Nilai MINIMUM  = 0.125
---------------------------------------------------

CONSTRAINT (1, 12)

(0, 13) = 0.166666666666667
(1, 13) = 0.166666666666667

Nilai MAKSIMUM = 0.166666666666667
Nilai MINIMUM  = 0.166666666666667
---------------------------------------------------

CONSTRAINT (2, 15)

(1, 14) = 0.5
(1, 15) = 0.5

Nilai MAKSIMUM = 0.5
Nilai MINIMUM  = 0.5
```

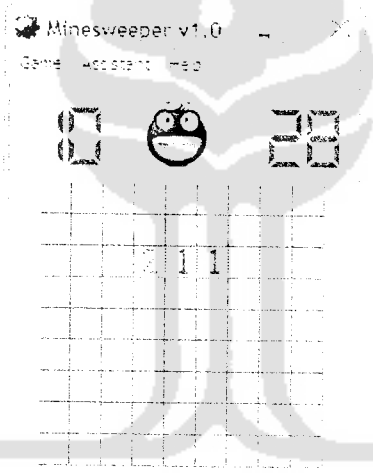
Gambar 6.5 Hasil inferensi *solver engine* pada level *intermediate* dengan data normal

Dalam pengujian ini terdapat beberapa variabel yang mempunyai nilai peluang yang sama yang menyebabkan agen tidak dapat memilih kotak mana yang akan dimainkan, sehingga agen gagal untuk menyelesaikan permainan.

6.2.1.2 Dengan Data Tidak Normal

6.2.1.2.1 Pengujian Level Beginner

Pada pengujian level *beginner* dengan menggunakan data tidak normal, kotak pada papan permainan telah dibuka sebanyak 3 buah dengan posisi berhimpitan secara horisontal (gambar 6.6).



Gambar 6.6 Papan permainan ukuran 9 X 9 untuk pengujian data tidak normal

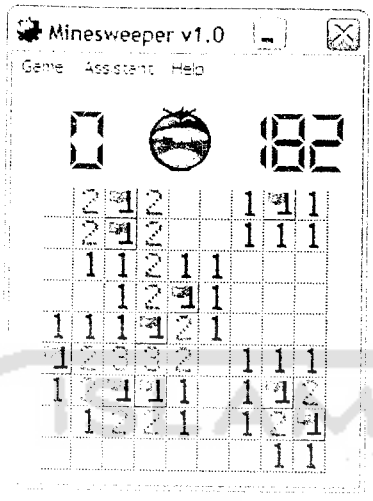
Informasi yang diberikan dari papan permainan seperti pada gambar 6.6 akan mengaktifkan *solver engine* untuk menghitung nilai peluang variabel – variabel yang tidak dapat dipecahkan oleh *propagator engine* (gambar 6.7).

```
CONSTRAINT (3, 2)

(2, 1) = 0.285714285714286
(2, 2) = 0.285714285714286
(2, 3) = 0.285714285714286
```

```
(3, 1) = 0.0476190476190476
(3, 3) = 0.0476190476190476
(4, 1) = 0.00680272108843537
(4, 3) = 0.00680272108843537

Nilai MAKSIMUM = 0.285714285714286
Nilai MINIMUM  = 0.00680272108843537
-------------------------------------------------

CONSTRAINT (4, 2)

(3, 3) = 0.0666666666666667
(4, 1) = 0.00952380952380952
(4, 3) = 0.00952380952380952
(5, 1) = 0.0285714285714286
(5, 3) = 0.0285714285714286

Nilai MAKSIMUM = 0.0666666666666667
Nilai MINIMUM  = 0.00952380952380952
-------------------------------------------------

CONSTRAINT (5, 2)

(4, 1) = 0.00952380952380952
(4, 3) = 0.00952380952380952
(5, 1) = 0.0285714285714286
(5, 3) = 0.0285714285714286
(6, 1) = 0.142857142857143
(6, 2) = 0.142857142857143
(6, 3) = 0.142857142857143

Nilai MAKSIMUM = 0.142857142857143
Nilai MINIMUM  = 0.00952380952380952
```

Gambar 6.7 Hasil inferensi *solver engine* pada level *beginner* dengan data tidak normal

Dari data yang diberikan oleh *solver engine*, kotak yang berpeluang paling besar untuk menyimpan ranjau adalah kotak dengan koordinat (4, 1) dan (4, 3) sehingga pada *constraint* (4, 2) dan (5, 2), kotak dengan peluang aman yang lebih besar akan dibuka / dimainkan oleh agen (gambar 6.8).

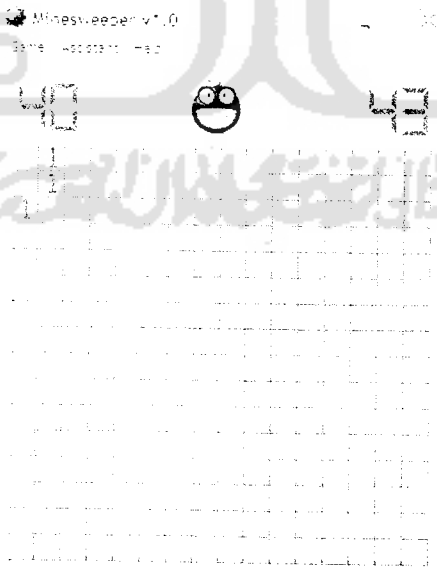


Gambar 6.8 Eksekusi *autoplay* pada level *beginner* dengan data tidak normal

Pada pengujian level *beginner* dengan data tidak normal, agen berhasil untuk menyelesaikan permainan.

6.2.1.2.2 Pengujian Level Intermediate

Pada pengujian terhadap level *intermediate* dengan menggunakan data tidak normal, telah dibuka 6 kotak pada papan permainan (gambar 6.9).



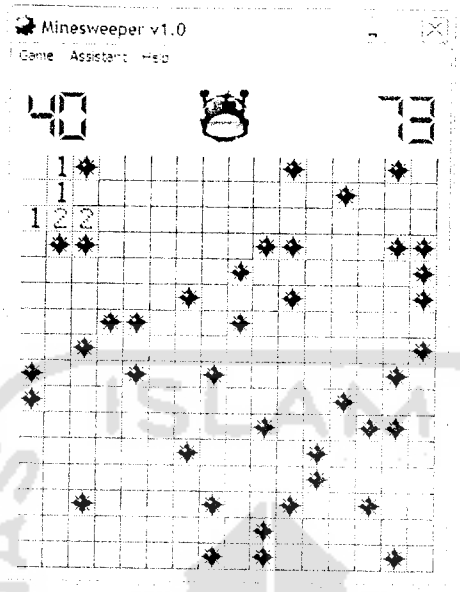
Gambar 6.9 Papan ukuran 16 X 16 untuk pengujian dengan data tidak normal

Informasi pada papan permainan seperti yang ditunjukkan pada gambar 6.8 akan memicu *solver engine* untuk menghitung nilai peluang variabel – variabel yang tidak dapat dipecahkan oleh *propagator engine* (gambar 6.10).

```
CONSTRAINT (0, 2)

(0, 3) = 0.25
(1, 3) = 0.25

Nilai MAKSIMUM = 0.25
Nilai MINIMUM  = 0.25
-----------------------------------------------------

CONSTRAINT (1, 0)

(2, 0) = 0.25
(2, 1) = 0.125

Nilai MAKSIMUM = 0.25
Nilai MINIMUM  = 0.125
-----------------------------------------------------

CONSTRAINT (1, 1)

(2, 0) = 0.25
(2, 1) = 0.125
(2, 2) = 0.1333333333333333

Nilai MAKSIMUM = 0.25
Nilai MINIMUM  = 0.125
-----------------------------------------------------

CONSTRAINT (1, 2)

(0, 3) = 0.25
(1, 3) = 0.25
(2, 1) = 0.125
(2, 2) = 0.1333333333333333
(2, 3) = 0.2

Nilai MAKSIMUM = 0.25
Nilai MINIMUM  = 0.125
```

Gambar 6.10 Hasil inferensi *solver engine* pada level *intermediate* dengan data tidak normal

Dari data yang didapatkan dari *solver engine*, kotak yang berpeluang aman terbesar pada setiap *constraint* akan dibuka (gambar 6.11).



Gambar 6.11 Eksekusi *autoplay* pada level *intermediate* dengan data tidak normal

Pada pengujian ini, agen gagal untuk menyelesaikan permainan karena membuka kotak yang di dalamnya terdapat ranjau.

6.3 Analisis Hasil Pengujian

6.3.1 Pengujian Dengan Data Normal

Pada pengujian dengan menggunakan data normal, hasil yang diperoleh adalah sebagai berikut :

1. Pada pengujian dengan menggunakan ukuran papan level *beginner*, agen berhasil untuk menyelesaikan permainan tanpa menggunakan bantuan dari *solver engine*.
2. Pada pengujian dengan menggunakan ukuran papan level *intermediate*, agen gagal menyelesaikan permainan karena tidak dapat menentukan kotak yang akan dimainkan berdasakan hasil inferensi dari *solver engine*.

Hal ini disebabkan karena variabel yang akan dipilih mempunyai nilai peluang yang sama.

Dari kedua pengujian tersebut dapat disimpulkan bahwa dengan data normal, informasi yang didapatkan oleh agen akan semakin banyak, sehingga memperkecil kemungkinan kesalahan agen dalam melakukan suatu aksi. Namun masih terdapat kemungkinan agen gagal dalam menyelesaikan permainan karena kondisi permainan akan terus berkembang dengan semakin besar ukuran papan permainan dan banyaknya jumlah ranjau yang ada.

6.3.2 Pengujian Dengan Data Tidak Normal

Pada pengujian dengan menggunakan data tidak normal, hasil yang diperoleh adalah sebagai berikut :

1. Pada pengujian dengan menggunakan ukuran papan level *beginner*, agen berhasil untuk menyelesaikan permainan walaupun informasi yang ada pada papan permainan sangat sedikit.
2. Pada pengujian dengan menggunakan ukuran papan level *intermediate*, agen gagal untuk menyelesaikan permainan karena membuka kotak yang mengandung ranjau di dalamnya.

Dari kedua pengujian tersebut dapat disimpulkan bahwa walaupun data yang didapatkan oleh agen tidak normal, masih ada kemungkinan untuk menyelesaikan permainan dengan menggunakan bantuan *solver engine*.

Dari beberapa analisis diatas, dapat disimpulkan bahwa ukuran papan, jumlah ranjau, dan jumlah informasi yang bisa didapatkan akan mempengaruhi peluang kebenaran suatu aksi yang dilakukan oleh agen. Hal ini terbukti pada pengujian

dengan data tidak normal, agen dapat menyelesaikan permainan pada ukuran papan dan jumlah ranjau yang sedikit.

6.4 Kelebihan dan Kekurangan Sistem

Kelebihan dari sistem yang dibangun adalah :

1. Sistem dilengkapi dengan agen yang dapat membantu pemain dalam menyelesaikan permainan *Minesweeper*.
2. Terdapat tiga level yang berbeda yang dapat dimainkan dan pilihan menu *Custom Game* yang memungkinkan pemain untuk memilih ukuran papan dan jumlah ranjau permainan yang akan dimainkan.
3. Agen dapat menyelesaikan permainan tanpa campur tangan dari manusia.
4. Sistem menyediakan antarmuka yang menarik bagi pemain.
5. Terdapat menu *Sound* untuk memainkan bunyi sebagai peringatan bagi pemain saat melakukan suatu aksi.

Kekurangan dari sistem yang dibangun adalah :

1. Tidak terdapat fungsi untuk menampilkan *scrollbar* secara otomatis apabila ukuran papan permainan melebihi ukuran dari layar pengguna, sehingga nilai tinggi dan lebar maksimal papan permainan yang dapat dipilih oleh pemain pada pilihan menu *custom game* dibatasi hanya sebanyak 50 kotak.
2. Agen masih dapat melakukan kesalahan dalam melakukan aksi yang didasarkan pada nilai peluang yang diberikan oleh *solver engine*.

BAB VII

SIMPULAN DAN SARAN

7.1 Simpulan

Setelah dilakukan pengujian, dapat ditarik kesimpulan sebagai berikut :

1. Agen dapat menyelesaikan permasalahan dalam permainan *Minesweeper* dengan baik menggunakan *Propagator Engine*, namun tidak optimal saat menghadapi kondisi tertentu dan harus menentukan pilihan kotak yang akan dimainkan berdasarkan nilai peluang yang diberikan oleh *Solver Engine*.
2. *Set Propagator* yang digunakan dalam *Propagator Engine* sangat cocok digunakan dalam mencari solusi dalam permainan *Minesweeper*.
3. Algoritma yang digunakan dalam *Solver Engine* akan menentukan efektifitas agen dalam mengambil sebuah aksi atas kondisi yang tidak dapat dipecahkan oleh *Propagator Engine*.
4. Permainan *Minesweeper* mengandung permasalahan *non-trivial* yang pada saat tertentu tidak dapat dipecahkan secara matematis.

7.2 Saran

Saran untuk pengembangan dan penelitian selanjutnya adalah :

Para pengembang selanjutnya diharapkan dapat memberikan / menambah fungsi yang lebih lengkap terutama untuk batasan ukuran papan permainan yang merupakan salah satu kekurangan dari sistem yang telah dibangun, sehingga permainan dapat menjadi semakin menarik dan agen dapat diuji kemampuannya pada kondisi yang lebih kompleks.

DAFTAR PUSTAKA

[AD01] Adil, Qureshi, M, The Evolution of Agents, University of London, England, 2001.

[BAR06] Bartak, R, Constraint Propagation and Backtracking-based Search, Charles University, Czech Republic, 2006.

[COL01] Collet, R, Playing The Minesweeper With Constraints, Universite Catholique de Louvain, Belgium, 2001.

[MUL01] Muller, T, Constraint Propagation in Mozart, Universitat dees Saarlandes, Germany, 2001.

[VAN02] Van Roy, P, Haridi, S, Concepts, Techniques, and Models of Computer Programming, Swedish Institute of Computer Science, Sweden, 2002.

[WI06] WikiPedia, Constraint Propagation, http: www.wikipedia.org, diakses tanggal 23 Juni 2006.

[WI06a] Wikipedia, Constraint Satisfaction, http: www.wikipedia.org, diakses tanggal 23 Juni 2006.