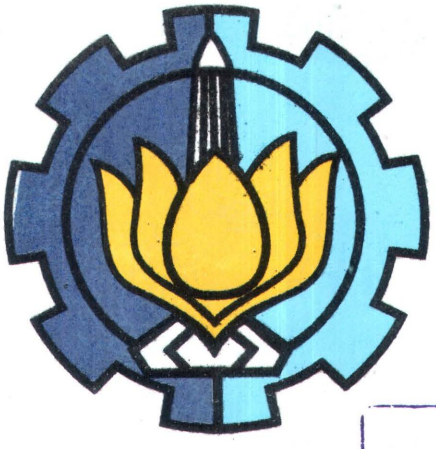
PERANCANGAN DAN PEMBUATAN PERANGKAT LUNAK *ROLE PLAYING GAME* BERBASIS WEB DENGAN TEKNOLOGI *JAVA APPLET*

TUGAS AKHIR



Disusun Oleh :

YUDHI KURNIAWAN
NRP : 5197 100 027

JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2003

PERANCANGAN DAN PEMBUATAN PERANGKAT LUNAK *ROLE PLAYING GAME* BERBASIS WEB DENGAN TEKNOLOGI *JAVA APPLET*

TUGAS AKHIR

Diajukan Guna Memenuhi Sebagian Persyaratan

Untuk Memperoleh Gelar Sarjana Komputer

Pada

Jurusan Teknik Informatika

Fakultas Teknologi Informasi

Institut Teknologi Sepuluh Nopember

Surabaya

Mengetahui/Menyetujui :

Dosen Pembimbing I	Dosen Pembimbing II
Ir. Aris Tjahyanto, M.Kom.	Ir. Suhadi Lili
NIP. 131 933 299	NIP. 132 048 148

DEPARTEMEN PENDIDIKAN NASIONAL INSTITUT TEKNOLOGI SEPULUH NOPEMBER FAKULTAS TEKNOLOGI INFORMASI JURUSAN TEKNIK INFORMATIKA

SURABAYA
Februari, 2003

ABSTRAK

Saat ini game berbasis web sudah menjadi tren dalam dunia internet karena selain praktis, juga pemain tidak perlu membeli produk game tersebut secara langsung. Selain itu game berbasis web juga berperan dalam meminimalkan pembajakan produk, karena saat ini pembajakan CD dan DVD sudah merajalela dimana-mana. Game berbasis web yang ada saat ini kebanyakan berjenis strategi, pemeliharaan, dan aksi. Sedangkan jenis Role Playing Game yang berbasis web masihlah sangat jarang. Role Playing Game sendiri artinya adalah suatu jenis permainan dimana pemain mengambil alih peran karakter utama dalam petualangannya di permainan serta mengontrol setiap aspek dari karakter tersebut. Dalam tugas akhir ini akan dirancang suatu aplikasi Role Playing Game yang dapat dijalankan dalam lingkungan internet. Aplikasi ini terdiri dari aplikasi server dan aplikasi klien yang dikembangkan dalam bahasa pemrograman Java. Permasalahan yang dihadapi dalam pengerjaan tugas akhir ini antara lain perancangan proses dalam game, penanganan area penjelajahan, pembuatan dunia permainan, penyimpanan status permainan, serta penanganan keamanan aplikasi.

Metode-metode yang digunakan dalam pembuatan game ini antara lain 2D tile based engine yakni membagi suatu area image menjadi petak-petak bernilai. Untuk penanganan animasi digunakan metode double buffering serta media tracker. Dilakukan pula pendekatan pemrograman berorientasi obyek dalam mengimplementasikan aplikasi game ini dengan memilah-milah obyek-obyek yang ada dalam permainan. Untuk komunikasi antara game dengan server digunakan metode data stream pada obyek yang terserialisasi melalui socket TCP.

Aplikasi yang dihasilkan pada tugas akhir ini telah diuji coba sesuai dengan kebutuhan sistem dan target kemampuan yang harus dimiliki oleh aplikasi Role Playing Game berbasis web yakni mampu melakukan komunikasi data informasi antara server dengan klien game. Dengan demikian Role Playing Game bisa dijadikan sebagai alternatif dalam pengembangan game berbasis web.

KATA PENGANTAR

Segala puji dan syukur semata ditujukan ke hadirat Allah SWT yang telah memberikan rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan Tugas Akhir berjudul "**Perancangan dan Pembuatan Perangkat Lunak Role Playing Game Berbasis Web Dengan Teknologi Java Applet**".

Mata kuliah Tugas Akhir dengan beban sebesar 4 satuan kredit disusun dan diajukan sebagai salah satu syarat untuk menyelesaikan program strata satu (S-1) pada jurusan Teknik Informatika di Institut Teknologi Sepuluh Nopember Surabaya.

Dalam penyusunan Tugas Akhir ini, Penulis berusaha untuk menerapkan ilmu yang telah didapat selama menjalani perkuliahan dengan tidak terlepas dari petunjuk, bimbingan, bantuan dan dukungan berbagai pihak.

Penulis menyadari bahwa Tugas Akhir ini masih jauh dari kesempurnaan dan mengandung kekurangan di sana-sini sehingga dengan segala kerendahan hati Penulis masih dan insya Allah akan terus mengharap saran serta kritik yang membangun dari rekan-rekan pembaca.

Surabaya, Februari 2003

UCAPAN TERIMA KASIH

Dengan mengucapkan syukur Alhamdulillah kepada Allah SWT, pada kesempatan ini Penulis hendak menyampaikan penghormatan dan terima kasih kepada pihak-pihak yang telah memberi bantuan baik itu berupa moril maupun material dan langsung maupun tidak langsung kepada :

1. Mama dan Papa tercinta, Tatiek Hairani dan Imam Sudjarwo atas segala do'a, bimbingan dan kasih sayang selama ini. Juga untuk adikku Viddy Riyanti.
2. Bapak Ir. Aris Tjahyanto,M.Kom. dan Bapak Ir. Suhadi Lili yang telah membimbing penulis dalam menyelesaikan Tugas Akhir ini.
3. Bapak dan Ibu dosen Jurusan Teknik Informatika – ITS; Pak Agus Zainal., Bu Esther H., Pak Febriliyan S., Pak Dwi S., Pak Waskitho W., Pak Hari G., Pak Rully S dan para dosen lainnya yang telah berkenan menularkan ilmunya selama Penulis menempuh kuliah.
4. Tim desain grafik dan animasi game yakni Saudara Robi Kristian dan Saudara Arie Affianto yang telah sangat membantu dalam pembangunan aplikasi game ini.
5. Sahabatku Yenny Kurniawati atas pinjaman scanner-nya yang sangat membantu dalam penyelesaian desain grafik dan animasi game ini serta dukungan morilnya selama ini.
6. Saudara Dody Rahmat W. atas pinjaman referensi buku yang menunjang pengerjaan tugas akhir ini dan Saudaraku Vera Meliana atas dukungan morilnya selama ini.

7. Sahabat-sahabatku sebagai sesama warga RPL; Yuniawan, Rezin, Gae Siang, Henky, Dewa, Deddy, Sofiah, Hadian, dan Maya atas dukungannya serta warga RPL yang lain. Hidup RPL!!!
8. Saudara Agung Prabowo atas bimbingan pemrograman Java dan Socket-nya sehingga akhirnya aplikasi game ini dapat diimplementasikan.
9. Saudara Aryo dan Ronnie atas hardware support-nya.
10. Teman-temanku sesama penghuni Perumdos Blok U-133; Agus, Hendik, Sigit, serta lainnya.
11. Teman-temanku angkatan C0D; Wahyu, Komo, Novan, Suki, Juned, Gershom, Arief, Vicka, Dhanie, Winnie, serta lainnya.
12. Kakak-kakak angkatan C0C; Darlis, Andik, Ajie, Arman, Bowo, serta lainnya.
13. Adik-adikku angkatan C0E; Budianto, Ririn, Leo, Arief, Isye, Ario, Trisno, serta lainnya.
14. Adik-adikku angkatan C0F; Nuning, Arman, Indra, Eri, Rully, serta lainnya.
15. Seluruh staf dan karyawan Jurusan Teknik Informatika – ITS; Mas Yudi, Mas Soleh, Pak Mu'in, Mbak Davi, Pak Karmono dan para karyawan lainnya.
16. Rekan-rekan lain yang tidak dapat Penulis sebutkan satu-persatu.

Tiada untaian kata yang cukup yang dapat penulis sampaikan sebagai balas atas jasa yang penulis terima melainkan hanya harapan semoga Allah SWT membalas semua amal tersebut.

DAFTAR ISI

DAFTAR GAMBAR

BAB I

PENDAHULUAN

BAB I

PENDAHULUAN

1.1 LATAR BELAKANG

Tugas akhir ini dilatarbelakangi oleh kenyataan-kenyataan yang ada saat ini bahwa game berbasis web sudah menjadi tren dalam dunia internet karena selain praktis, juga pemain tidak perlu membeli produk game tersebut secara langsung. Selain itu game berbasis web juga berperan dalam meminimalkan pembajakan produk, karena saat ini pembajakan CD dan DVD sudah merajalela dimana-mana. Game berbasis web juga memudahkan dalam meng-update bagian dari permainan yang bisa dilakukan oleh administrator pada server game.

Game berbasis web yang ada saat ini kebanyakan berjenis strategi, pemeliharaan, dan aksi. Sedangkan jenis *Role Playing Game* yang berbasis web masihlah sangat jarang.

Role Playing Game sendiri artinya adalah suatu jenis permainan dimana pemain mengambil alih peran karakter utama dalam petualangannya di permainan serta mengontrol setiap aspek dari karakter tersebut. Karakter dalam *Role Playing Game* biasanya berdasarkan pada status dan kemampuan. Status adalah hal-hal seperti kekuatan, ketangkasan, kepandaian, dan sebagainya. Status ini dipresentasikan dalam angka, yang menunjukkan seberapa berkembang status tersebut.

Role Playing Game biasanya adalah game dengan gaya 2D isometri. Ia menggunakan sprite untuk grafik yang merepresentasikan karakter serta makhluk

lain. Sprite juga digunakan untuk sesuatu seperti pohon, sungai, bangunan, dan item seperti senjata, pakaian, harta, dan sebagainya.

1.2 TUJUAN

Tujuan pembuatan tugas akhir ini adalah menghasilkan suatu perangkat lunak *Role Playing Game* yang dapat dijalankan dalam lingkungan internet.

Manfaat dari pembuatan tugas akhir ini adalah :

- Mengenalkan bagaimana membangun suatu *Role Playing Game.*
- Menunjukkan bahwa *Role Playing Game* bisa dikembangkan sebagai game berbasis web.

1.3 PERMASALAHAN

Permasalahan dalam pengerjaan tugas akhir ini adalah :

- Perancangan proses dalam game.
- Penanganan area penjelajahan tempat karakter dalam game bertualang dalam permainan.
- Pembuatan dunia permainan.
- Penyimpanan status permainan pada server game.
- Penanganan komunikasi ketiga komponen permainan di atas(area penjelajahan, dunia permainan dan status permainan) antara server dan klien game.
- Penanganan keamanan penggunaan aplikasi game.

1.4 BATASAN MASALAH

Adapun batasan masalah dalam tugas akhir ini adalah:

- Penyimpanan status permainan hanya dapat dilakukan pada tempat-tempat tertentu dalam permainan.
- Diasumsikan jaringan komputer yang akan digunakan dalam menjalankan applet tersebut mempunyai kecepatan transfer data yang tinggi yakni di atas 100 Kbps.
- Perubahan rancangan pengaturan dunia permainan dilakukan secara berkala yang ditentukan sendiri oleh administrator game.
- Bahasa pemrograman yang akan digunakan dalam pembuatan perangkat lunak game ini adalah *Java* karena sifatnya yang *cross platform* (tidak tergantung pada jenis sistem operasi maupun jenis browser untuk menjalankannya) sehingga sesuai untuk lingkungan internet.

1.5 METODOLOGI

Pembuatan tugas akhir ini dilakukan dengan mengikuti metodologi sebagai berikut :

a. Studi Literatur

Pada tahap ini dipelajari konsep-konsep tentang teknologi *Java Applet* serta distribusi obyek beserta hal-hal yang melatarbelakanginya. Konsep ini didapat dari buku(terdapat dalam daftar pustaka).

Disamping itu, pada tahap ini juga dipelajari metodologi dan algoritma yang akan digunakan dalam pembuatan perangkat lunak sehingga membantu pada tahap perancangan perangkat lunak.

b. Perancangan Sistem dan Aplikasi

- Perancangan *Use-Case*

Pada tahap ini akan ditentukan fungsi-fungsi apa saja yang diberikan oleh sistem, dalam hal ini *Role Playing Game*.

- Perancangan Obyek

  Pada tahap ini akan ditentukan obyek-obyek yang berperan dalam membangun sistem game.

- Perancangan Arsitektur

  Pada tahap ini dibuat rancangan arsitektur baik di sisi klien maupun di sisi server. Arsitektur ini akan dibuat secara *user friendly* sehingga memudahkan baik dalam hal pengoperasian(di server) maupun kemudahan pengendalian game(di klien).

- Perancangan *Model Deployment*

  Pada tahap ini dibuat gambaran distribusi komponen dalam game yang meliputi pemrosesan elemen serta proses perangkat lunak yang ada di dalamnya.

c. Pembuatan Aplikasi

Dalam tahap ini, dilakukan implementasi berdasarkan rancangan yang telah dibuat pada tahap sebelumnya.

Perangkat lunak yang dihasilkan ada dua buah yakni yang berjalan di klien dengan teknologi *Java Applet* serta di server dengan teknologi *Java Application*.

d. Uji Coba dan Evaluasi

Pada tahap ini aplikasi yang telah dibuat akan diuji dengan parameter sebagai berikut:

- Tidak adanya bug dalam permainan
- Manajemen penyimpanan data semua klien pada basis data

e. Penyempurnaan Arsitektur

Pada tahap ini, segala arsitektur disempurnakan seperti pembuatan animasi, desain grafis obyek, serta tata suara.

f. Penyusunan Buku Tugas Akhir

Pada tahap terakhir ini disusun buku sebagai dokumentasi dari pelaksanaan tugas akhir.

1.6 SISTEMATIKA PEMBAHASAN

Pembahasan dalam Tugas Akhir ini dibagi menjadi beberapa bab sebagai berikut :

- Bab I, Pendahuluan, berisi latar belakang, permasalahan, tujuan, batasan masalah, metodologi dan sistematika pembahasan.
- Bab II, Teori Penunjang, akan dibahas dasar ilmu yang mendukung pembuatan Tugas Akhir ini, seperti internet, *2D tile based engine*, konsep jaringan dan pemrograman aplikasi terdistribusi, serta pemrograman *thread* dan *applet* pada bahasa Java.
- Bab III, Perancangan Sistem. Pada bab ini akan dibahas sistem yang menjadi penerapan aplikasi yaitu *Role Playing Game*. Dari rancangan sistem ini akan dibuat desain perangkat lunaknya untuk implementasi.
- Bab IV, Implementasi Perangkat Lunak. Pada bab ini akan dilakukan pembahasan mengenai pembuatan perangkat lunak, baik itu berupa aplikasi

yang berjalan di sisi *server* maupun aplikasi di sisi *client* yang dapat berjalan di atas *browser*.

- Bab V, Uji Coba Perangkat Lunak, akan dilakukan uji coba berdasarkan parameter-parameter yang ditetapkan, dan kemudian dilakukan analisa terhadap hasil uji coba tersebut.
- Bab VI, Penutup, berisi kesimpulan yang dapat diambil dari Tugas Akhir ini beserta saran untuk pengembangan selanjutnya.

BAB II

TEORI PENUNJANG

BAB II

TEORI PENUNJANG

Dalam bab ini dibahas dasar teori yang menunjang pembuatan Tugas Akhir. Yang pertama dibahas tentang konsep *Role Playing Game* serta metode *2D tile based engine*. Selanjutnya dibahas tentang konsep pemrograman *thread* dan *applet*, metode *double buffering*, serta aplikasi terdistribusi dan penerapannya jika menggunakan bahasa Java. Pembahasan terakhir adalah mengenai UML, notasi untuk memodelkan rancangan aplikasi dalam Tugas Akhir ini.

2.1 *Role Playing Game*

Role Playing Game adalah salah satu dari sekian banyak jenis permainan yang ada saat ini dalam dunia game.

2.1.1 Definisi

Role Playing Game adalah suatu jenis permainan dimana pemain mengambil alih peran sebagai karakter utama dalam petualangannya di permainan serta mengontrol setiap aspek dari karakter tersebut."(Perry, Jim (Machaira); VB RPG Tutorial; http://rookscape.com/vbgaming/vbrpgtut/vbrpgtut.php). Karakter dalam *Role Playing Game* biasanya berdasarkan pada status dan kemampuan. Status adalah hal-hal seperti kekuatan, ketangkasan, kepandaian, dan sebagainya. Status ini dipresentasikan dalam angka, yang menunjukkan seberapa berkembang status tersebut.

2.1.2 Pengelompokan

Role Playing Game dikelompokkan berdasarkan sudut pandang pemain terhadap permainan tersebut. Untuk saat ini telah terdapat tiga kelompok RPG yang akan dijelaskan di bawah ini.

2.1.2.1 *Tile Based*

Pada kelompok ini, sudut pandang pemain terhadap permainan dari atas kepala karakter game tersebut. Jadi diibaratkan pemain melihat area permainan dari atas langit. Area tempat bertualang karakter game dibagi secara berpetak-petak dan setiap nilai pada petak mewakili suatu obyek. Contoh game dari kelompok ini adalah Final Fantasy 6.

2.1.2.2 *First Person*

Pada kelompok ini, permainan dilihat dari sudut pandang diri pemain itu sendiri. Dalam hal ini sudut pandang karakter game diwakili oleh pemain. Jadi apa saja yang dilihat oleh pemain dalam permainan menentukan petualangan dalam game tersebut. Contoh game dari kelompok ini adalah seri Might and Magic.

2.1.2.3 *3D*

Pada kelompok ini, sudut pandang pemain terhadap permainan bisa dari berbagai sisi dari karakter game tersebut. Pemain bisa memutar-mutar kamera sudut pandang untuk mendapatkan sisi pandang yang sesuai dengan keinginan pemain itu sendiri. Contoh game dari kelompok ini adalah Grandia 2.

2.2 Studi Banding

Sebelum memasuki tahap perancangan, akan dibahas mengenai studi banding dengan beberapa *Role Playing Game* yang telah ada. Disini akan diambil tiga sampel game yakni Final Fantasy 6, Breath of Fire 4, dan Grandia 2.

2.2.1 Final Fantasy 6

Game ini dirilis pada tahun 1994 pada konsol Super Nintendo oleh Squaresoft. Game Final Fantasy merupakan *Role Playing Game* dengan seri terbanyak dibandingkan dengan beberapa game lain yang sejenis. Sebegitu fenomenalnya sehingga saat ini telah dirilis sampai seri ke-11.

Petualangan karakter dalam game ini dilakukan dalam sebuah dunia yang luas. Pada dunia tersebut, karakter bisa singgah di berbagai tempat termasuk daerah rahasia. Dalam perjalanan di dunia tersebut karakter juga bisa berjumpa dengan *monster* sehingga akan terjadi pertempuran disana. Pada bagian akhir cerita, pemain bisa menggunakan pesawat udara dalam menjelajahi dunia dan untuk menjangkau tempat-tempat tertentu.

Sistem perlengkapan karakternya meliputi *R-hand*, *L-hand*, *head*, dan *body*. Masing-masing dari *R-hand* dan *L-hand* bisa dipasang senjata atau perisai. Jadi setiap karakter bisa menggunakan senjata dan perisai atau dua senjata sekaligus. *Head* digunakan untuk memasang helm, sedangkan *body* digunakan untuk memasang baju. Setiap pemasangan perlengkapan tersebut akan mempengaruhi status daripada karakter tersebut.

Untuk menggunakan sihir, setiap karakter harus memasang esper pada dirinya. Esper merupakan makhluk sihir yang memiliki kemampuan yang berbeda-beda dan dapat mempelajari sihir. Kemampuan setiap esper dapat digunakan dalam

pertempuran. Setiap esper juga dapat mempelajari sihir dari poin yang didapat dari pertempuran. Sihir-sihir yang dipelajari oleh masing-masing esper berbeda-beda. Sihir-sihir tersebut nantinya digunakan oleh karakter yang memasang esper tersebut pada dirinya.

Setiap karakter memiliki keahlian yang berbeda antara satu dengan yang lain. Keahlian tersebut antara lain mencuri item lawan, mengeluarkan jurus kombinasi, melompat, mengkopi lawan, berubah bentuk, dan sebagainya.

Untuk mendapatkan kemampuan tambahan, setiap karakter dapat memasang relic pada dirinya. Setiap relic dapat memberikan kemampuan tambahan pada karakter yang mengenakannya antara lain menggunakan dua senjata sekaligus, kebal terhadap racun, membalas serangan lawan, dan sebagainya.

Status dari karakternya selain hp dan mp adalah *vigor*, *speed*, *stamina*, *magic power*, battle *power*, *defense*, *evade*, *magic defend*, dan *magic* block. *Vigor* adalah nilai yang menyatakan kecepatan karakter dalam melakukan aksi ketika perintah telah diberikan oleh pemain. *Speed* adalah nilai yang menyatakan kecepatan waktu tunggu karakter untuk menerima perintah dari pemain. *Stamina* adalah nilai yang menyatakan daya tahan karakter secara keseluruhan. *Magic power* adalah nilai yang menyatakan kekuatan sihir yang digunakan oleh karakter. Battle *power* adalah nilai yang menyatakan kekuatan serangan karakter dalam pertempuran. *Defense* adalah nilai yang menyatakan daya tahan karakter terhadap serangan lawan dalam pertempuran. *Evade* adalah nilai yang menyatakan kemampuan karakter untuk mengelak serangan lawan. *Magic defend* adalah nilai yang menyatakan daya tahan karakter terhadap serangan sihir lawan. *Magic* block adalah nilai yang menyatakan kekebalan karakter terhadap sihir yang menyebabkan status tidak normal pada karakter.

Monster dalam game ini tidak nampak dilayar sehingga pertemuan dengan *monster* ditentukan oleh langkah serta status karakter. Pertempuran dalam game ini bisa melibatkan sampai empat karakter. Sistem pertempurannya menggunakan sistem tunggu berdasarkan kecepatan waktu tunggu tiap karakter. Pemain bisa memasukkan perintah bila karakter telah menyelesaikan waktu tunggunya dan kemudian karakter tersebut akan segera melakukan aksinya berdasarkan kecepatan aksinya.

2.2.2 Breath of Fire 4

Game ini dirilis pada tahun 2000 pada konsol Sony Playstation oleh Capcom. Pada seri-seri awal game ini dirilis di Super Nintendo. Pada setiap serinya selalu menampilkan tokoh Ryu sang naga dan Nina putri kerajaan burung Windia.

Pada seri keempat ini dalam melakukan petualangan, pemain disuguhi peta dunia yang di dalamnya terdapat *state-state* yang nantinya akan disinggahi oleh karakter game tersebut. Karakter tidak dapat bertualang sebebas dalam game Final Fantasy, namun hanya bisa bertualang di dalam *state* tersebut.

Perlengkapan karakter meliputi senjata, baju, dan helm. Setiap perlengkapan tersebut akan mempengaruhi status daripada karakter yang mengenakannya.

Untuk mempelajari sihir, karakter harus menahan serangan sihir lawan. Setiap karakter bisa mempelajari beberapa sihir tertentu dan sihir tersebut baru bisa dipelajari bila cocok dengan karakternya.

Setiap karakter memiliki kemampuan berbeda-beda dan kemampuan tersebut bisa digunakan untuk berbagai keperluan misalnya Ryu bisa membabat rumput untuk dapat dilewati atau Nina bisa terbang untuk melihat keadaan sekitar.

Status dari karakternya selain hp dan mp yakni *power*, *defend*, *agility*, dan wisdom. *Power* adalah nilai yang menyatakan daya serang karakter dalam pertempuran. *Defend* adalah nilai yang menyatakan daya tahan karakter dalam pertempuran. *Agility* adalah nilai yang menyatakan kecepatan karakter untuk melakukan aksinya. Wisdom adalah nilai yang menyatakan kekuatan sihir karakter.

Monster dalam game ini juga tidak nampak dalam layar karena pertemuan dengan *monster* ditentukan oleh langkah karakter. Pertempuran dalam game ini bisa menggunakan seluruh karakter dalam tim. Tiga karakter yang berada di depan berhadapan langsung dengan *monster* dan yang melakukan aksinya dalam pertempuran sedangkan sisanya merupakan karakter cadangan. Pemain bisa langsung menukar karakter di depan dengan karakter cadangan pada saat pertempuran bila ingin menyelamatkan karakter yang hp-nya kritis. Pertempuran dalam game ini menggunakan sistem antrian, maksudnya sudah ditentukan urutan untuk memasukkan perintah untuk karakter. Pemain langsung memasukkan perintah untuk seluruh karakter di posisi depan berdasarkan urutannya kemudian setelah itu aksi akan dilakukan oleh karakter berdasarkan kecepatan aksinya.

2.2.3 Grandia 2

Game ini dirilis pada tahun 2001 pada konsol Sega Dreamcast dan PC oleh GameArts. Prekuelnya dirilis di konsol Sega Saturn. Game ini termasuk *Role Playing Game* pendatang baru yang bisa langsung disejajarkan dengan RPG papan atas. Game ini menonjolkan keunikan pada sistem pertempurannya.

Petualangan karakter dalam game ini tidak sebebas Final Fantasy karena tempat-tempat yang ditampilkan dan bisa disinggahi oleh karakter telah ditentukan berdasarkan alur cerita game.

Perlengkapan karakter meliputi senjata, baju, tutup kepala, alas kaki, aksesoris, dan *mana egg*. Senjata akan mempengaruhi daya serang karakter. Baju akan mempengaruhi daya tahan karakter. Tutup kepala mempengaruhi daya tahan karakter. Alas kaki mempengaruhi pergerakan karakter. Aksesoris akan memberikan kemampuan tambahan pada karakter. *Mana egg* akan memberikan kemampuan sihir pada karakter.

Setiap karakter memiliki gerakan spesial yang berbeda-beda. Gerakan spesial mengkonsumsi sp, tidak seperti halnya sihir yang mengkonsumsi mp.

Untuk mempelajari sihir dan gerakan spesial, pemain harus memperolehnya dengan koin. Dalam setiap pertempuran akan diperoleh koin spesial dan koin sihir. Koin spesial digunakan untuk memperoleh serta meningkatkan level gerakan spesial. Koin sihir digunakan untuk memperoleh serta meningkatkan level sihir pada *mana egg*. Setiap *mana egg* menyediakan sihir yang berbeda-beda.

Status karakternya meliputi *strength*, *vitality*, *agility*, *speed*, *magnitude* dan *mentality*. *Strength* adalah nilai yang menyatakan daya serang karakter. *Vitality* adalah nilai yang menyatakan daya tahan karakter. *Agility* adalah nilai yang menyatakan kecepatan waktu tunggu karakter untuk menerima perintah. *Speed* adalah nilai yang menyatakan kecepatan karakter dalam melakukan aksinya. *Magnitude* adalah nilai yang menyatakan jarak serangan karakter terhadap *monster*. *Mentality* nilai yang menyatakan kecerdasan karakter dalam mencari musuh.

Pertemuan dengan *monster* terjadi bila karakter menabrak *monster* karena dalam game ini *monster* akan nampak di layar. Total maksimum karakter yang bisa turut serta dalam pertempuran empat orang. Pertarungan dalam game ini menggunakan sistem yang agak unik yakni semi realtime sehingga jarak antara karakter terhadap *monster* mempengaruhi hasil serangan. Sistem tunggu

pertarungannya mirip game Final Fantasy yakni karakter memperoleh giliran menunggu perintah bila waktu tunggunya telah habis. Setelah karakter menerima perintah dari pemain karakter akan melakukan aksinya setelah waktu tunggu aksinya habis. Perbedaannya adalah bila aksinya menyerang maka karakter tersebut akan mengejar *monster* yang menjadi targetnya dan akan melakukan serangan bila telah sampai. Karakter tersebut akan berhenti mengejar bila jarak serangannya terlampau jauh. *Monster* juga bisa mendahului membalas serangan pada saat karakter tersebut menyerang. Ketika *monster* target mati di tengah serangan, karakter tersebut akan segera mencari target lain berdasarkan kecerdasannya dalam mencari target.

2.3 *2D Tile Based Engine*

Pada dasarnya, *tile based engine* biasanya adalah sudut pandang dari atas kepala terhadap area dimana karakter dalam game melakukan petualangan."(DarkDread; RPGs in QBASIC Tutorial; April, 1997 http://www.geocities.com/SiliconValley/Pines/1732/tut.htm). Area yang nantinya menjadi tempat bertualang karakter tersebut akan dibagi secara berpetak-petak dan setiap nilai pada petak mewakili suatu kondisi tertentu.

2.3.1 Umum

Dalam permainan *Role Playing Game*, karakter utama dalam game tersebut akan melakukan banyak petualangan di berbagai area yang berbeda-beda baik secara geografis maupun tata letak. Area tersebut bisa berupa suatu kota, hutan, pegunungan, padang pasir, dan sebagainya. Tentunya di setiap area tersebut terdapat berbagai obyek yang menjadi unsur darinya seperti rumah, orang, pohon,

sungai, jembatan, gunung, dan sebagainya. Nantinya karakter dalam permainan tersebut akan berinteraksi dengan obyek-obyek diatas dalam petualangannya.

2.3.2 Prinsip Kerja *Tile Based Engine*

Tile based engine dilakukan dengan membagi suatu area menjadi berpetak-petak yang ukurannya disesuaikan dengan ukuran lebar dan tinggi gambar. Biasanya satuan ukuran yang digunakan adalah *pixel* (titik). Semakin rapat petak-petak yang dihasilkan maka semakin tajam pula tingkat ketelitiannya.

Setiap petak yang dihasilkan akan diisi dengan suatu nilai tertentu yang fungsinya untuk mewakili suatu kondisi tertentu pada area tersebut. Beberapa kondisi tersebut antara lain transaksi pada toko, membuka harta karun, mengaktifkan area selanjutnya, dan sebagainya. Nilai-nilai tersebut juga berfungsi untuk menentukan bagian-bagian area yang bisa dilalui atau tidak oleh karakter.

2.4 *Thread*

Sebuah program dengan kendali aliran tunggal disebut program sekuensial. Pada satu waktu eksekusi program, komputer mengeksekusi pada satu titik tunggal (*single point*). Di sisi lain, sebuah program dengan banyak titik eksekusi (*multiple points*) disebut program konkuren.

Terminologi *thread* diturunkan dari frasa *thread of execution* (eksekusi *thread*). Pada setiap instan waktu dalam *thread* tunggal, ada satu titik tunggal eksekusi. Menggunakan banyak *thread* dalam satu program berarti program memiliki banyak titik eksekusi pada setiap instan yang diberikan.

Thread dapat menciptakan *thread* lain dan mematikannya. *Thread* yang tercipta lebih baru akan dijalankan pada *address space* yang sama, sehingga memungkinkan untuk saling berbagi data.

Ketika *thread* memberi gambaran bahwa dua atau lebih even terjadi pada saat yang sama, komputer (diasumsikan hanya memilik satu CPU) sebenarnya hanya dapat mengeksekusi satu *thread* pada satu waktu. Gambaran bahwa *thread* dieksekusi secara simultan adalah akibat dari pengalihan secara cepat dari satu *thread* ke yang lain. Tiap *thread* dieksekusi untuk satu periode waktu yang singkat, lalu *thread* itu melemparkan kendali ke *thread* berikutnya.

2.5 *Applet*

Applet tidak lain merupakan kelas Java yang mewarisi kelas Applet API Java dan mengimplementasikan beberapa *method* tertentu untuk menangani pemunculannya dan untuk menanggapi event yang disampaikan kepadanya.

Applet Java ditangani pada halaman HTML sebagai sebuah elemen *inlined*, yaitu menyatu dengan halaman *web*, bukan di *window* yang terpisah. *Browser* akan membaca *file* HTML tersebut dan memeriksa apakah ia perlu mengambil data yang berkaitan dengan elemen applet ini. Sebagian besar *browser* (terutama yang *Java-capable*) bersifat *multithreaded*. Artinya, sementara *browser* akan melanjutkan menangani data HTML lainnya dalam satu *thread*, iapun membuat *thread* baru untuk mengambil data elemen *inlined*. Untuk kasus applet, data ini berupa kelas Java. Agar *browser* dapat mengambil suatu kelas Java, maka ia perlu mengimplementasikan satu kelas Java tertentu. *Class loader* akan mengambil kelas yang diindikasikan oleh *tag* applet di HTML melintasi jaringan.

Applet sepenuhnya memutuskan data-data gambar, suara atau yang lain yang perlu diambil. Pada tahapan ini, applet memberi perintah secara langsung kepada *browser* untuk menangani apa-apa yang diperlukannya. Semua yang berkaitan dengan applet ini berjalan dalam *thread* terpisah, sehingga sementara semua hal-hal tersebut berlangsung, *browser* akan terus melanjutkan pengambilan dan menampilkan elemen halaman HTML yang lain. Pada saat applet selesai diambil dan diinisialisasi, *browser* akan memerintahkan applet untuk menampilkan sendiri dirinya. Sejak saat ini applet bertanggung jawab mengelola daerah tampilannya sendiri.

Agar applet dapat digunakan dalam halaman web, halaman ini harus memuat *tag* applet pada file HTML-nya. *Tag* itu adalah <APPLET></APPLET> seperti berikut:

<APPLET CODE = nama_kelas WIDTH = jml_piksel HEIGHT = jml_piksel>
</APPLET>

2.6 *Double Buffering*

Double buffering adalah suatu metode yang biasanya banyak diterapkan oleh para animator untuk menghilangkan efek kedipan pada animasi yang disebabkan oleh pergantian layar.

2.6.1 Umum

Meskipun animasi sederhana telah berjalan dengan baik, semakin banyak proses menggambar dilakukan pada tampilan jendela dalam *applet*, akan mulai terlihat pergantian layar dalam animasi menyebabkan kedipan.

Hal ini sepertinya memang telah menjadi masalah sejak dahulu oleh para animator atau perancang game karena telah banyak solusi yang ada dalam berbagai macam platform untuk mempercepat proses pembuatan animasi. Baik yang membuat dalam perangkat keras atau pemrograman dalam perangkat lunak, solusi yang paling umum digunakan adalah *double buffering.*

2.6.2 Pengertian *Double Buffering*

Double buffering adalah proses dari penyimpanan salinan layar ke dalam bagian dari memori dan semua proses menggambar dilakukan pada salinan tersebut sebagaimana seperti yang dilakukan pada layar.

Karena semua proses menggambar dilakukan di balik layar, maka proses menggambar yang dilakukan pada layar utama adalah hanya menggambar salinan aktual dari *buffer* ke layar yang dapat diatur waktunya dalam proses pergantian layar.

2.7 *MediaTracker*

MediaTracker merupakan salah satu kelas dari paket Java yang merupakan alat bantu yang amat berguna dalam membuat animasi.

2.7.1 Umum

Dalam proses animasi di web, salah satu kendala utamanya adalah kadang-kadang tidak semua bagian dari image dapat di-load dengan sempurna ketika image tersebut diperlukan dalam proses animasi. Hal ini menyebabkan animasi menjadi patah-patah karena gambar yang belum sempurna.

2.7.2 Penggunaan MediaTracker

Kelas MediaTracker memonitor status daripada tipe media baik itu berupa image, suara dan sebagainya. MediaTracker dapat me-load data secara asynchronous(di background) atau synchronous(menunggu data untuk di-load terlebih dahulu). Proses load image dipercepat oleh thread yang ada padanya dengan suatu grup image yang diidentifikasikan dengan ID tertentu.

2.8 Pemrograman Terdistribusi

Penyimpanan status permainan serta login pemain yang telah terdaftar bisa diterapkan pada suatu sistem terdistribusi, dimana pada sistem ini terdapat aplikasi yang bertindak sebagai *server* dan lainnya sebagai *client*.

Untuk mengimplementasikan hal ini maka dibutuhkan beberapa teknis pemrograman aplikasi terdistribusi.

2.8.1 Umum

Sistem terdistribusi terdiri dari sekumpulan komputer yang terhubung bersama dalam sebuah *network* (jaringan)."(Mahmoud, Qusay H; Distributed Programming with Java; Manning Pubications Co.; 2000). Network tersebut dilengkapi dengan *software* sistem terdistribusi yang memungkinkan komputer-komputer untuk saling mengkoordinasikan aktifitas dan berbagi sumber daya (*hardware*, *software* atau data).

Aplikasi terdistribusi didasarkan pada arsitektur *client/server*, yaitu terdiri dari sekumpulan proses *server* dan sekumpulan proses *client*. Proses *server* bertindak sebagai *resource manager* (pengelola sumber daya) dan proses *client*

masing-masing melakukan suatu tugas yang membutuhkan akses ke beberapa *shared resource* (sumber daya yang dipakai bersama).

2.8.2 *Internetworking*

Terminologi "*network*" (jaringan) biasanya berarti sekumpulan komputer dan *peripheral* (printer, modem, scanner dan sebagainya) yang terhubung bersama oleh beberapa medium. Hubungan tersebut bisa terjadi secara langsung (via kabel) atau tidak langsung (via *modem*). *Device-device* (alat/perlengkapan) yang berbeda pada jaringan saling berkomunikasi melalui aturan-aturan yang telah disepakati bersama, yang umumnya disebut *protocol* (protokol).

Terminologi "*protocol*" digunakan untuk mereferensikan sekumpulan aturan dimana dua atau lebih komputer harus mengikutinya untuk saling bertukar *message* (pesan). Sebuah protokol mendeskripsikan baik itu format *message* yang akan dikirim ataupun apa saja tanggapan yang seharusnya dilakukan komputer terhadap tiap *message* yang diterimanya. Protokol yang biasa digunakan di internet disebut *Internet Protocol*, disingkat IP. *Message-message* berjalan di internet dalam bentuk *packet-packet* (paket-paket), biasa disebut *internet packet* atau *datagram*. Protokol yang paling umum digunakan untuk mentransmisikan paket data adalah protokol *TCP* dan *UDP*.

Protokol komunikasi yang digunakan dalam aplikasi pada Tugas Akhir ini adalah protokol *Transmission Control Protocol* (*TCP*).

2.8.3 Pemrograman Aplikasi Terdistribusi

Seperti telah disebutkan pada bagian 2.5.1, sistem terdistribusi terdiri dari sekumpulan komputer yang terhubung oleh satu jaringan dan dilengkapi dengan *software* sistem terdistribusi, sehingga memungkinkan komputer-komputer tersebut mengkoordinasi aktifitasnya dan saling berbagi sumber daya. Sumber daya dalam sistem terdistribusi secara fisik terbungkus dalam satu komputer; komputer lain hanya bisa mengaksesnya melalui mekanisme komunikasi tertentu. Sumber daya dikelola oleh *resource manager*, yang merupakan komponen penting dalam sistem ini. Karena itu, dalam sistem terdistribusi, pengguna sumber daya berkomunikasi dengan *resource manager* jika ingin mengakses sumber daya bersama.

Web merupakan contoh dari sistem terdistribusi. *Web server-web server* berjalan di atas komputer-komputer yang berlainan dan masing-masing *server* menangani banyak dokumen dan informasi dalam berbagai media. *Web server-web server* tersebut bertindak sebagai *resource manager*.

Pemrograman aplikasi untuk sistem terdistribusi memiliki kelebihan sebagai berikut:

- Aplikasi *multiuser*: Komputasi terdistribusi membuat aplikasi-aplikasi dari sistem yang berdiri sendiri memungkinkan untuk saling terhubung.
- Berbagi sumber daya: Komputasi terdistribusi memungkinkan organisasi untuk mengatur penggunaan sumber daya fisik yang dimilikinya. Misalnya adalah penggunaan bersama sebuah *printer* atau *scanner* oleh sekumpulan karyawan di kantor.

- Skalabilitas: Sistem terdistribusi yang dibangun dengan komponen bermacam-macam, memungkinkannya untuk menambah komponen baru sebanyak yang diperlukan untuk menyelesaikan satu masalah.
- Efisiensi: Macam-macam *platform* dalam lingkungan heterogen dapat diutilisasikan sehingga tiap komputer hanya menyelesaikan tugas spesifik tertentu saja dari satu rangkaian masalah kompleks.
- Transparansi: Sistem terdistribusi terasa bagai satu kesatuan daripada sekumpulan komponen independen. Kerenanya, obyek lokal maupun *remote* dapat diakses menggunakan cara yang sama, tanpa pengguna perlu tahu di mana sebenarnya lokasi fisik mereka berada.

Selain kelebihan-kelebihan di atas, terdapat juga kesulitan dari pemrograman terdistribusi:

- Macam-macam kemungkinan kegagalan: Aplikasi terdistribusi mungkin saja gagal karena mereka dibangun dengan komponen yang lebih banyak dari aplikasi *stand alone* (berdiri sendiri). Misalnya aplikasi terdistribusi yang terdiri dari komponen A dan B, yang berjalan pada komputer berbeda, mungkin saja kegagalan terjadi pada komponen A, komponen B atau *network* yang menghubungkan keduanya.
- Penggunaan bermacam-macam teknologi: Aplikasi terdistribusi mungkin memerlukan penggunaan teknologi yang bermacam-macam. Teknologi-teknologi ini berasal dari berbagai *vendor*, sehingga bisa saja terjadi mereka tidak dapat saling bekerja sama dengan baik.

Testing (pengujian) dan *debugging* (pencarian kesalahan): *Testing* dan *debugging* aplikasi terdistribusi lebih sukar daripada aplikasi *stand alone*, karena

aplikasi terdistribusi seharusnya dapat dijalankan pada lebih dari satu mesin (komputer).

2.8.4 Pemrograman Terdistribusi pada Java

Perangkat lunak yang akan dibuat dalam tugas akhir ini merupakan aplikasi terdistribusi dan ditulis menggunakan bahasa pemrograman Java. Java digunakan karena sifat bahasa ini yang *platform independent* (tidak tergantung *platform* tertentu untuk menjalankannya), sehingga aplikasi yang dihasilkan dapat dijalankan pada sembarang sistem asalkan memiliki *interpreter* Java di dalamnya. Sifat ini penting karena aplikasi terdistribusi (yang pada dasarnya berbasis *network*) seharusnya bisa dijalankan pada *platform* internet yang berbeda-beda.

Sebagai bahasa pemrograman, Java menyediakan beberapa *feature* untuk pembuatan aplikasi terdistribusi, diantaranya:

- *Socket*; Java mendukung protokol *connection oriented* (*TCP*) dan *connectionless* (*UDP*), dimana *socket* berkomunikasi di atasnya.
- Java RMI (*Remote Method Invocation*); merupakan *feature* Java yang mengijinkan *method* dari suatu obyek Java untuk dijalankan secara *remote* (jarak jauh) oleh Java Virtual Machine lain pada komputer berbeda melalui jaringan. Dalam model ini, komunikasi antar obyek bisa terjalin jika semua obyek yang terlibat dibuat dengan bahasa Java.
- Java IDL (*Interface Definition Language*); mengijinkan sebuah program Java *client* untuk menjalankan sebuah obyek IDL yang tersedia pada *remote server* secara transparan. IDL juga mengijinkan program Java *server* untuk mendefinisikan obyek yang dapat dijalankan secara transparan dari *client* IDL.

Beda dari RMI adalah, JavaIDL dapat berkomunikasi dengan obyek lain walaupun obyek lain itu tidak ditulis dalam bahasa Java.

Karena aplikasi dalam Tugas Akhir ini mengimplementasikan *feature socket*, maka penjelasan lebih detil nantinya akan difokuskan hanya pada *feature* tersebut.

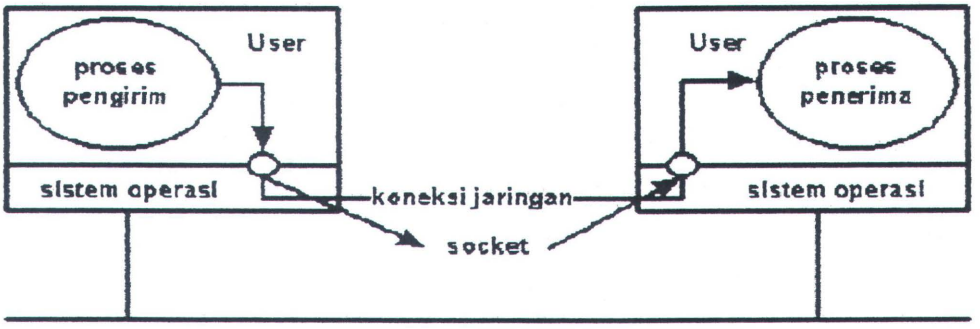
2.9 Pemrograman *Socket*

Socket adalah level paling bawah dari semua *feature* yang disediakan Java untuk mendukung pemrograman terdistribusi. Karena itu pada pemrograman *socket*, perlu didefinisikan secara eksplisit tipe protokol yang dipakai beserta nomor *port*-nya.

2.9.1 Umum

Socket merupakan sebuah kanal komunikasi; dapat dianalogikan dengan telepon dimana telepon menyediakan pelanggannya sebuah antarmuka ke sistem telepon. Demikian juga dengan *socket* yang menyediakan *user* sebuah antarmuka ke *network*. Siklus hidup *socket* adalah: *membuat* (membuka *socket*), *membaca dan menulis* (menerima dan mengirim ke *socket*) dan *menghapus* (menutup *socket*).

Gambar 2.1 mengilustrasikan bagaimana *socket* menyediakan antarmuka di antara *user* dan *network*.



Gambar 2.1 ***Socket*; antarmuka antara *user* dan *network***

Socket digunakan untuk saling bertukar *message* (pesan/data). Ketika suatu *message* dikirim, *message* tersebut akan diantrikan pada *socket* pengirim hingga protokol jaringan mengirimkannya. Ketika *message* tiba, mereka akan diantrikan pada *socket* penerima hingga proses penerima melakukan sesuatu untuk memproses *message-message* itu. Tujuan dari *message* (*message destination*) dispesifikasikan dalam alamat-alamat *socket* (*socket addresses*). Tiap alamat *socket* merupakan pengidentifikasi komunikasi yang terdiri dari alamat internet berikut nomor *port*-nya.

Dua atau lebih *socket* harus dihubungkan sebelum mereka dapat dipakai untuk mentransfer data. Ada sejumlah tipe koneksi yang dapat dipilih, tetapi kebanyakan perangkat lunak yang mengimplementasikan komunikasi *socket* antar komputer menggunakan *TCP* dan *UDP*.

2.9.2 Komunikasi *TCP/IP* dan *UDP/IP*

Terdapat dua protokol komunikasi untuk pemrograman *socket* yakni komunikasi *datagram* dan komunikasi *stream*.

Protokol komunikasi datagram, dikenal sebagai *UDP*, adalah sebuah *connectionless protocol*, artinya bahwa setiap kali mengirim datagram, diharuskan

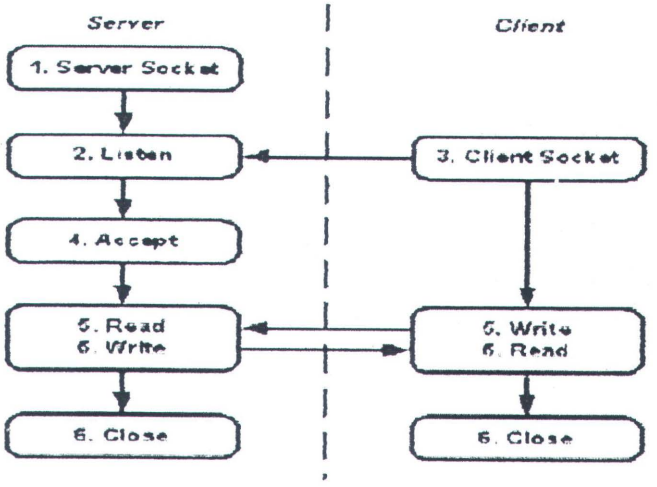
pula untuk mengirim deskripsi *socket* lokal dan alamat *socket* penerima. Jadi bisa dikatakan, tambahan data harus dikirimkan setiap kali komunikasi dibuat.

Protokol komunikasi stream dikenal sebagai *TCP*. Tidak seperti *UDP*, *TCP* adalah sebuah *connection-oriented protocol.* Untuk berkomunikasi melalui protokol *TCP*, sebuah koneksi harus terlebih dahulu diciptakan antara *socket.* Ketika salah satu *socket* mendengar untuk permintaan koneksi (server), *socket* yang lain meminta untuk sebuah koneksi (klien). Setelah kedua *socket* telah terhubung, mereka dapat digunakan untuk mengirim data dalam kedua arah.

2.9.3 Komunikasi *Client/Server*

Dalam pemrograman aplikasi terdistibusi, digunakan model *client/server*, dimana satu proram (*client*) berkomunikasi dengan program lain (*server*) untuk tujuan pertukaran informasi.

Gambar 2.2 menunjukkan serangkaian langkah yang umumnya diambil untuk menyusun komunikasi *socket* dan bertukar data antara satu *client* dan satu *server.*



Gambar 2.2 Komunikasi *socket* antar *client/server*

Supaya komunikasi dapat terjalin, *client* dan *server* harus berbicara dalam bahasa yang sama, atau *protocol*, dimana merupakan sekumpulan perintah yang mengijinkan *client* dan *server* bertukar informasi.

2.9.4 Pemrograman *Socket* pada Java

Inti dukungan Java terhadap pemrograman jaringan adalah kelas *Socket* dan *DatagramSocket*, dimana merupakan bagian dari paket *java.net*. Karena aplikasi dalam Tugas Ahir ini berbasis komunikasi *stream*, maka yang akan dibahas pada sub-bab ini hanya *feature-feature* Java yang berhubungan dengan model komunikasi tersebut.

2.9.4.1 Data *stream*

Inti dari semua input/output di Java adalah *data stream*. Data *stream* dapat diibaratkan seperti pipa saluran data dimana informasi dapat diletakkan (menulis ke *stream*) atau diambil (membaca dari *stream*) dari pipa tersebut.

Dengan *socket*, informasi dibaca dan ditulis dengan membungkusnya dalam beberapa cara. Informasi dinulis dengan membentuk sebuah *OutputStream* (yang menyediakan *method* write()), atau sebuah *InputStream* (yang menyediakan *method* read()). Ketika hubungan *socket* telah berhasil terbentuk, setiap *end point* (titik akhir) membuat sebuah InputStream dan OutputStream. InputStream dibuat menggunakan *method* getInputStream dan OutputStream dibuat memakai getOutputStream(). Stream-*stream* lainnya yang termasuk dalam paket *java.io* adalah:

- *BufferedReader* dan *BufferedWriter* menyangga (*buffering*) data dalam operasi pembacaan dan penulisan, sehingga mengurangi jumlah akses yang dibutuhkan ke data sebenarnya.
- *FilterInputStream* dan *FilterOutputStream* dibangun menggunakan *instance* dari InputStream dan OutputStream yang tak ter-*buffer*. Karenanya, untuk peningkatan performa, teknik yang dipakai adalah *caching* dan *flushing*.
- *DataInputStream* dan *DataOutputStream* menyediakan layanan yang lebih tinggi terhadap pembacaan dan penulisan tipe data primitif.
- *FileReader*, *FileWriter*, *FileInputStream* dan *FileOutputStream* digunakan secara kolektif untuk membaca dari atau menulis ke sebuah *file*.
- *ObjectInputStream* dan *ObjectOutputStream* digunakan untuk membaca dan menulis obyek yang telah diserialisasi.

2.9.4.2 *Socket TCP*

Koneksi *socket TCP* dibuat melalui kelas *Socket*. Ada empat langkah untuk memprogram *client* atau *server* memakai *socket*:

1. Membuka *socket*
2. Membuat data input *stream*
3. Membuat data output *stream*
4. Menutup *socket*

2.10 Unified Modeling Language

UML (Unified Modeling Language) adalah bahasa untuk menspesifikasikan, memvisualisasikan dan membangun produk sistem perangkat lmenggunakan pendekatan yang berorientasi obyek."(Quatrani, Terry; Visual Modeling With Rational Rose and UML; Copyright © 1998 by Addison Wesley Longman, Inc.).

UML merupakan standar industri untuk model yang berorientasi obyek yang dibangun dari tiga metode pendahulunya, yaitu metode Booch (oleh Grady Booch), metode OMT (Object Modeling Technique oleh James Rumbaugh) dan metode OOSE (Object Oriented Software Engineering oleh Ivar Jacobson). Sebuah badan standardisasi industri, OMG (Object Management Group) kemudian menetapkan standar UML v1.0 pada November 1997

UML memiliki sembilan jenis diagram sistem, yaitu : *Activity Diagram*, *Class Diagram*, *Collaboration Diagram*, *Component Diagram*, *Deployment Diagram*, *Object Diagram*, *Use Case Diagram*, *Sequence Diagram* dan *State Diagram*. Kesembilan diagram tersebut dapat dikelompokkan ke dalam tiga kategori, yaitu statis, dinamis dan arsitektural.

Sebuah diagram statis (*Static Diagram*) menggambarkan struktur dari suatu sistem dan fungsi serta tanggung jawabnya. Yang termasuk ke dalam diagram ini adalah diagram *Class* dan *Object*. Diagram dinamis (*Dynamic Diagram*) menggambarkan interaksi atau hubungan-hubungan yang didukung oleh sistem tersebut. Yang termasuk di dalam kategori ini diantaranya diagram *Activity*, *Collaboration*, *Sequence*, *State* dan *Use Case*. Diagram arsitektural (*Architectural Diagram*) menggambarkan realisasi dari sistem tersebut menjadi komponen-

komponen yang dapat dieksekusi. Yang termasuk ke dalamnya adalah diagram *Component* dan *Deployment*.

Pada penerapannya nanti, setiap proyek yang ingin digambarkan, paling tidak akan memiliki minimum tiga diagram, yaitu *Use Case*, *Class* dan *Sequence*.

BAB III

PERANCANGAN SISTEM

BAB III

PERANCANGAN SISTEM

Pada Bab 3 akan dibahas perancangan sistem yang akan digunakan dalam Tugas Akhir ini sehubungan dengan implementasi *Role Playing Game*. Notasi yang dipakai untuk memodelkan sistem pada bab ini adalah notasi *Unified Modeling Language* (UML) dengan menggunakan *tool* Rational Rose 2001 Enterprise Edition.

Perancangan sistem yang dilakukan meliputi perancangan game dan perancangan server game. Untuk perancangan game dibagi menjadi empat bagian besar yakni perancangan obyek, struktur pembangun game, sistem, dan *tool*. Obyek-obyek tersebut digunakan dalam game dan saling berinteraksi dengan antar bagian di dalamnya. Sistem-sistem yang ada dalam game merupakan aktivitas atau transaksi yang dilakukan dalam game. Tool merupakan sarana atau alat bantu yang digunakan dalam game. Server game berfungsi untuk melayani login serta penyimpanan status game. Penjelasan tentang perancangan game dan server game secara detilnya akan dijelaskan di bawah.

Dalam implementasi desain ke perangkat lunak akan menggunakan pendekatan *round-trip software development*, yaitu desain yang telah selesai, ada kemungkinan mengalami beberapa perubahan bila telah memasuki tahap implementasi.

3.1 Metodologi Perancangan

Dalam perancangan sistem ini, dilakukan langkah-langkah sebagai berikut:

1. Skenario sistem dan analisa kebutuhan sistem

   Pada tahap ini diberikan deskripsi dari sistem berikut kebutuhan dan aturan yang ada di dalamnya.

2. Analisa *use case* dan proses

   Dalam tahap ini dilakukan analisa terhadap proses yang terlibat dalam sistem game beserta aktor (pelaku) yang terlibat. Dari analisa, akan dihasilkan *use case diagram* yang menggambarkan proses-proses yang ada dan para aktornya.

   Dalam setiap *use case diagram*, terdapat beberapa *sequence diagram.* Diagram-diagram ini berisi rancangan urutan proses untuk setiap *use case* beserta aliran interaksi antar obyek (*class*).

3. Perancangan game

   Dalam tahap ini digambarkan obyek apa saja yang akan terlibat untuk membangun aplikasi. Obyek-obyek yang ada kemudian dikelompokkan menjadi empat bagian yaitu obyek game, struktur pembangun game, sistem dan *tool* game.

4. Perancangan server game

   Dalam tahap ini digambarkan fungsi apa saja yang akan dilakukan oleh server dalam hubungannya dengan game..

3.2 Skenario dan Kebutuhan Sistem

Pada bagian ini dibahas sistem yang digunakan dalam game. Dari game ini diidentifikasi beberapa kebutuhan yang harus ada pada sistem yang akan dibuat.

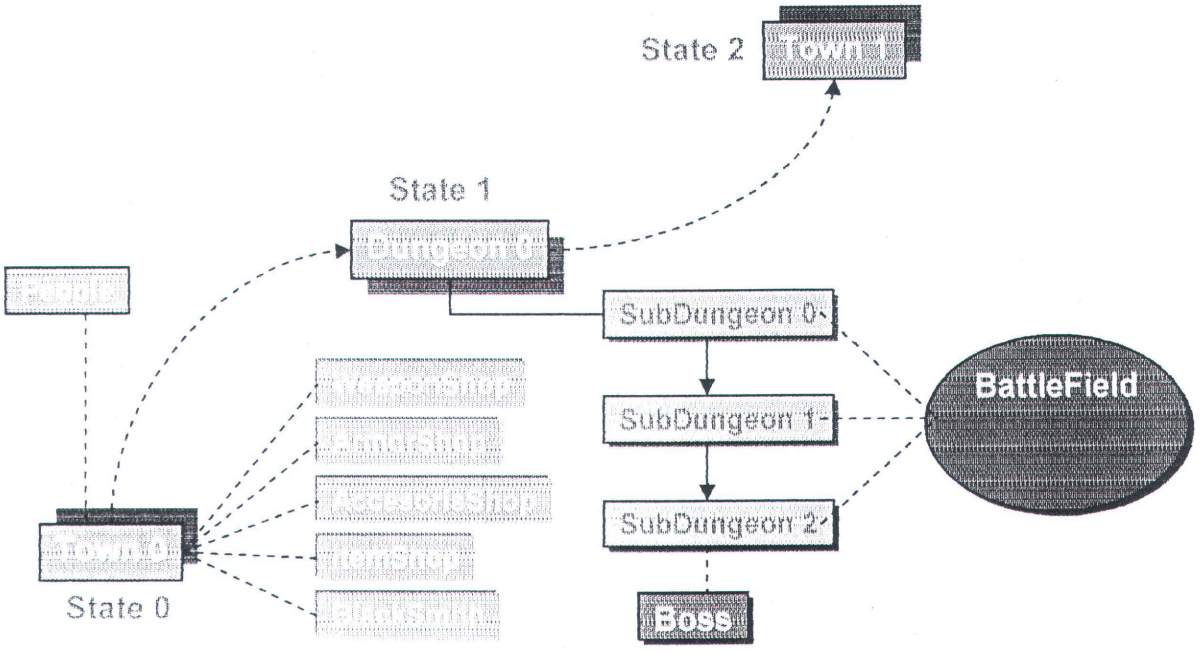
3.2.1 Skenario Sistem

Ketika pemain memulai permainan baru, pemain akan disuguhi suatu cerita pembuka dan diberi tim *hero* awal yang nantinya akan terus digunakan oleh pemain dan bisa berubah setiap saat di tengah permainan. *Hero* adalah tokoh yang menjadi pelaku dan bertualang dalam game.

Kemudian pemain akan disuguhkan suatu peta dunia yang di dalamnya telah tersusun *state-state* yang membentuk suatu alur perjalanan yang akan dilalui oleh *hero*. Susunan state-state tersebut diperoleh melalui proses inisialisasi dari obyek pengaturan state yang telah ditentukan oleh administrator pada server game. *State* adalah tempat-tempat yang akan disingggahi oleh *hero*. *State-state* tersebut disusun secara saling berurutan dan saling terkait antara satu dengan yang lain, jadi suatu *state* baru bisa muncul dan dapat disinggahi jika *hero* telah selesai melakukan suatu misi tertentu di *state* sebelumnya.

State-state tersebut diinisialisasikan sebagai *town* atau *dungeon*. *Town* adalah tempat *hero* beristirahat dan melakukan transaksi. *Dungeon* adalah tempat *hero* bertualang dan bertarung melawan *monster*. *Monster* adalah musuh dalam game yang harus dilawan oleh *hero*. Pada setiap *state*, pemain bisa menyimpan status permainannya yang nantinya bisa di-load dari *state* tersebut.

Permainan berakhir jika *hero* telah menyelesaikan misi pada *state* paling akhir dari game tersebut.



Gambar 3.1 Gambaran Sistem Game

3.2.2 Kebutuhan Sistem

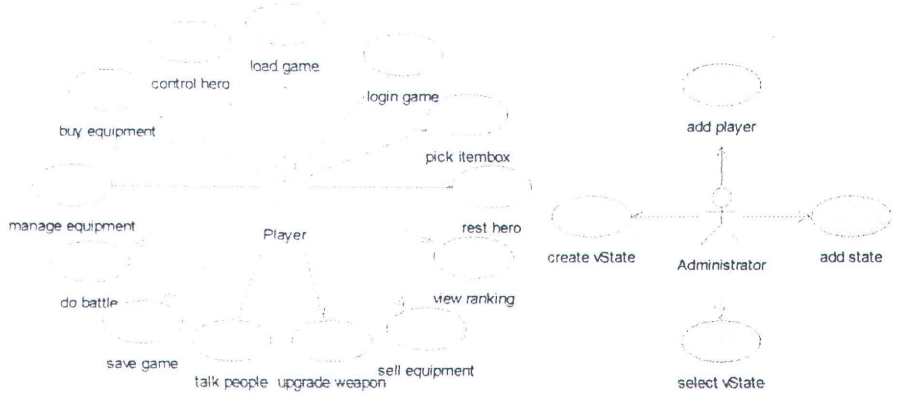
Sistem game yang akan dirancang mempunyai spesifikasi kebutuhan sistem sebagai berikut:

- Sistem menyediakan suatu area dalam permainan bisa dijelajahi oleh pemain melalui hero.
- Sistem memiliki proses yang melibatkan pengaturan serta transaksi jual-beli perlengkapan hero.
- Sistem memiliki suatu proses arena pertempuran antara hero melawan monster dalam permainan.
- Sistem menyediakan fasilitas untuk menciptakan dan memilih obyek dunia permainan pada server game yang dilakukan oleh administrator. Obyek tersebut berisi rancangan pengaturan dalam game serta segala aspek yang ada di dalamnya.

- Sistem dapat menangani proses penyimpanan dan loading status permainan antara server dan klien game.
- Sistem dapat menangani keamanan aplikasi game melalui proses login bagi pemain.

3.3 Perancangan *Use-Case*

Dalam sistem *Role Playing Game* ini terdapat 14 use case dan dua aktor. Use case-use case tersebut yakni "*login game*", "*control hero*", "*save game*", "*load game*", "*talk people*", "*manage equipment*", "*buy equipment*", "*sell equipment*", "*do battle*", "*upgrade weapon*", "*pick itembox*", "*view ranking*", "*rest hero*", "*add player*", "*add state*", dan "*reset state*". Sedangkan sebagai aktor yakni pemain. Gambar 2.5 menggambarkan use case tersebut.



Gambar 3.2 Use Case Diagram : Use Case View / Main

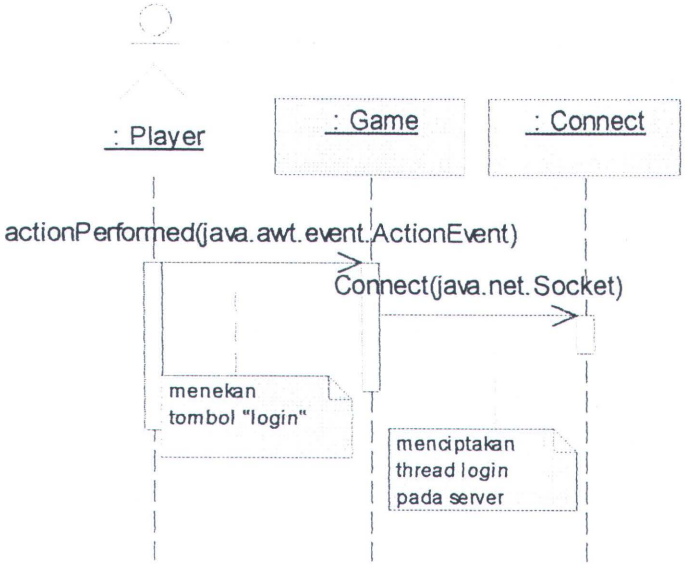
3.3.1 Deskripsi Aktor

Player(pemain) adalah orang yang memainkan *Role Playing Game* ini. Jadi nasib *hero* dalam game ini ditentukan oleh player.

Administrator adalah orang yang mengatur dunia permainan serta pemainnya.

3.3.2 *Use Case – Login Game*

Use case ini menggambarkan rangkaian proses yang dilakukan oleh pemain untuk login dalam game. Ketika memulai game, pemain akan disuguhkan antar muka login game. Dari sini pemain memasukkan login serta password. Gambar 3.3 di bawah ini menggambarkan rangkaian proses di atas.



Gambar 3.3 Sequence Diagram : Login Game

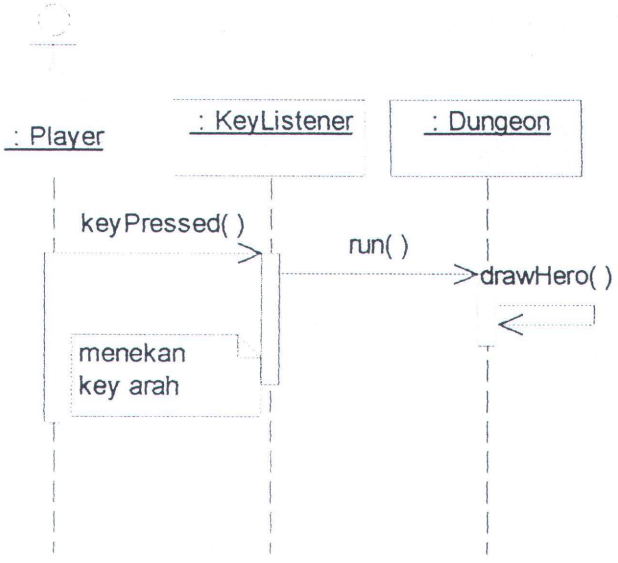
Rangkaian proses yang terjadi dalam sequence diagram ini adalah :

1. Pemain menekan tombol *"login"*, kemudian game akan mengirim login dan password tersebut ke server game melalui *socket* port 9999.
2. Setelah *socket* tersebut diterima oleh server game, diciptakanlah *thread* untuk menangani proses tersebut. Lalu server game membandingkan login dan password tadi dengan basis data yang ada pada server. Hasil dari pembandingan tersebut lalu dikirim kembali ke game lewat *socket* tadi.

3. Jika game menerima konfirmasi ditemukannya login tersebut oleh server game pada basis data, maka pemain bisa memulai permainan dengan permainan baru atau melanjutkan permainan terdahulu.

3.3.3 *Use Case – Control Hero*

Use case ini menggambarkan rangkaian proses yang dilakukan pemain dalam mengontrol pergerakan *hero* dalam game. *Hero* bisa dikontrol arah pergerakannya oleh pemain di dalam *town* atau *dungeon*. Pemain mengontrol *hero* menggunakan key arah pada keyboard. Gambar 3.4 dibawah ini menggambarkan rangkaian proses di atas.



Gambar 3.4 Sequence Diagram : Control *Hero*

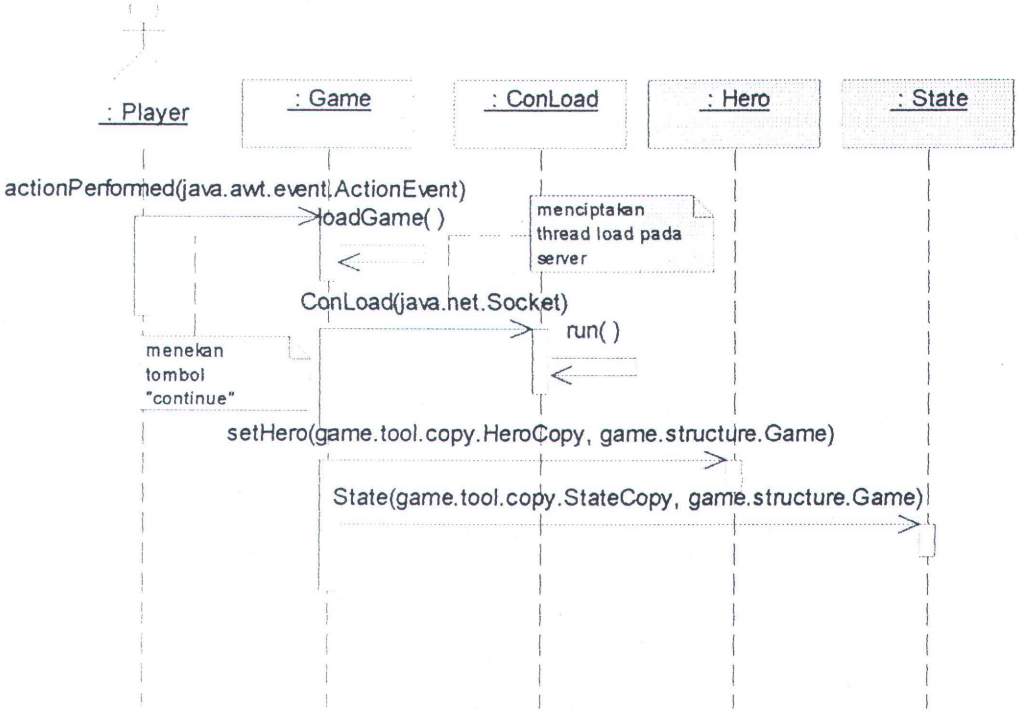
Rangkaian proses yang terjadi dalam sequence diagram ini adalah :

1. Ketika pemain menekan tombol key arah pada keyboard, maka *thread* gambar pada *town* atau *dungeon* dibangkitkan.
2. Dalam *thread* gambar tersebut dijalankanlah metode *drawHero()* yang akan mengubah setiap posisi koordinat *hero* pada *town* atau *dungeon*.

3. Karena sudut pandang game dari atas *hero* maka setiap perubahan koordinat *hero* mengakibatkan pergeseran koordinat *town/dungeon* serta obyek-obyek *town/dungeon* yang lain.
4. Kemudian semua obyek tersebut digambar kembali pada *town/dungeon* berdasarkan koordinat yang telah berubah tadi.

3.3.4 *Use Case – Load Game*

Use case ini menggambarkan rangkaian proses yang dilakukan pemain untuk me-load permainan. Setelah melakukan proses login, pemain akan diberi tombol pilihan untuk memulai permainan baru atau melanjutkan permainan terdahulu. Jika pemain memilih untuk melanjutkan maka proses load dilakukan. Gambar 3.5 di bawah ini menggambarkan rangkaian proses di atas.



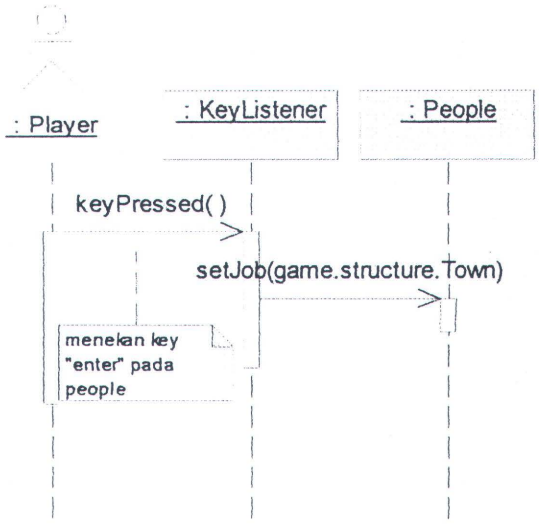
Gambar 3.5 Sequence Diagram : Load Game

Rangkaian proses yang terjadi dalam sequence diagram ini adalah :

1. Ketika pemain menekan tombol *"continue"*, metode *loadGame()* dijalankan.
2. Dalam metode *loadGame()*, game meminta server game untuk mengirimkan obyek *SavePack* berdasarkan login pemain. *SavePack* adalah suatu kelas yang berisi paket penyimpanan segala informasi status dalam game. Penjelasan lebih rinci tentang *SavePack* akan dijelaskan di bawah.
3. Setelah obyek *SavePack* diterima oleh game, kemudian obyek-obyek yang digunakan dalam game seperti *hero*, *state*, serta segala perbekalan *hero* diinisialisasikan berdasarkan salinan dari *SavePack*.
4. Kemudian dari *state* sanalah pemain kemudian melanjutkan permainannya.

3.3.5 *Use Case – Talk People*

Ketika *hero* yang dikontrol oleh pemain memasuki *town*, di sana akan dijumpai penduduk kota yang disebut sebagai *people*. Use case ini menggambarkan rangkaian proses yang dilakukan oleh pemain agar *hero* bisa melakukan percakapan dengan mereka. Peran *people* di *town* bisa sebagai apa saja sesuai dengan pekerjaannya seperti penduduk biasa, penjual perlengkapan, pemilik penginapan, penyimpan status permainan, dan sebagainya. Gambar 3.6 di bawah ini menjelaskan proses di atas.



Gambar 3.6 Sequence Diagram : Talk *People*

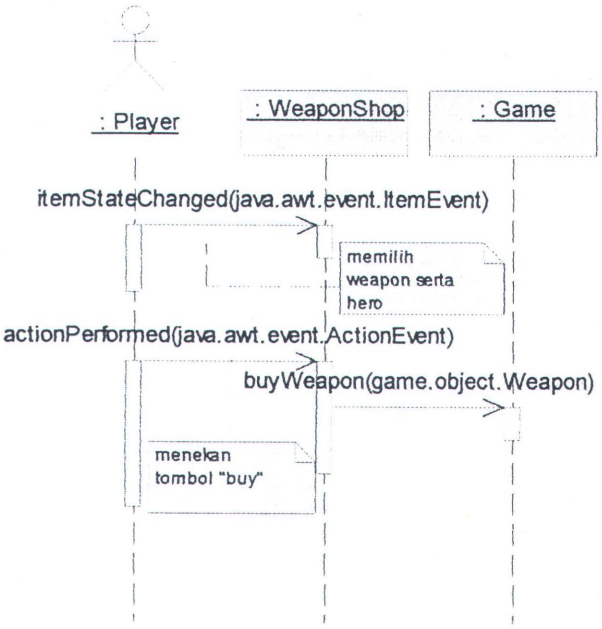
Rangkaian proses yang terjadi dalam sequence diagram ini adalah :

1. Ketika posisi *hero* saling berhadapan dengan *people* dan pemain menekan key *"enter"* pada keyboard maka akan membangkitkan reaksi pekerjaan dari *people* melalui metode *setJob()*.
2. Dalam metode *setJob()* akan ditampilkan bermacam-macam reaksi dari *people* tergantung dari pekerjaannya. Jika sebagai penduduk biasa menampilkan dialog teks, jika sebagai penjual menampilkan form belanja, dan sebagainya.

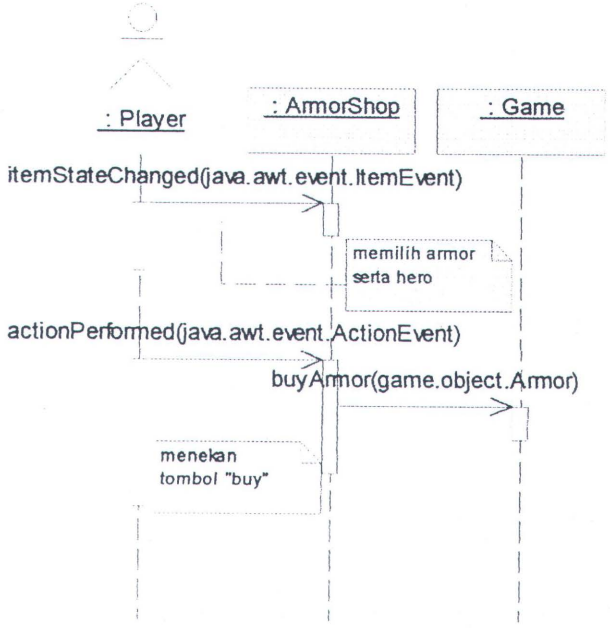
3.3.6 *Use Case – Buy Equipment*

Use case ini menggambarkan rangkaian proses yang dilakukan oleh pemain untuk membeli *equipment*. Jika pemain ingin membeli perlengkapan *hero* maka ia harus mencari *people* yang pekerjaannya sebagai penjual toko. Ada empat macam toko yakni *WeaponShop*(toko senjata), *ArmorShop*(toko baju), *AccesorisShop*(toko aksesoris), dan *ItemShop*(toko item). Ketika form belanja dari toko tersebut telah muncul maka ditampilkan daftar perlengkapan yang ditawarkan oleh penjual yang akan dibeli oleh pemain, daftar perlengkapan yang ingin dijual oleh pemain, dan

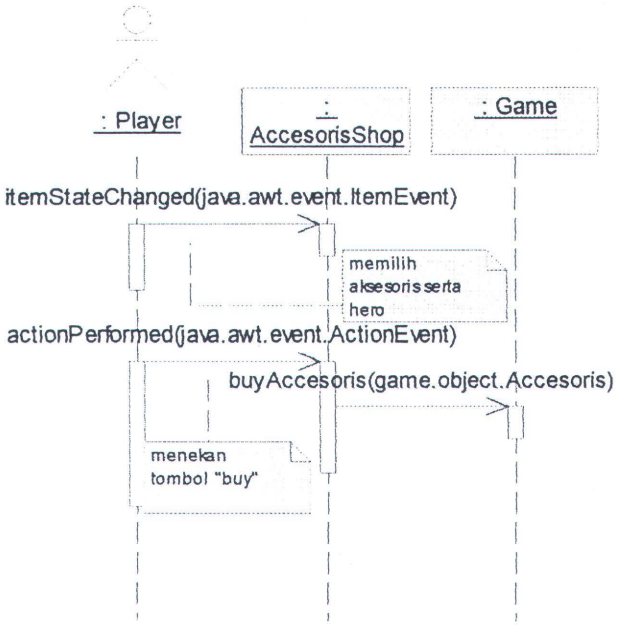
daftar *hero* yang bisa mengenakan perlengkapan tersebut. Gambar-gambar di bawah ini akan menjelaskan satu-persatu proses membeli perlengkapan dari tiap toko.



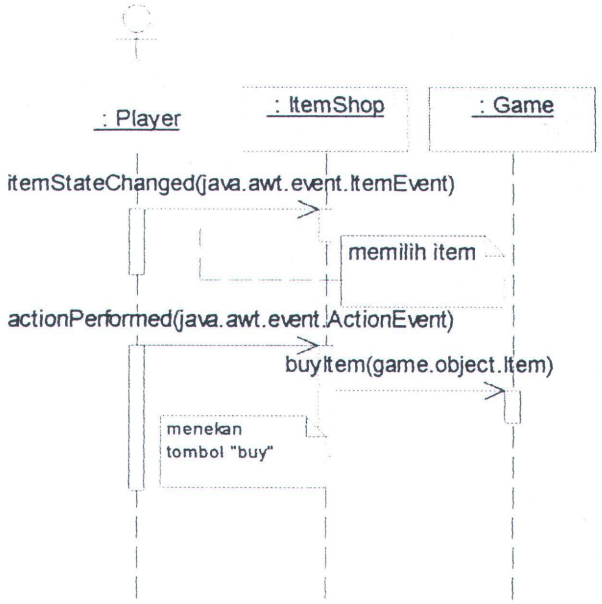
Gambar 3.7 **Sequence Diagram : Buy *Equipment* / Buy *Weapon***



Gambar 3.8 **Sequence Diagram : Buy *Equipment*/Buy *Armor***



Gambar 3.9 Sequence Diagram : Buy *Equipment* / Buy *Accesoris*



Gambar 3.10 Sequence Diagram : Buy *Equipment* / Buy Item

Rangkaian proses yang terjadi dalam sequence diagram di atas adalah :

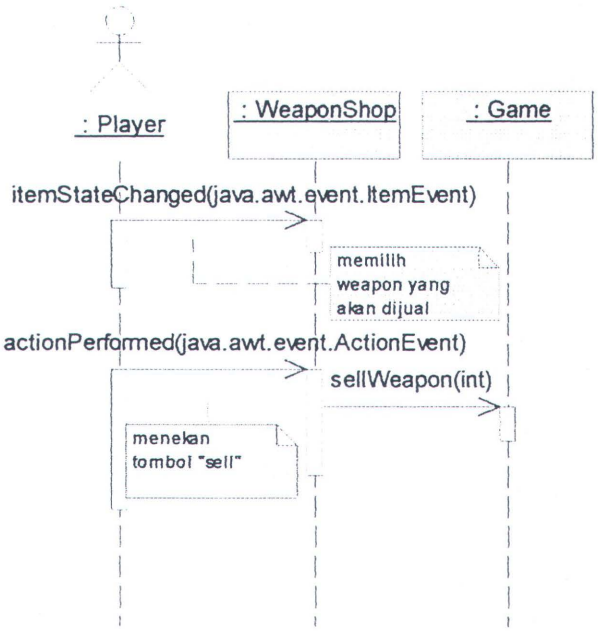
1. Mula-mula pemain memilih *hero* yang akan berbelanja pada daftar hero sehingga isi daftar perlengkapan pada daftar beli akan ditampilkan sesuai dengan tipe *hero* tersebut. Kecuali pada toko item, pemain tidak perlu memilih

hero karena item bisa digunakan oleh semua *hero* dan hanya diperlukan pada saat-saat tertentu saja.

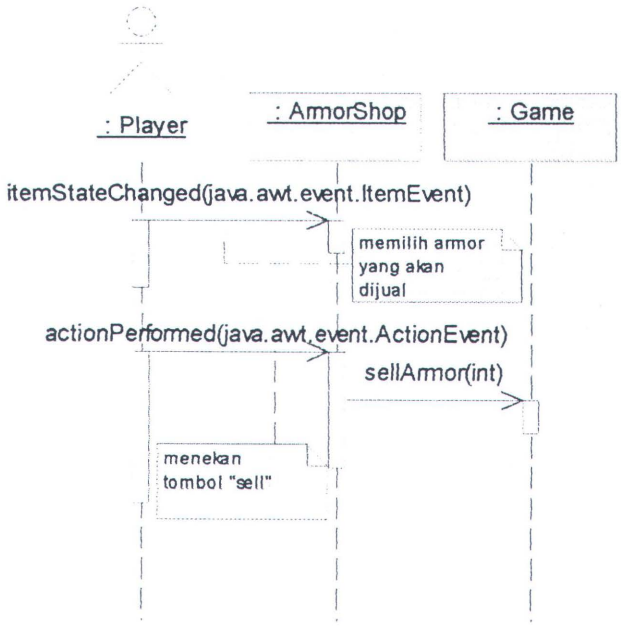
2. Kemudian pemain memilih perlengkapan yang ingin dibeli pada daftar beli. Pemain memilih perlengkapan tersebut berdasarkan efek perubahan status pada *hero* jika mengenakannya.
3. Jika sudah ditentukan, lalu pemain menekan tombol *"buy"* sehingga dijalankanlah metode *buy()* yang akan menambahkan perlengkapan yang telah dibeli tadi pada perbekalan *hero*.

3.3.7 *Use Case – Sell Equipment*

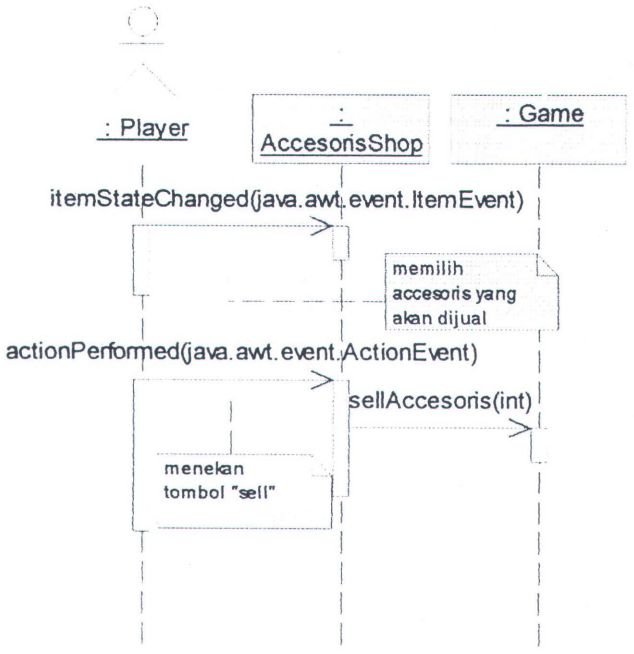
Use case ini menggambarkan rangkaian proses yang dilakukan oleh pemain untuk menjual perlengkapan. Jika pemain kelebihan perlengkapan tidak berguna pada perbekalan *hero* dan sedang membutuhkan uang, ia juga bisa menjualnya pada *people* yang pekerjaannya sebagai penjual toko di *town*. Pada toko tersebut barang yang dijual akan diberi harga setengah dari harga beli. Gambar-gambar di bawah ini akan menjelaskan satu-persatu proses menjual perlengkapan dari tiap toko.



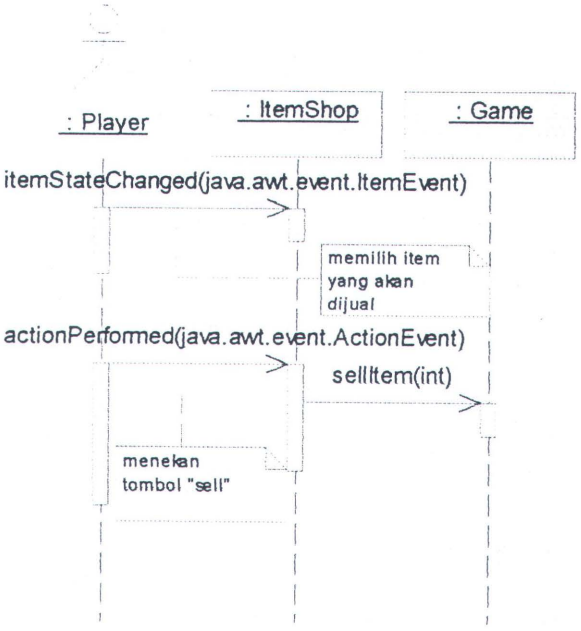
Gambar 3.11 **Sequence Diagram : Sell *Equipment* / Sell *Weapon***



Gambar 3.12 **Sequence Diagram : Sell *Equipment* / Sell *Armor***



Gambar 3.13 Sequence Diagram : Sell *Equipment* / Sell *Accesoris*



Gambar 3.14 Sequence Diagram : Sell *Equipment* / Sell Item

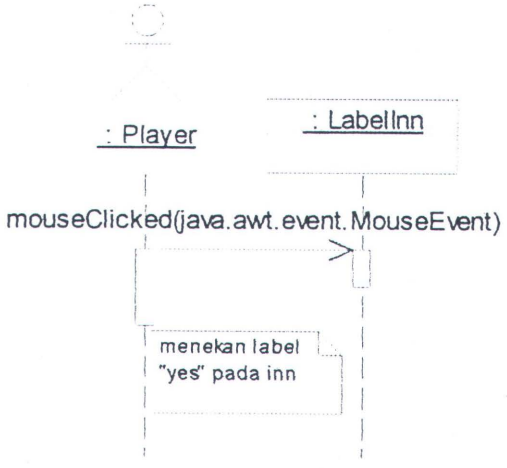
Rangkaian proses yang terjadi dalam sequence diagram di atas adalah :

1. Pemain memilih perlengkapan yang ingin dijual pada daftar jual.

2. Lalu pemain menekan tombol *"sell"* sehingga metode *sell()* dijalankan. Metode ini menyebabkan perlengkapan pada *hero* berkurang sesuai dengan yang dipilih pada daftar jual serta uang akan bertambah sesuai dengan harga jual barang tersebut.

3.3.8 *Use Case – Rest Hero*

Use case ini menggambarkan rangkaian proses yang dilakukan oleh pemain untuk mengistirahatkan *hero*. Pemain bisa mencari *people* yang pekerjaannya sebagai pemilik penginapan di *town* untuk mengistirahatkan *hero*. Istirahat di penginapan berguna untuk mengembalikan kondisi *hero* seperti sedia kala. Tentunya pemain harus membayar ongkos penginapannya juga. Gambar 3.15 di bawah ini menjelaskan proses di atas.



Gambar 3.15 Sequence Diagram : Rest *Hero*

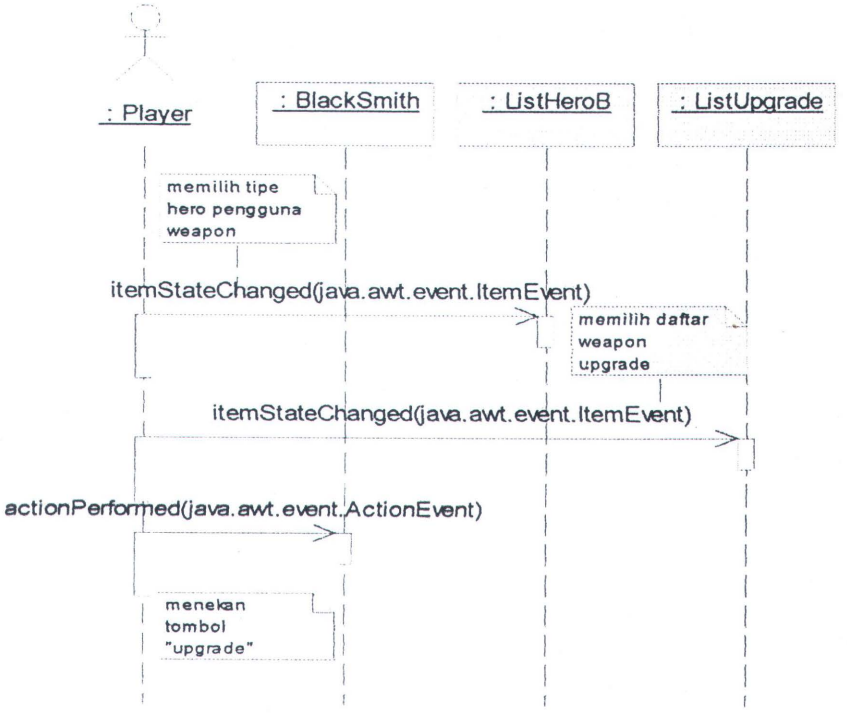
Rangkaian proses yang terjadi dalam sequence diagram di atas adalah :

1. Pada penginapan, pemain diberi tawaran untuk menginap.
2. Jika pemain menekan label *"yes"* pada penginapan tersebut maka proses istirahat *hero* dijalankan.

3. Kondisi *hero* akan kembali seperti semula dan uang berkurang sesuai dengan ongkos penginapan.

3.3.9 *Use Case – Upgrade Weapon*

Pemain bisa meng-upgrade senjata yang dimiliki oleh *hero* menjadi senjata yang lebih kuat. Use case ini menggambarkan rangkaian proses yang dilakukan oleh pemain untuk meng-upgrade senjata *hero*. Pemain bisa melakukannya dengan mencari *people* yang pekerjaannya sebagai *BlackSmith*(pandai besi) di *town*. Senjata hasil upgrade merupakan kombinasi antara senjata dan aksesoris tertentu. Kombinasi yang dibutuhkan dari tiap senjata upgrade berbeda-beda sesuai dengan tipenya. Upgrade senjata pada pandai besi bisa dilakukan jika kombinasi senjata dan aksesoris yang dibutuhkan terpenuhi. Gambar 3.16 di bawah ini menjelaskan proses di atas.



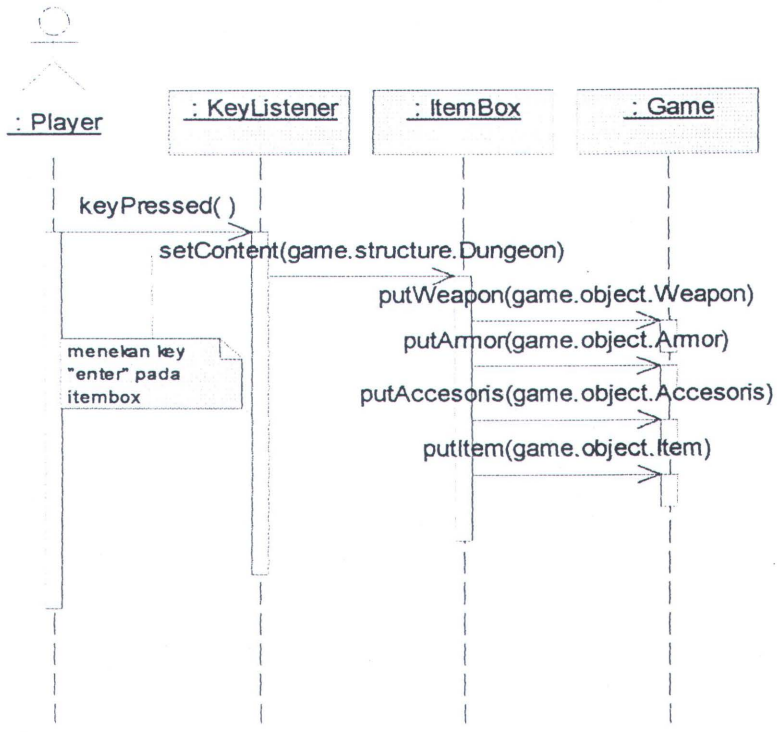
Gambar 3.16 Sequence Diagram : Upgrade *Weapon*

Rangkaian proses yang terjadi dalam sequence diagram di atas adalah :

1. Mula-mula pemain memilih *hero* yang senjatanya ingin di-upgrade pada *ListHero*. Dari pemilihan tersebut akan ditampilkan daftar senjata upgrade beserta kombinasi yang dibutuhkan sesuai dengan *hero* yang telah dipilih pada *ListUpgrade*.
2. Kemudian pemain memilih senjata upgrade yang kebutuhannya telah terpenuhi pada *ListUpgrade*. Tombol *"upgrade"* akan berfungsi jika kombinasi yang dibutuhkan oleh senjata yang dipilih pada *ListUpgrade* terpenuhi.
3. Lalu pemain menekan tombol *"upgrade"* sehingga senjata hasil upgrade diperoleh dan diletakkan pada perbekalan *hero*. Tentu saja proses upgrade tersebut memerlukan biaya sehingga uang berkurang sesuai dengan biayanya.

3.3.10 *Use Case – Pick Itembox*

Jika *hero* berada di *dungeon*, akan ditemui banyak kotak barang di tempat-tempat tersembunyi. Jika pemain bisa menemukan letaknya, *hero* bisa memungut isi dari kotak barang tersebut. Use case ini menggambarkan rangkaian proses yang dilakukan oleh pemain untuk mengambil isi dari kotak barang. Kotak barang tersebut bisa berisi senjata, baju, aksesoris, item, uang, atau *save point*. Gambar 3.17 di bawah ini menjelaskan proses di atas.



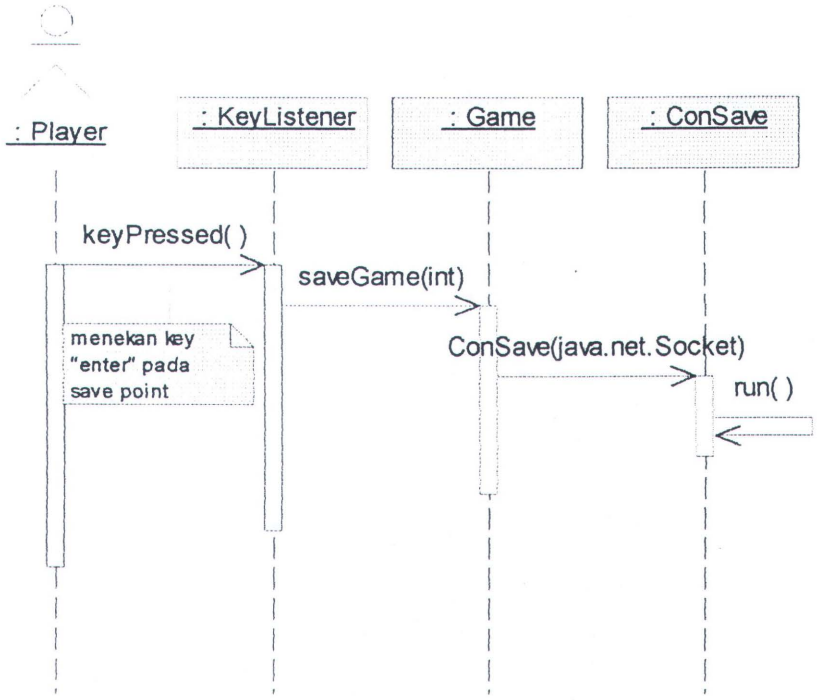
Gambar 3.17 Sequence Diagram : Pick *Itembox*

Rangkaian proses yang terjadi dalam sequence diagram di atas adalah :

1. Jika kotak barang telah berada di hadapan *hero*, pemain menekan key *"enter"* pada keyboard.
2. Kemudian akan dijalankan metode *set Content()* pada kotak barang. Metode ini akan meletakkan isi dari kotak barang ke dalam perbekalan *hero*.

3.3.11 *Use Case – Save Game*

Use case ini menggambarkan rangkaian proses yang dilakukan oleh pemain untuk menyimpan status permainannya. Pemain bisa menyimpan status permainannya pada *save point* yang berada di *town* atau *dungeon*. *Save point* yang berada di *town* adalah *people* yang pekerjaannya sebagai pencatat perjalanan sedangkan yang berada di *dungeon* berupa prasasti. Gambar 3.18 di bawah ini menjelaskan proses di atas.



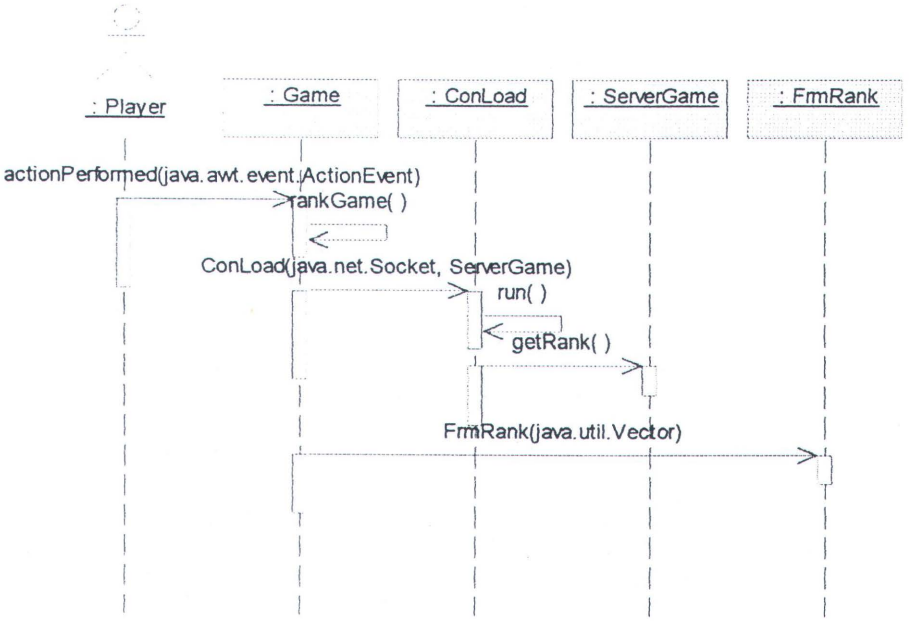
Gambar 3.18 Sequence Diagram : Save Game

Rangkaian proses yang terjadi dalam sequence diagram di atas adalah :

1. Jika *hero* telah berada di depan *save point*, pemain menekan key *"enter"* pada keyboard sehingga metode *saveGame()* dijalankan.
2. Dalam metode *saveGame()* diciptakanlah obyek *SavePack* dengan id *state* tempat *save point* tersebut berada.
3. Lalu obyek *SavePack* tersebut dikirim ke server game melalui *socket* port 10000.
4. Setelah *socket* tersebut diterima oleh server game, lalu diciptakanlah *thread ConSave* yang didalamnya berisi proses untuk menyimpan obyek *SavePack* tadi ke harddisk server game.

3.3.12 *Use Case – View Ranking*

Use case ini menggambarkan rangkaian proses yang dilakukan pemain untuk melihat ranking dari seluruh pemain game ini yang telah terdaftar pada basis data server game. Urutan ranking didasarkan pada status dari tim hero yang dimiliki oleh pemain yang telah disimpan di server game. Gambar 3.19 di bawah ini menjelaskan proses di atas.



Gambar 3.19 Sequence Diagram : View Ranking

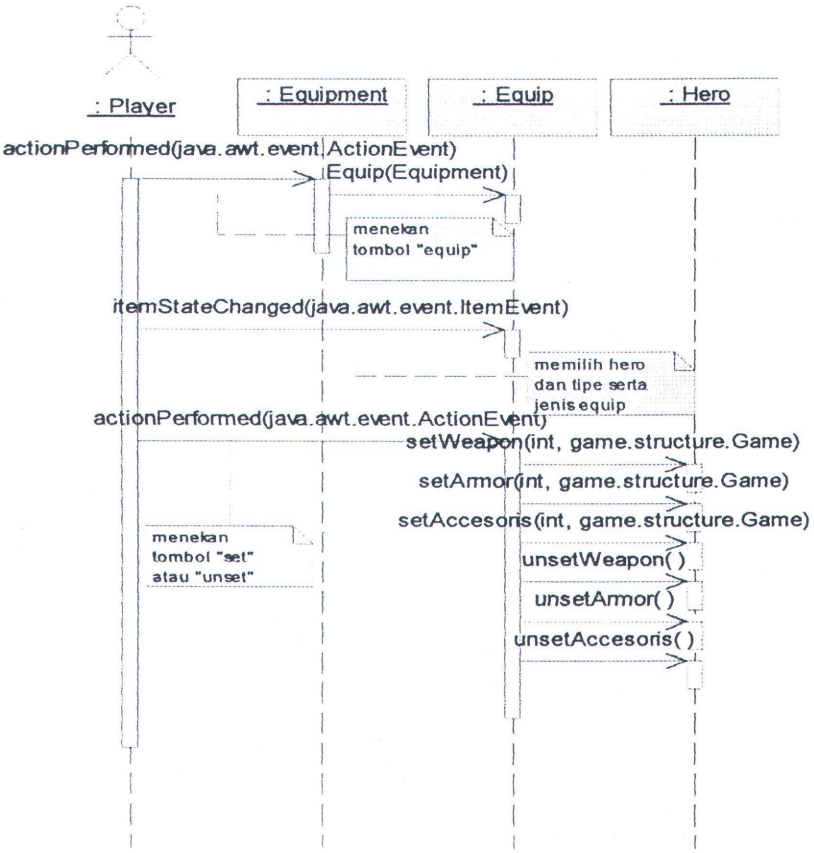
Rangkaian proses yang terjadi dalam sequence diagram di atas adalah :

1. Ketika pemain menekan tombol *"ranking"* maka metode *rankGame()* akan dijalankan.
2. Game kemudian meminta server game untuk mengirimkan vector ranking sehingga terciptalah thread *ConLoad.*
3. Thread tersebut memanggil metode *getRank()* untuk mendapatkan obyek vector yang berisi urutan ranking pemain.

4. Setelah obyek tersebut diterima game, maka ditampilkanlah urutan ranking seluruh pemain pada bingkai ranking.

3.3.13 *Use Case – Manage Equipment*

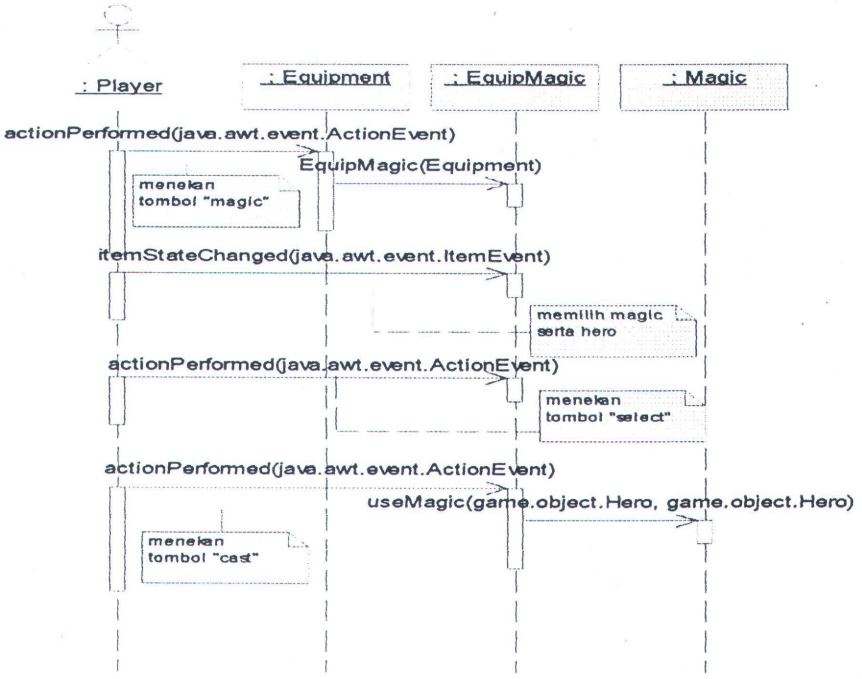
Use case ini menggambarkan rangkaian proses yang dilakukan oleh pemain untuk mengatur perlengkapan *hero*. Ketika *hero* sedang berada di *town* atau *dungeon*, pemain bisa mengatur perlengkapan yang sedang dibawa oleh tim *hero* dengan menekan key *"esc"* pada keyboard. Pengaturan perlengkapan tersebut meliputi memasang/melepas perlengkapan pada *hero*, menggunakan sihir, menggunakan item, dan melihat status masing-masing *hero*. Gambar-gambar di bawah ini akan menjelaskan masing-masing dari tiap proses di atas.



Gambar 3.20 Sequence Diagram : Manage *Equipment* / Set-Unset *Equipment*

Rangkaian proses yang terjadi dalam sequence diagram di atas adalah :

1. Jika pemain menekan tombol *"equip"* pada menu perlengkapan maka akan muncul menu *equip* yang merupakan antar muka untuk penggantian perlengkapan *hero* yang di dalamnya ditampilkan daftar *hero*, label pemilih jenis perlengkapan, serta daftar perlengkapan yang tersedia di perbekalan *hero*.
2. Kemudian pemain memilih *hero* yang akan diganti perlengkapannya pada daftar hero.
3. Lalu pemain memilih jenis perlengkapan yang akan ditampilkan pada daftar perlengkapan. Ada tiga jenis perlengkapan yakni senjata, baju, dan aksesoris.
4. Setelah itu pemain memilih tipe perlengkapan yang diinginkan pada daftar perlengkapan. Pemilihan tersebut berdasarkan pada perubahan status *hero* jika mengenakannya.
5. Dan akhirnya pemain menekan tombol *"unset"* untuk melepas perlengkapan yang sedang dipakai *hero* atau kemudian menekan tombol *"set"* untuk memasang perlengkapan yang telah dipilih tadi pada daftar perlengkapan.

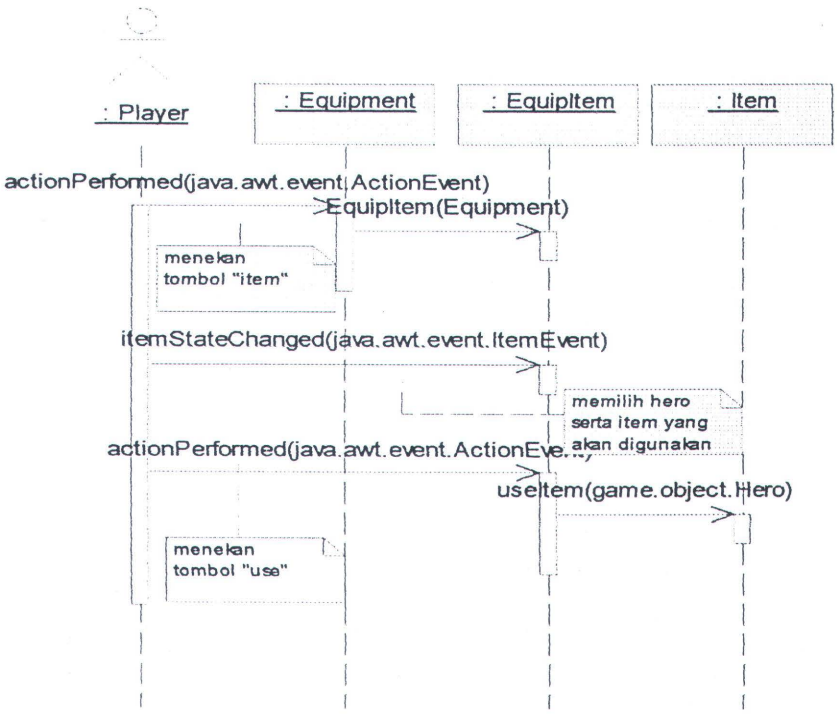


Gambar 3.21 Sequence Diagram : Manage *Equipment* / Cast *Magic*

Rangkaian proses yang terjadi dalam sequence diagram di atas adalah :

1. Jika pemain menekan tombol *"magic"* pada menu perlengkapan maka akan muncul menu *equip magic* yang merupakan antar muka untuk menggunakan sihir yang dimiliki oleh *hero*. Pada menu ini akan ditampilkan daftar *hero* dan daftar sihir yang dimiliki *hero*.
2. Lalu pemain memilih *hero* yang akan menggunakan sihir pada daftar hero sehingga daftar sihir yang ditampilkan hanya sihir yang dimiliki oleh *hero* tersebut.
3. Kemudian pemain memilih sihir yang akan digunakan pada daftar sihir. Di sini hanya sihir penyembuhan yang bisa digunakan, jadi sihir yang bersifat menyerang tidak bisa digunakan karena hanya bisa dalam pertempuran saja.
4. Lalu pemain menekan tombol *"select"* jika telah menentukan sihir yang akan digunakan. Tombol *"select"* bisa ditekan jika mp *hero* mencukupi untuk menggunakan sihir tersebut.

5. Lalu pemain memilih *hero* yang akan menjadi target sihir pada daftar hero.
6. Setelah itu pemain menekan tombol *"cast"* yang akan menjalankan metode *useMagic()* pada sihir. Pada metode ini akan ditentukan efek yang ditimbulkan terhadap targetnya berdasarkan tipe sihir tersebut.

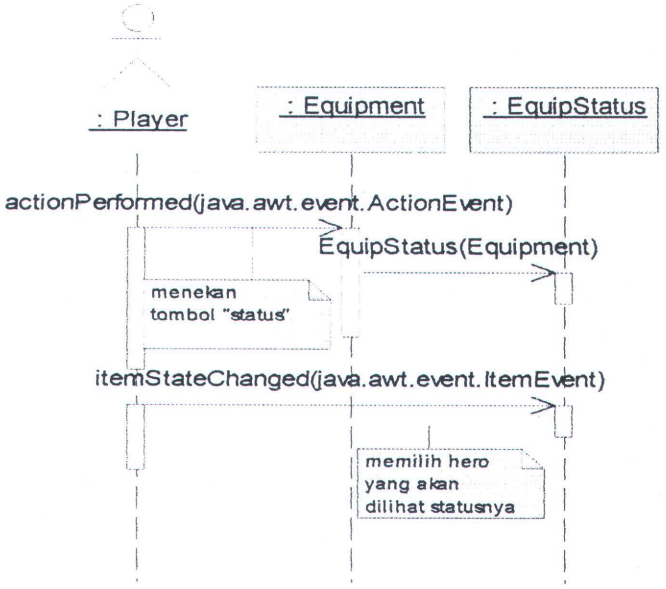


Gambar 3.22 Sequence Diagram : Manage *Equipment* / Use Item

Rangkaian proses yang terjadi dalam sequence diagram di atas adalah :

1. Jika pemain menekan tombol *"item"* pada menu perlengkapan maka akan muncul menu *equip item* yang merupakan antar muka untuk menggunakan item yang ada pada perbekalan *hero*. Pada menu ini akan ditampilkan daftar item pada perbekalan dan daftar *hero*.
2. Lalu pemain memilih item yang akan digunakan pada daftar item.
3. Kemudian pemain memilih *hero* yang akan menggunakan item tersebut pada daftar hero.

4. Setelah itu pemain menekan tombol *"use"* sehingga metode *effect()* pada item dijalankan. Pada metode tersebut akan ditentukan efek yang ditimbulkan pada *hero* pengguna berdasarkan tipe dari item tersebut.



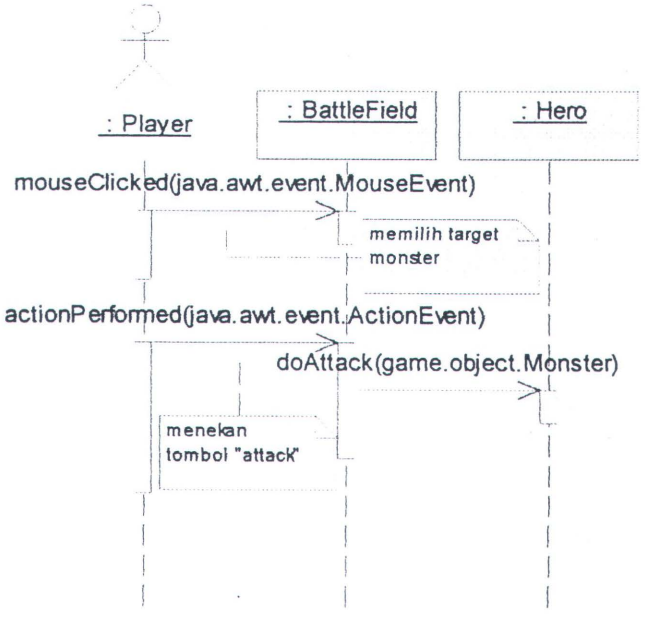
Gambar 3.23 Sequence Diagram : Manage *Equipment* / See Status

Rangkaian proses yang terjadi dalam sequence diagram di atas adalah :

1. Jika pemain menekan tombol *"status"* pada menu *equipment* maka akan muncul menu *equip status* yang merupakan antar muka untuk melihat status para *hero* dalam tim serta *hero* siapa saja yang masuk dalam tim tempur. Pada menu ini akan ditampilkan daftar *hero*.
2. Pemain bisa memilih *hero* untuk dilihat statusnya pada daftar hero. Status *hero* meliputi *hp*(*health* point), *mp*(*magic* point), *maxHp*, *maxMp*, *level*, *attack*, *defend*, *speed*, *agility*, *point*, serta *nextPoint*. Penjelasan tentang status *hero* tersebut akan dijelaskan di bawah.
3. Pemain juga bisa menekan tombol *"change"* untuk mengubah susunan tim tempur *hero*.

3.3.14 *Use Case – Do Battle*

Ketika pemain mengontrol *hero* di dalam *dungeon*, jumlah langkah *hero* menentukan pertemuan dengan *monster*. Pertemuan dengan *monster* ditentukan oleh suatu nilai random yang jika jumlah langkah *hero* telah mencapainya maka diciptakanlah suatu *BattleField*(arena pertempuran antara tim *hero* melawan tim *monster*). Use case ini menggambarkan rangkaian proses yang dilakukan oleh pemain untuk melakukan pertempuran. Pemain bisa menentukan setiap aksi *hero* dalam pertempuran melalui *BattlePanel*(panel tempur) yang di dalamnya terdapat tombol *"attack"*, *"magic"*, *"item"*, *"guard"*, *"run"*, dan *"SP"*. Tombol-tombol tersebut akan aktif bila waktu tunggu hero untuk menerima perintah dari pemain telah habis. Gambar-gambar di bawah ini akan menjelaskan masing-masing proses yang dilakukan oleh tombol-tombol di atas.



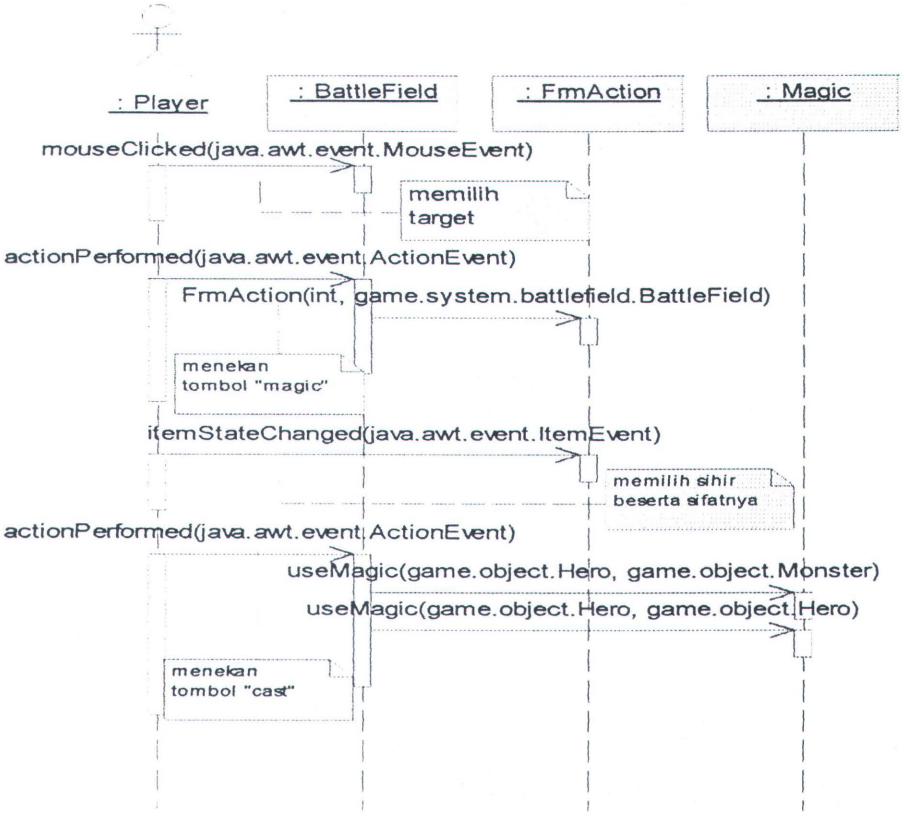
Gambar 3.24 Sequence Diagram : Do Battle / *Attack Monster*

Rangkaian proses yang terjadi dalam sequence diagram di atas adalah :

1. Mula-mula pemain harus menentukan target *monster* yang akan diserang pada *BattleField.*
2. Setelah itu pemain menekan tombol *"attack"* pada panel tempur sehingga metode *doAttack()* pada *hero* dijalankan. Serangan yang dilakukan oleh *hero* ditentukan oleh nilai kesempatan kena sasaran yang diperoleh secara random antara 100. Jika melebihi nilai *agility* dari *monster* target maka serangan tersebut mengenai sasaran dan menyebabkan *hp monster* berkurang sebesar efek serangan yang ditimbulkan. Efek serangan dihitung dengan rumus :

   *effect = (((1000*attack)/100)-((((1000*attack)/100)*target.defend)/100)).*

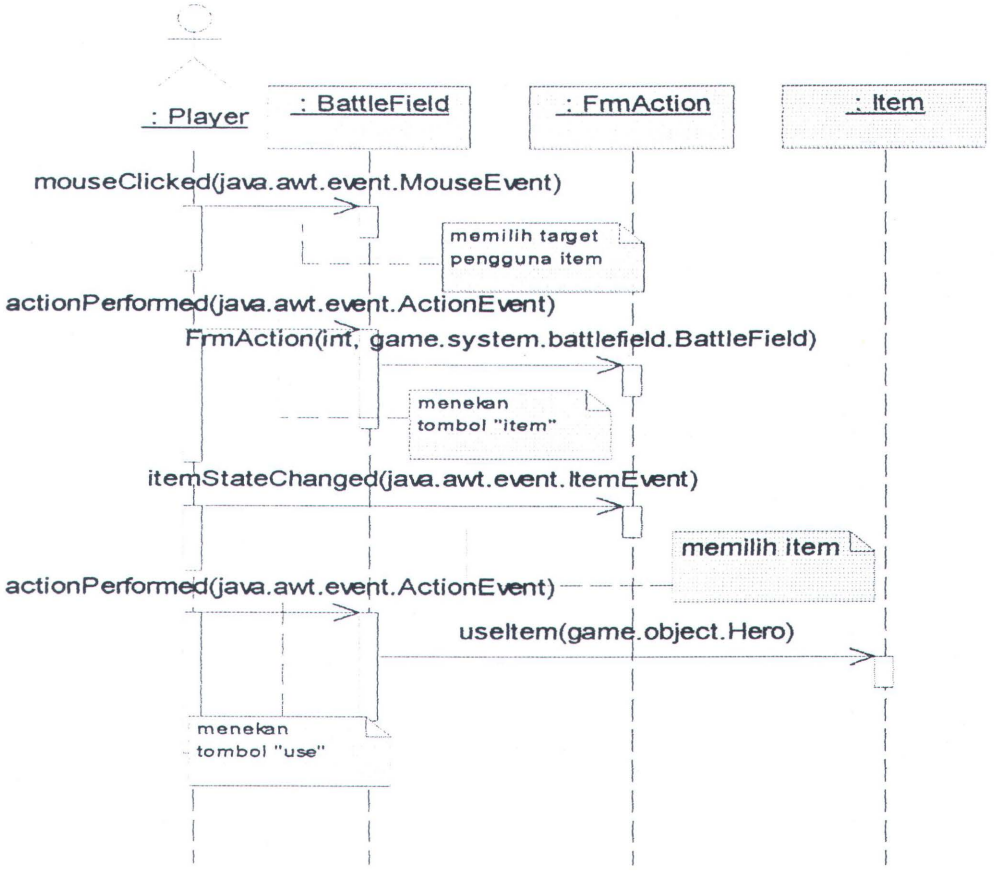
   Attack dikalikan 1000 karena nilai kerusakan maksimum adalah 1000.



Gambar 3.25 Sequence Diagram : Do Battle / Cast *Magic*

Rangkaian proses yang terjadi dalam sequence diagram di atas adalah :

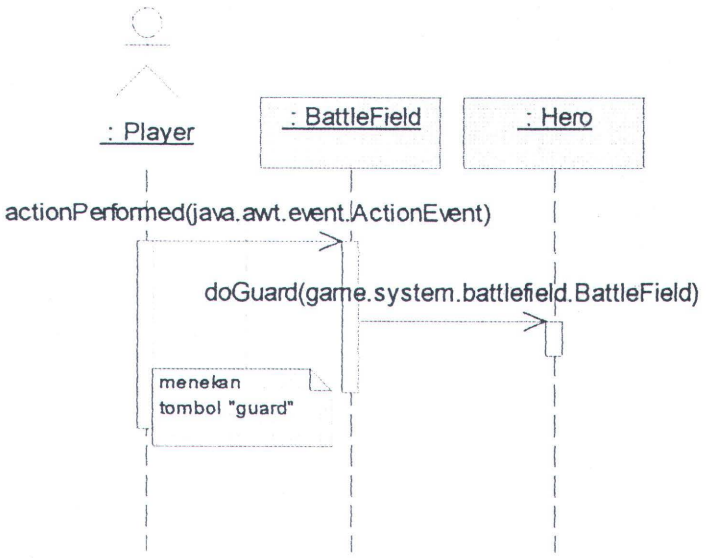
1. Mula-mula pemain harus menentukan target untuk penggunaan sihir dengan mengklik mouse pada *BattleField*. Target bisa sebagai *hero* atau *monster*. Pemain memilih target *hero* untuk sihir yang bersifat penyembuh atau memilih target *monster* untuk sihir yang bersifat menyerang.
2. Kemudian pemain menekan tombol *"magic"* pada panel tempur sehingga ditampilkan menu sihir tempur yang berisi daftar sihir yang bisa digunakan dan cek pilihan untuk memilih sifat sihir.
3. Pemain memilih sihir yang bersifat penyembuh(white magic) atau menyerang(black magic) pada cek pilihan sihir. Pemilihan tersebut mempengaruhi daftar isi dari daftar sihir.
4. Lalu pemain memilih sihir yang akan digunakan pada daftar sihir.
5. Setelah itu pemain menekan tombol *"cast"* sehingga menjalankan metode *useMagic()* pada sihir yang telah dipilih tadi. Pada metode ini akan ditentukan efek yang ditimbulkan terhadap targetnya berdasarkan tipe sihir tersebut. Tombol *"cast"* bisa ditekan jika *mp hero* mencukupi untuk menggunakan sihir tersebut.



Gambar 3.26 Sequence Diagram : Do Battle / Use Item

Rangkaian proses yang terjadi dalam sequence diagram di atas adalah :

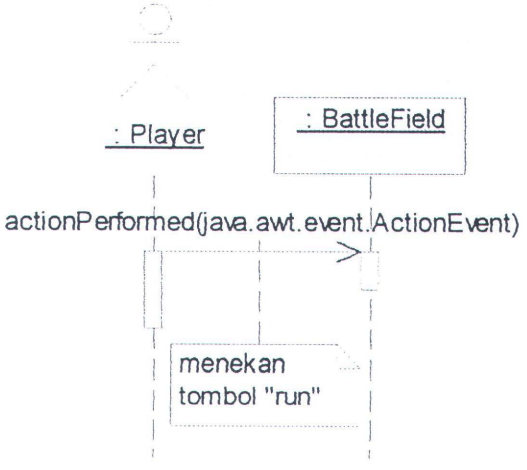
1. Mula-mula pemain harus menentukan *hero* pengguna item dengan mengklik mouse pada *BattleField.*
2. Kemudian pemain menekan tombol *"item"* pada panel tempur sehingga ditampilkan menu item tempur yang berisi daftar item yang bisa digunakan dalam pertempuran.
3. Lalu pemain memilih item yang akan digunakan pada daftar item.
4. Setelah itu pemain menekan tombol *"use"* sehingga metode *effect()* pada item tersebut dijalankan. Pada metode tersebut akan ditentukan efek yang ditimbulkan pada *hero* pengguna berdasarkan tipe dari item tersebut.



Gambar 3.27 Sequence Diagram : Do Battle / *Guard*

Rangkaian proses yang terjadi dalam sequence diagram di atas adalah :

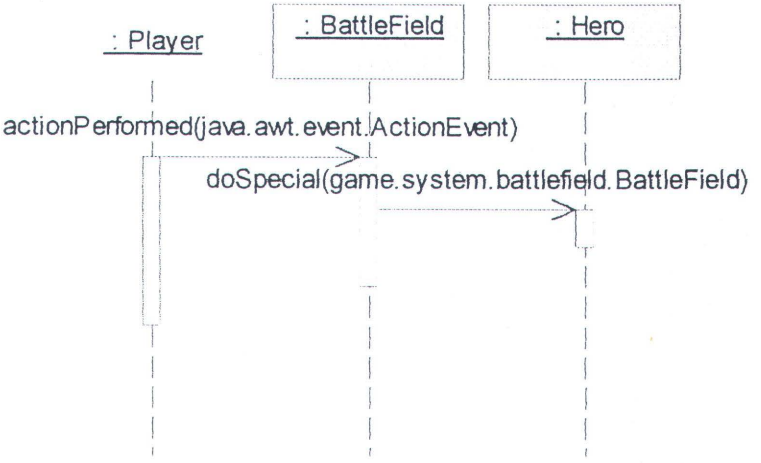
1. Pemain menekan tombol *"guard"* pada panel tempur sehingga metode *doGuard()* pada *hero* yang sedang aktif dijalankan.
2. Bertahan adalah menggandakan *defend* dari *hero* sehingga pertahanannya terhadap serangan *monster* lebih kuat.



Gambar 3.28 Sequence Diagram : Do Battle / Run

Rangkaian proses yang terjadi dalam sequence diagram di atas adalah :

1. Jika pemain ingin kabur dari pertempuran, pemain bisa menekan tombol *"run"* pada panel tempur. Tombol *"run"* tidak berfungsi pada saat melawan bos *monster*.
2. Kemungkinan lolos dari pertempuran 50/50. Jadi ada kemungkinan gagal untuk kabur. Jika *hero* berhasil kabur maka ia akan kembali ke *dungeon* semula.



Gambar 3.29 Sequence Diagram : Do Battle / Special

Rangkaian proses yang terjadi dalam sequence diagram di atas adalah :

1. Untuk mengeluarkan jurus spesial dari *hero*, pemain bisa menekan tombol *"SP"* pada panel tempur. Tombol *"SP"* baru akan aktif jika nilai kondisi spesial *hero* terpenuhi. Penambahan nilai kondisi spesial tiap *hero* berbeda-beda sesuai dengan tipenya yang akan dijelaskan nanti di bawah.
2. Setelah itu maka metode *doSpecial()* oleh *hero* dijalankan yang akan menyebabkan *hero* melakukan serangan atau penyembuhan spesial tergantung dari tipe *hero* tersebut.

3.3.15 *Use Case – Add Player*

Use case ini menggambarkan rangkaian proses yang dilakukan administrator untuk mendaftarkan pemain baru untuk bisa memainkan game tersebut pada basis data server. Pendaftaran tersebut meliputi login, password, serta nama pemain.

3.3.16 *Use Case – Add State*

Use case ini menggambarkan rangkaian proses yang dilakukan administrator untuk menambahkan *state* baru pada game. *State* tersebut bisa berupa *town* atau *dungeon*. Administrator yang mengatur segala aspek yang ada di dalam *town* atau *dungeon* tersebut.

3.3.17 *Use Case – Create vState*

Use case ini menggambarkan rangkaian proses yang dilakukan administrator untuk menciptakan obyek vState baru. Obyek vState adalah vector yang berisi rancangan segala aspek yang ada dalam state yang dirangkum dalam obyek SetState. Obyek vState tersebut disimpan dalam bentuk file "*.sta" pada harddisk server game. Tindakan ini diambil oleh administrator bila ingin menciptakan dunia permainan yang baru.

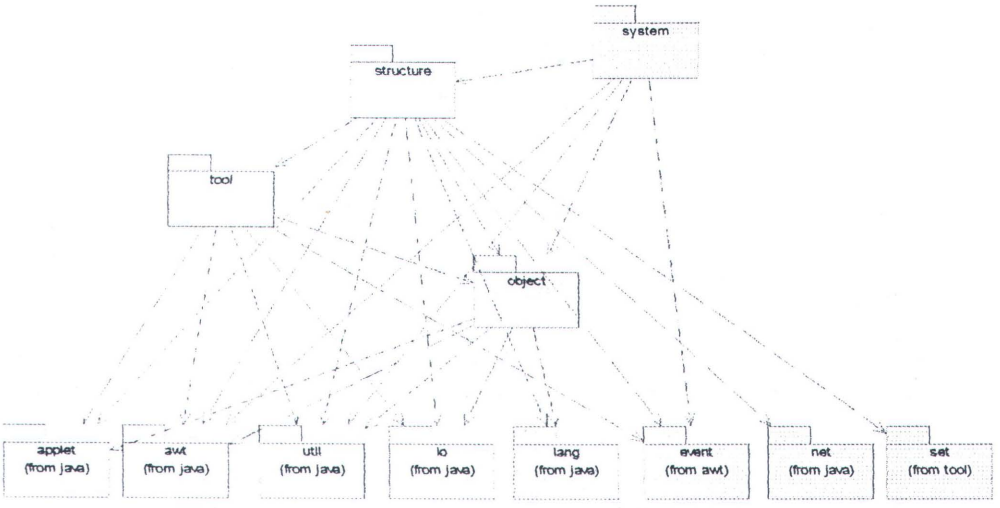
3.3.18 *Use Case – Select vState*

Use case ini menggambarkan rangkaian proses yang dilakukan administrator untuk memilih obyek vState yang telah ada untuk di-load. Administrator setiap saat bisa mengubah dunia permainan game berdasarkan setting yang telah ada.

3.4 Perancangan Game

Bagian ini adalah bagian yang berjalan pada sisi klien dalam hal ini pemain. Game ini berupa applet yang dijalankan oleh pemain melalui web browser.

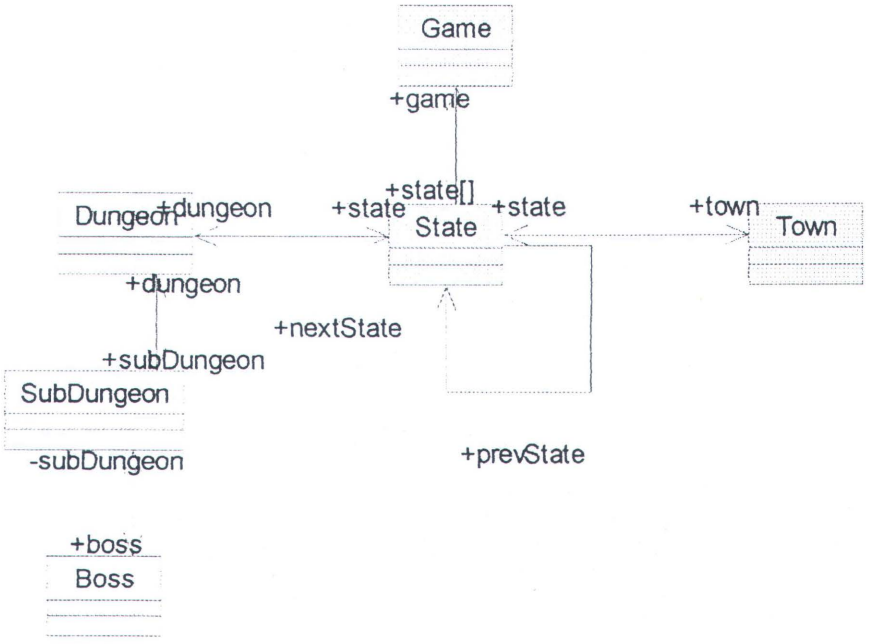
Game ini dibagi menjadi empat bagian besar yakni obyek, struktur pembangun game, sistem, dan *tool* yang masing-masing dibentuk sebagai paket. Pada bagian ini akan dibahas kelas-kelas yang ada dalam tiap paket di atas.



Gambar 3.30 Class Diagram : game / Package Overview

3.4.1 Paket *Structure*

Paket ini berisi kelas-kelas yang membangun struktur dari game ini. Terdapat enam kelas dalam paket ini yakni Game, *State*, *Town*, *Dungeon*, *SubDungeon* dan *Boss*.



Gambar 3.31 Class Diagram : structure / Package Overview

Masing-masing dari kelas di atas akan dibahas di bawah ini.

3.4.1.1 Kelas Game

Kelas ini merupakan kelas utama dari game ini yang merupakan turunan dari kelas Applet dari paket Java. Inisialisasi seluruh obyek yang akan digunakan dalam game dilakukan dalam kelas ini. Dalam kelas ini pula perbekalan tim *hero* diletakkan seperti senjata, baju, aksesoris, item dan uang. Kelas ini juga berisi metode-metode utama yang digunakan oleh obyek-obyek game yang lain. Di kelas ini diletakkan pula tim tempur serta tim keseluruhan *hero*.

Penjelasan dari tiap metode utama dari kelas Game akan dijelaskan di bawah.

3.4.1.1.1 Metode *newGame()*

Metode ini dijalankan ketika pemain memilih untuk menciptakan permainan baru di awal game. Dalam metode ini seluruh *state* serta tim *hero* awal akan

diinisialisasikan. Setelah itu cerita pembuka game akan ditampilkan oleh metode ini.

3.4.1.1.2 Metode *buyWeapon()*, *buyArmor()*, *buyAccesoris()*, dan *buyItem()*

Metode-metode di atas digunakan untuk membeli perlengkapan *hero*. Metode-metode di atas digunakan oleh toko-toko perlengkapan yang ada di *town*. Metode di atas akan menambah perlengkapan pada perbekalan *hero* sesuai barang yang dibeli dan akan mengurangi uang *hero* sesuai dengan harga beli barang tersebut.

3.4.1.1.3 Metode *sellWeapon()*, *sellArmor()*, *sell Accesoris()*, dan *sellItem()*

Metode-metode di atas digunakan untuk menjual perlengkapan *hero*. Metode-metode di atas juga digunakan oleh toko-toko perlengkapan yang ada di *town*. Metode di atas akan mengurangi perlengkapan pada perbekalan *hero* sesuai yang ingin dijual dan akan menambah uang *hero* sesuai dengan harga jual barang tersebut.

3.4.1.1.4 Metode *putWeapon()*, *putArmor()*, *putAccesoris()*, dan *putItem()*

Metode-metode di atas digunakan untuk meletakkan perlengkapan ke dalam perbekalan *hero*. Metode di atas digunakan untuk berbagai keperluan seperti meletakkan isi kotak barang atau meletakkan senjata hasil upgrade.

3.4.1.1.5 Metode *saveGame()*

Metode ini digunakan untuk menyimpan status permainan. Metode ini dipanggil ketika pemain menekan key "enter" pada save point yang berada di *town* atau *dungeon*.

Dalam metode ini akan diciptakan obyek *SavePack* yang merupakan obyek yang berisi salinan seluruh obyek utama pada posisi *state* dimana *hero* saat ini berada seperti *hero*, *state*, serta seluruh perbekalan perlengkapan *hero* yang

digunakan untuk inisialisasi game pada proses load game. Kemudian obyek *SavePack* ini dikirim ke server game melalui *socket* port 10000.

3.4.1.1.6 Metode *loadGame()*

Metode ini digunakan untuk melanjutkan permainan yang pernah disimpan oleh pemain. Metode ini dipanggil ketika pemain menekan tombol "continue" pada awal permainan.

Dalam metode ini game akan meminta server game untuk mengirimkan obyek *SavePack* berdasarkan login pemain saat itu melalui *socket* port 10001. Kemudian setelah obyek *SavePack* tersebut diterima oleh game maka seluruh obyek utama game seperti *hero*, *state* dan seluruh perbekalan perlengkapan *hero* diinisialisasikan untuk memulai permainan dari posisi *state* sesuai dengan id *state* pada *SavePack*.

3.4.1.1.7 Metode *rankGame()*

Metode ini digunakan untuk menampilkan urutan ranking seluruh pemain game ini yang telah terdaftar pada basis data server game.

Dalam metode ini game akan meminta server game untuk mengirimkan obyek yang berisi urutan ranking seluruh pemain melalui *socket* port 10001. Kemudian setelah obyek tersebut diterima oleh game maka urutan ranking pemain tersebut ditampilkan pada bingkai ranking.

3.4.1.2 Kelas *State*

Kelas ini dalam game akan disusun dalam bentuk urutan *state-state* yang membentuk alur perjalanan *hero*. Kelas ini merupakan turunan dari kelas Label yang mengimplementasikan *MouseListener* dari paket Java.

Hero akan memasuki *state-state* tersebut satu-persatu secara berurutan. Untuk masuk ke dalam suatu *state* adalah dengan mengklik *state* tersebut. Pemain tidak bisa memilih *state* secara tidak berurutan atau melompati suatu *state*. Pemain hanya bisa menuju *state* selanjutnya atau sebelumnya.

Pada tiap *state* terdapat kunci untuk memasukinya yang hanya bisa dibuka oleh suatu kondisi tertentu saja. Biasanya kunci pembuka tersebut berada pada *state* sebelumnya. Pemain harus menyelesaikan suatu misi tertentu untuk membuka kunci *state* selanjutnya. Misi tersebut biasanya antara lain menemui *people* khusus atau mengalahkan bos *monster*.

Setiap *state* bisa berupa *town* atau *dungeon* tergantung dari inisialisasi pada metode newGame() atau loadGame() di kelas Game. Setiap *state* juga memiliki id yang digunakan untuk inisialisasi pada saat proses load game.

3.4.1.3 Kelas *Town*

Kelas ini merupakan turunan dari kelas Label yang mengimplementasikan *Runnable* dari paket Java sehingga mempunyai sifat *thread*. *Town* berupa suatu area yang di dalamnya terdapat obyek-obyek kota antara lain rumah, tanaman, *people*, dan sebagainya. Susunan dari obyek-obyek beserta gambar wilayah *town* tersebut diatur berdasarkan inisialisasi yang diperoleh dari server game sehingga setiap *town* memiliki lingkungan yang berbeda-beda.

Pada *town* terdapat suatu *thread* gambar yang berfungsi untuk menggambar ulang seluruh isi *town* berdasarkan pergerakan *hero*. Pergerakan *hero* pada *Town* dilakukan pemain melalui key arah pada keyboard. Penekanan key tersebut akan membangkitkan *thread* gambar tersebut.

Di bawah ini akan dijelaskan masing-masing dari tiap metode utama yang ada pada kelas *Town*.

3.4.1.3.1 Metode *insertMap()*

Metode ini digunakan untuk meletakkan gambar wilayah *town*. Ditentukan pula panjang dan lebar dari gambar tersebut yang akan menjadi ukuran luas *town*.

3.4.1.3.2 Metode *putHero()*

Metode ini digunakan untuk meletakkan *hero* pada *town*. Peletakkan *hero* tersebut akan mempengaruhi koordinat gambar wilayah *town* karena sudut pandang game dari atas kepala *hero*.

3.4.1.3.3 Metode *putObject()*

Metode ini digunakan untuk meletakkan obyek pada *town*. Peletakkan obyek tersebut berdasarkan pada koordinat gambar wilayah *town*.

3.4.1.3.4 Metode *putPeople()*

Metode ini digunakan untuk meletakkan *people* pada *town*. Seperti halnya pada obyek, peletakkan *people* tersebut berdasarkan pada koordinat gambar wilayah *town*. *People* akan diletakkan pada *town* jika atribut kenampakannya bernilai true.

3.4.1.3.5 Metode *setHero()*

Metode ini digunakan untuk mengambil gambar *hero* pada tim tempur untuk digunakan ketika menggambar *hero* pada *town*. *Hero* pada urutan pertama dalam tim tempur yang akan ditampilkan *town*.

3.4.1.3.6 Metode *preDraw()*

Metode ini digunakan untuk inisialisasi seluruh obyek *town* beserta *people*. Obyek serta *people* tersebut akan dimasukkan dalam array yang nantinya digunakan dalam proses menggambar *town*.

3.4.1.3.7 Metode *paint()*

Metode ini merupakan bawaan dari kelas Component dari paket Java. Dalam kelas *Town* yang juga merupakan turunan dari kelas Component metode tersebut digunakan untuk menggambar seluruh gambar obyek dalam *town* termasuk di dalamnya wilayah *town*, *hero*, rumah, *people*, serta obyek lainnya.

Metode ini dijalankan secara berulang ketika *thread* gambar dibangkitkan. *Thread* gambar dibangkitkan pada saat pemain menekan key arah pada keyboard.

3.4.1.3.8 Metode *drawHero()*

Metode ini digunakan untuk mempersiapkan gambar *hero* yang akan digambar serta mengubah koordinat dari gambar wilayah *town* serta seluruh obyek dan *people* dalam *town* berdasarkan perubahan posisi *hero*. Perubahan posisi *hero* ditentukan oleh key arah pada keyboard yang ditekan oleh pemain.

Pembandingan nilai pada peta *town* juga dilakukan pada metode ini. Reaksi nilai peta *town* dibangkitkan ketika posisi *hero* berada pada petak yang berisi nilai tersebut.

Metode ini dipanggil oleh metode paint() sebelum menggambar *hero* dan seluruh obyek serta *people* pada *town*.

3.4.1.3.9 Metode *setTown()*

Metode ini digunakan untuk menginisialisasi suatu *town* dari salinan *state* pada saat proses load game berdasarkan id *state* tersebut. Seluruh atribut beserta obyek yang ada dalam *town* akan diinisialisasikan dari salinan *state* tersebut.

3.4.1.4 Kelas *Dungeon*

Kelas ini memiliki sifat yang identik dengan kelas *Town*. Kelas ini juga merupakan turunan dari kelas Label yang mengimplementasikan *Runnable* dari paket Java sehingga mempunyai sifat *thread*. *Dungeon* juga berupa suatu area yang

di dalamnya terdapat berbagai obyek seperti pohon, bebatuan dan sebagainya. Susunan dari obyek-obyek beserta gambar wilayah *dungeon* tersebut juga diatur berdasarkan inisialisasi yang diperoleh dari server game sehingga setiap *dungeon* memiliki lingkungan yang berbeda-beda. Sistem pemetaan *dungeon* juga identik dengan *town*.

Perbedaan kelas ini adalah kelas *Dungeon* disusun oleh beberapa *subDungeon* yang di dalamnya berisi seluruh obyek yang akan digambar oleh *dungeon* serta gambar wilayahnya.

Sebagian besar metode pada kelas *Dungeon* sama dengan kelas *Town*. Berikut ini akan dijelaskan metode-metode yang sifatnya berbeda dengan yang ada pada kelas *Town*.

3.4.1.4.1 Metode *addDungeon()*

Metode ini digunakan untuk menambahkan *subDungeon* pada *Dungeon*.

3.4.1.4.2 Metode *preDraw()*

Metode ini digunakan untuk mempersiapkan *subDungeon* yang akan digambar oleh *thread* gambar *Dungeon* pada saat *hero* memasuki *dungeon*.

3.4.1.4.3 Metode *preDrawNext()*

Metode ini digunakan untuk mempersiapkan *subDungeon* selanjutnya yang akan digambar oleh *thread* gambar *Dungeon*. Jika telah melewati *subDungeon* terakhir maka *hero* akan keluar dari *Dungeon*.

3.4.1.4.4 Metode *preDrawPrev()*

Metode ini digunakan untuk mempersiapkan *subDungeon* sebelumnya yang akan digambar oleh *thread* gambar *Dungeon*. Jika telah melewati *subDungeon* pertama maka *hero* akan keluar dari *Dungeon*.

3.4.1.4.5 Metode *drawHero()*

Metode ini identik dengan milik kelas *Town*. Perbedaan pertama pada saat posisi *hero* pada petak yang bernilai (total obyek + 2 + total kotak barang) akan dijalankan metode preDrawPrev(). Jika petak bernilai (total obyek + 3 + total kotak barang) akan dijalankan metode preDrawNext() atau jika *subDungeon* merupakan tempat bos *monster* maka akan disiapkan pertarungan melawan bos *monster*.

Perbedaan kedua adalah karena *monster* yang ada di *dungeon* tidak nampak di layar maka langkah *hero* pada *dungeon* diperhitungkan. Pada *dungeon* terdapat suatu nilai encounter yang diperoleh secara random. Jika langkah *hero* telah mencapai nilai tersebut maka diciptakanlah *BattleField*(arena pertempuran antara tim *hero* melawan tim *monster*).

3.4.1.5 Kelas *SubDungeon*

Kelas ini merupakan bagian dari kelas *Dungeon* yang berisi seluruh obyek yang akan digambar oleh *dungeon*. Kelas ini juga memiliki sifat yang identik dengan kelas *Town* dalam hal penempatan obyek-obyek di dalamnya. Perbedaannya hanya pada *subDungeon* yang diletakkan bukanlah *people* melainkan kotak barang. Pada *subDungeon* juga terdapat bos *monster*.

Berikut ini akan dijelaskan beberapa metode khusus yang terdapat pada kelas *SubDungeon*.

3.4.1.5.1 Metode *putItemBox()*

Metode ini digunakan untuk meletakkan kotak barang pada *dungeon*. Peletakkan kotak barang tersebut berdasarkan pada koordinat gambar wilayah *subDungeon*.

3.4.1.5.2 Metode *putBoss()*

Metode ini digunakan untuk meletakkan bos *monster* pada *dungeon*. Peletakkan bos *monster* tersebut berdasarkan pada koordinat gambar wilayah *subDungeon*.

3.4.1.5.3 Metode *setDungeon()*

Metode ini digunakan untuk menginisialisasi suatu *subDungeon* dari salinan *state* pada saat proses load game berdasarkan id *state* tersebut. Seluruh atribut beserta obyek yang ada dalam *subDungeon* akan diinisialisasikan dari salinan *state* tersebut.

3.4.1.6 Kelas *Boss*

Kelas ini merupakan perwakilan dari bos *monster* yang diletakkan pada *subDungeon* terakhir pada *dungeon*. Melalui kelas ini akan ditentukan kejadian yang terjadi ketika *hero* bertemu dengan bos *monster* tersebut.

Berikut ini dijelaskan metode-metode yang ada pada kelas *Boss*.

3.4.1.6.1 Metode *preBattle()*

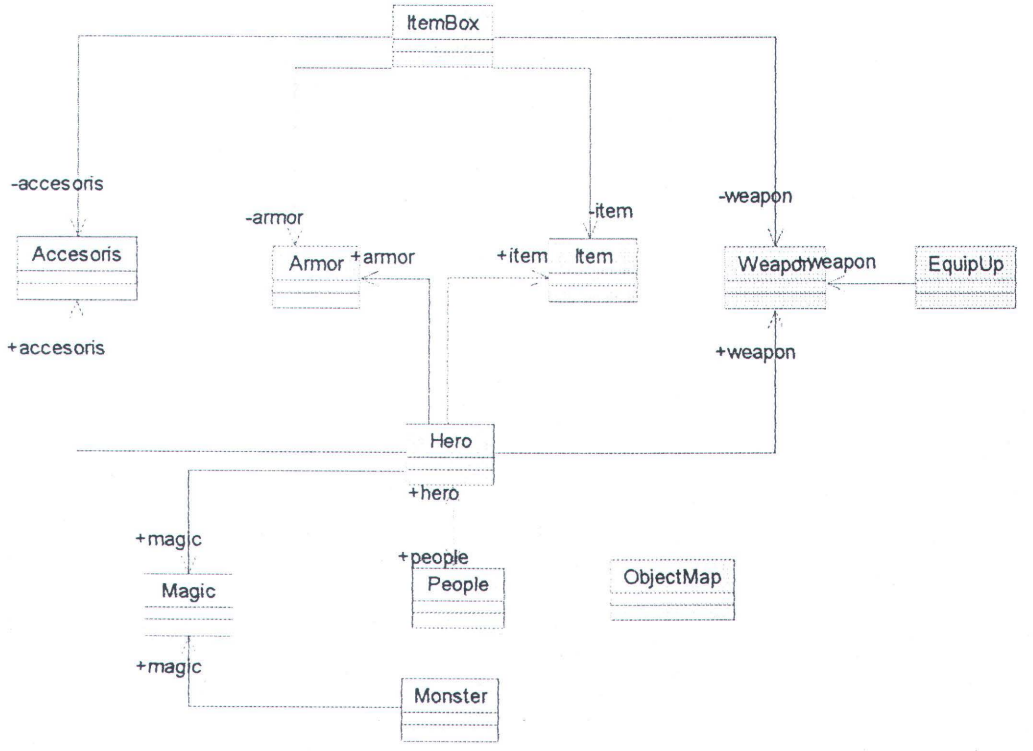
Metode ini digunakan untuk menampilkan kejadian yang terjadi ketika *hero* bertemu dengan bos *monster*. Dalam metode ini akan ditampilkan cerita pendahulu kemudian akan diciptakan *BattleField* untuk melawan bos *monster* tersebut.

3.4.1.6.2 Metode *setBoss()*

Metode ini digunakan untuk menginisialisasi bos *monster* dari salinan *state* pada saat proses load game berdasarkan id *state* tersebut.

3.4.2 Paket Object

Paket ini berisi kelas-kelas yang akan menjadi obyek-obyek yang akan berperan untuk berbagai keperluan dalam game. Terdapat 10 kelas dalam paket ini yakni *Hero*, *People*, *Monster*, *ItemBox*, *Weapon*, *Armor*, *Accesoris*, Item, *Magic*, dan ObjectMap.



Gambar 3.32 Class Diagram : object / Package Overview

Masing-masing dari kelas di atas akan dibahas dibawah ini.

3.4.2.1 Kelas *Hero*

Hero merupakan suatu kelas yang merupakan turunan dari kelas *Thread* dari paket Java. *Thread* tersebut digunakan untuk menjalankan aksi *hero*. *Hero* adalah tokoh yang menjadi pelaku dan bertualang dalam game.

Hero memiliki status antara lain *hp*(*health* point), *mp*(*magic* point), *maxHp*, *maxMp*, *level*, *attack*, *defend*, *speed*, *agility*, *point*, serta *nextPoint*.

Hero dapat melengkapi dirinya dengan senjata, baju atau aksesoris yang sesuai dengan tipenya. Senjata dapat meningkatkan *attack hero*. Baju dapat meningkatkan *defend hero*. Aksesoris dapat memberikan kemampuan tambahan bagi *hero*.

Hero juga memiliki sihir yang diperoleh melalui kenaikan levelnya. Sihir *hero* bisa digunakan bila mp *hero* mencukupi.

Dalam game ini, terdapat empat tipe hero yakni swordman, sorcerer, brawler, dan shooter. Setiap tipe hero memiliki fungsi kenaikan status yang spesifik pada saat kenaikan level serta sihir yang dipelajari oleh tiap tipe juga berbeda pula.

Hero memiliki pula nilai spesial yang bila telah mencapai 5 hero bisa mengeksekusi jurus spesialnya.

Berikut ini akan dijelaskan metode-metode penting yang ada pada kelas *Hero*.

3.4.2.1.1 Metode *setWeapon()*, *setArmor()*, dan *setAccesoris()*

Metode-metode ini digunakan untuk memasang perlengkapan pada *hero*. Pemasangan perlengkapan tersebut memberikan efek yang berbeda-beda pada *hero* sesuai dengan jenisnya.

3.4.2.1.2 Metode *unsetWeapon()*, *unsetArmor()*, dan *unsetAccesoris()*

Metode-metode ini digunakan untuk melepas perlengkapan pada *hero*. Pelepasan perlengkapan tersebut mengakibatkan efek perlengkapan tersebut pada *hero* akan lepas pula.

3.4.2.1.3 Metode *setHero()*

Metode ini digunakan untuk menginisialisasi *hero* dari salinan *hero* pada saat proses load game. Seluruh status beserta perlengkapannya akan diinisialisasikan berdasarkan salinan *hero* tersebut.

3.4.2.1.4 Metode *setLevel()*

Metode ini digunakan untuk menset level *hero*.

3.4.2.1.5 Metode *setBattle()*

Metode ini digunakan untuk mempersiapkan *hero* dalam pertempuran.

3.4.2.1.6 Metode *doAttack()*

Metode ini digunakan untuk menyerang *monster* lawan dalam pertempuran.

3.4.2.1.7 Metode *doGuard()*

Metode ini digunakan untuk bertahan dari serangan *monster* lawan dalam pertempuran.

3.4.2.1.8 Metode *doSpecial()*

Metode ini digunakan untuk mengeluarkan jurus spesial *hero* dalam pertempuran.

3.4.2.1.9 Metode *cekLevel()*

Metode ini digunakan untuk mengecek poin *hero* apakah sudah memenuhi untuk menaikkan level *hero*. Jika level *hero* naik maka maxHp, maxMp, *attack*, *defend*, *speed*, serta *agility hero* juga meningkat pula.

3.4.2.1.10 Metode *cekWin()*

Metode ini digunakan untuk mengecek apakah *hero* telah berhasil membunuh *monster* lawan. Jika *hero* berhasil membunuh *monster* lawan akan diperoleh point serta uang dari *monster* tersebut.

3.4.2.1.11 Metode *run()*

Metode ini merupakan bawaan dari kelas *Thread*. Metode ini digunakan untuk memasukkan *hero* ke dalam antrian tempur. Kecepatan masuk antrian tempur ditentukan oleh status *speed hero*.

3.4.2.2 Kelas *Monster*

Kelas *Monster* memiliki sifat sebagian besar identik dengan kelas *Hero*. Kelas ini juga merupakan turunan dari kelas *Thread* dari paket Java. Memang ada beberapa bagian dari kelas *Hero* yang tidak terdapat pada kelas *Monster*.

Monster adalah musuh dalam game yang harus dilawan oleh *hero* dalam pertempuran. *Monster* juga memiliki status yang sama dengan *hero*.

Perbedaannya adalah pada setiap *monster* memiliki status moneyGet serta pointGet yang berbeda-beda. MoneyGet adalah uang yang didapat *hero* jika berhasil membunuh *monster* tersebut. PointGet adalah poin yang didapat *hero* jika berhasil membunuh *monster* tersebut.

Monster juga memiliki status elemen antara lain water, fire, dan undead. Elemen monster mempengaruhi efek yang diterima monster ketika mendapat serangan sihir dari hero.

Berikut ini akan dijelaskan metode khusus yang terdapat pada kelas *Monster*.

3.5.2.2.1 Metode *doAction()*

Metode ini digunakan untuk menentukan setiap aksi *monster* dalam pertempuran berdasarkan tipenya. Dalam pertempuran, *monster* memilih *hero* yang akan diserangnya serta aksi yang akan dilakukannya secara random. Setiap tipe *monster* memiliki kombinasi aksi yang berbeda-beda.

3.4.2.3 Kelas *Weapon*

Weapon adalah senjata yang digunakan *hero* untuk meningkatkan daya serang *hero* dalam pertempuran. Senjata memiliki status antara lain plus*Attack*, plus*Defend*, plus*Speed*, plusAgile, user, buyPrice, serta sellPrice.

Plus*Attack* adalah efek status *attack* yang diperoleh *hero* bila menggunakannya. Plus*Defend* adalah efek status *defend* yang diperoleh *hero* bila menggunakannya. Plus*Speed* adalah efek status *speed* yang diperoleh *hero* bila menggunakannya. PlusAgile adalah efek status *agility* yang diperoleh *hero* bila menggunakannya. User adalah tipe *hero* yang bisa menggunakan senjata tersebut. BuyPrice adalah harga beli senjata tersebut. SellPrice adalah harga jual senjata tersebut.

Setiap tipe senjata juga memiliki index yang digunakan untuk menemukan posisinya pada *VectorEquip*.

3.4.2.4 Kelas *Armor*

Armor adalah baju yang dikenakan *hero* untuk meningkatkan daya tahan *hero* dalam pertempuran. Baju juga memiliki status yang sama dengan senjata antara lain plus*Attack*, plus*Defend*, plus*Speed*, plusAgile, user, buyPrice, serta sellPrice.

Setiap tipe baju juga memiliki index yang digunakan untuk menemukan posisinya pada *VectorEquip*.

3.4.2.5 Kelas *Accesoris*

Accesoris adalah aksesoris yang dikenakan *hero* untuk memberikan kemampuan tambahan bagi *hero* tersebut. Kemampuan tambahan tersebut antara lain peningkatan status *hero* atau kekebalan terhadap status tidak normal.

Setiap tipe aksesoris juga memiliki index yang digunakan untuk menemukan posisinya pada *VectorEquip*.

Setiap tipe aksesoris juga memiliki harga beli dan harga jual yang berbeda-beda pula.

3.4.2.6 Kelas *Item*

Item digunakan untuk menyembuhkan status tidak normal *hero*. Setiap tipe item memiliki khasiat yang berbeda-beda. Harga beli serta harga jual tiap tipe item juga berbeda-beda pula.

Setiap tipe item juga memiliki index yang digunakan untuk menemukan posisinya pada *VectorEquip*.

Item dibagi menjadi dua berdasarkan targetnya yakni all dan tidak all. All yakni efek item mengena pada semua target. Tidak all yakni hanya target yang dipilih pemain terkena efek item.

Berikut ini akan dijelaskan metode utama yang ada pada kelas Item.

3.5.2.6.1 Metode *useItem()*

Metode ini digunakan untuk memberikan efek item bagi *hero* penggunanya.

3.4.2.7 Kelas *Magic*

Magic adalah sihir yang dimiliki oleh *hero* atau *monster*. Sifat sihir dibagi menjadi dua macam yakni black *magic* dan white *magic*. Black *magic* adalah sihir

yang bersifat menyerang. White *magic* adalah sihir yang bersifat menyembuhkan. Sihir juga dibagi menjadi dua berdasarkan targetnya yakni all dan tidak all. All yakni efek sihir mengena pada semua target. Tidak all yakni hanya target yang dipilih pemain terkena efek sihir.

Untuk *hero*, sihir bisa digunakan bila mp *hero* mencukupi untuk menggunakan sihir tersebut. Setiap tipe sihir memiliki konsumsi mp yang berbeda-beda.

Berikut ini akan dijelaskan metode utama pada kelas *Magic*.

3.5.2.7.1 Metode *useMagic()*

Metode ini digunakan untuk memberikan efek sihir bagi pengguna atau targetnya tergantung dari sifat sihir tersebut.

3.4.2.8 Kelas *People*

People adalah penduduk yang ada pada *town*. Peran setiap *people* berbeda-beda sesuai dengan pekerjaannya. *Hero* akan berkomunikasi dengan poeple tersebut di *town*.

Pekerjaan dari tiap *people* berbeda-beda antara lain sebagai penduduk biasa, pemegang kunci *state* selanjutnya, teman baru *hero*, penjual toko, pemilik penginapan, pencatat perjalanan, atau kombinasi di antaranya.

Penduduk biasa hanya akan memberikan respon dialog percakapan bila bertemu *hero*.

Pemegang kunci *state* selain memberikan respon dialog juga akan membuka kunci *state* selanjutnya sehingga *hero* bisa memasuki *state* tersebut.

Teman baru *hero* akan menjadi anggota baru dalam tim *hero*. Level *hero* baru tersebut akan disesuaikan dengan *hero* yang telah ada.

Penjual toko akan menampilkan form belanja toko. Pemilik penginapan akan menampilkan menu penginapan.

Pencatat perjalanan akan menyimpan status permainan saat itu.

Berikut ini akan dijelaskan metode-metode pada kelas *People*.

3.4.2.8.1 Metode *setJob()*

Metode ini digunakan untuk membangkitkan reaksi pekerjaan dari *people* tersebut.

3.4.2.8.2 Metode *setPeople()*

Metode ini digunakan untuk menginisialisasi *people* dari salinan *people* pada saat proses load game.

3.4.2.8.3 Metode setImage()

Metode ini digunakan untuk mempersiapkan gambar *people* sesuai dengan pekerjaannya.

3.4.2.9 Kelas *ItemBox*

ItemBox adalah kotak barang yang berada tersembunyi di *dungeon* yang berisi perlengkapan, item, uang, atau prasasti perjalanan.

Berikut ini akan dijelaskan metode-metode yang ada pada kelas *ItemBox*.

3.4.2.9.1 Metode *setContent()*

Metode ini digunakan untuk meletakkan isi kotak barang ke dalam perbekalan *hero*. Bila merupakan prasasti perjalanan akan menyimpan status permainan.

3.4.2.9.2 Metode *setBox()*

Metode ini digunakan untuk menginisialisasi kotak barang dari salinan kotak barang pada saat proses load game.

3.4.2.9.3 Metode *addContent()*

Metode ini digunakan untuk mengisi kotak barang dengan sesuatu.

3.4.2.10 Kelas *ObjectMap*

ObjectMap merupakan obyek-obyek yang diletakkan pada *town* atau *dungeon*. ObjectMap tersebut bisa berupa rumah, pohon, batu, dan sebagainya.

Metode yang ada pada kelas ObjectMap akan dijelaskan sebagai berikut.

3.5.2.10.1 Metode *setObject()*

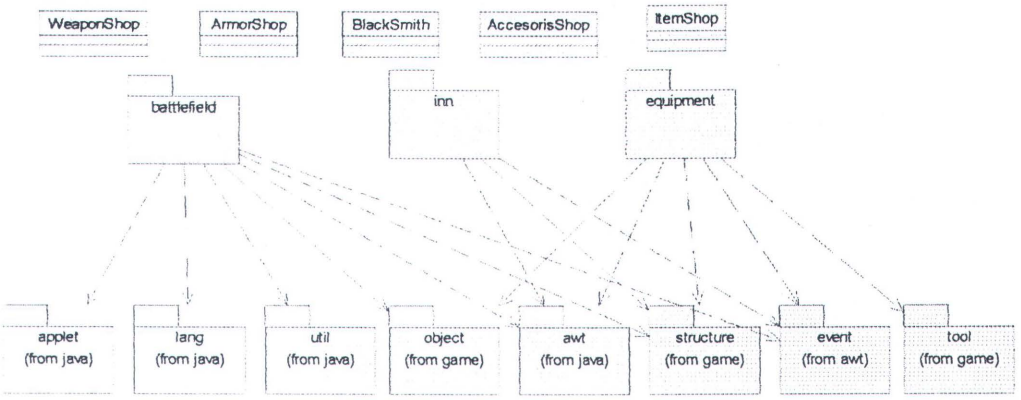
Metode ini digunakan untuk menginisialisasi obyek dari salinan obyek pada saat proses load game.

3.4.2.11 Kelas *EquipUp*

*Equip*Up adalah jenis senjata hasil upgrade beserta info tentang apa saja yang dibutuhkan untuk mendapatkannya.

3.4.3 Paket System

Paket ini berisi paket-paket yang merupakan aktivitas atau transaksi yang dilakukan *hero* dalam game. Terdapat lima kelas toko pada paket System ini yakni *WeaponShop*, *ArmorShop*, *AccesorisShop*, *ItemShop*, dan *BlackSmith* serta tiga paket sistem khusus yakni *Inn*, *Equipment*, dan *BattleField*.



Gambar 3.33 Class Diagram : system/ Package Overview

Masing-masing kelas dan paket yang ada dalam paket System ini akan dijelaskan satu-persatu di bawah ini.

3.4.3.1 Kelas *Weaponshop*

Weaponshop adalah toko yang melayani transaksi jual-beli senjata di *town*. Terdapat tiga daftar pada toko senjata yakni daftar senjata yang ditawarkan oleh penjual yang akan dibeli oleh pemain, daftar senjata yang ingin dijual oleh pemain, dan daftar *hero* yang bisa menggunakan senjata tersebut.

3.4.3.2 Kelas *Armorshop*

Armorshop adalah toko yang melayani transaksi jual-beli baju di *town*. Terdapat tiga daftar toko baju yakni daftar baju yang ditawarkan oleh penjual yang akan dibeli oleh pemain, daftar baju yang ingin dijual oleh pemain, dan daftar *hero* yang bisa menggunakan baju tersebut.

3.4.3.3 Kelas *AccesorisShop*

Accesorisshop adalah toko yang melayani transaksi jual-beli aksesoris di *town*. Terdapat tiga daftar pada toko aksesoris yakni daftar aksesoris yang

ditawarkan oleh penjual yang akan dibeli oleh pemain, daftar aksesoris yang ingin dijual oleh pemain, dan daftar *hero* yang akan menggunakan aksesoris tersebut.

3.4.3.4 Kelas *Itemshop*

Itemshop adalah toko yang melayani transaksi jual-beli item di *town*. Terdapat dua daftar pada toko item yakni daftar item yang ditawarkan oleh penjual yang akan dibeli oleh pemain dan daftar item yang ingin dijual oleh pemain.

3.4.3.5 Kelas *Blacksmith*

Blacksmith adalah pandai besi yang melayani transaksi upgrade senjata di *town*. Terdapat dua daftar pandai besi yakni daftar senjata upgrade yang ditawarkan oleh pandai besi dan daftar *hero* yang akan menggunakan senjata tersebut.

3.4.3.6 Paket *Inn*

Inn adalah penginapan tempat *Hero* beristirahat untuk memulihkan tenaga di *town*. Terdapat label pilihan pada paket inn yakni LabelInn.

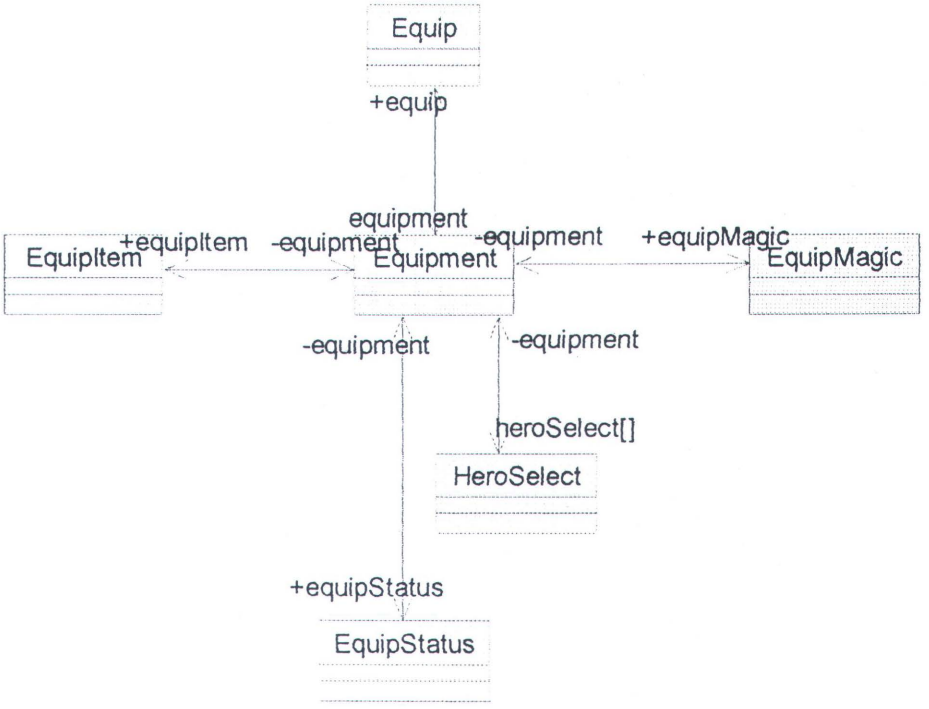
Gambar 3.34 Class Diagram : inn / Package Overview

LabelInn adalah label pilihan pada penginapan untuk menentukan apakah pemain mau mengistirahatkan *hero* atau tidak.

3.4.3.7 Paket *Equipment*

Equipment adalah menu yang digunakan pemain untuk mengatur segala perbekalan perlengkapan yang ada pada tim *hero*. Pada *equipment* terdapat

*Hero*Select yakni label yang digunakan untuk memilih *hero* yang aktif dalam *equipment*. Terdapat empat bagian besar dari paket *equipment* ini yakni *Equip*, *EquipMagic*, *Equip*Item, dan *Equip*Status.



Gambar 3.35 Class Diagram : *equipment* / Package Overview

Equip adalah menu yang digunakan pemain untuk mengatur perlengkapan yang dikenakan oleh *hero*. Pada menu ini terdapat dua daftar daftar perlengkapan yang akan dipilih pemain untuk dikenakan pada *hero* dan daftar tim *hero* yang akan diatur perlengkapannya oleh pemain. Terdapat pula label yang digunakan untuk memilih jenis perlengkapan yang akan ditampilkan pada List*Equip*.

EquipMagic adalah menu yang digunakan untuk menggunakan sihir yang ada pada tim *hero*. Pada menu ini terdapat dua daftar yakni daftar sihir yang dimiliki oleh *hero* yang sedang aktif dan daftar *hero* yang akan digunakan sihirnya.

*Equip*Item adalah menu yang digunakan untuk menggunakan item yang ada pada perbekalan tim *hero*. Pada menu ini terdapat dua daftar yakni daftar item pada

perbekalan tim *hero* yang akan digunakan oleh *hero* dan daftar *hero* yang akan menggunakan item tersebut.

*Equip*Status adalah menu yang digunakan untuk menampilkan status tim *hero*. Pada menu ini terdapat satu daftar yakni daftar *hero* yang akan ditampilkan statusnya.

3.4.3.8 Paket *Battlefield*

Battlefield adalah arena pertempuran antara tim *hero* melawan *monster*. Kelas *BattleField* merupakan turunan dari kelas Label yang mengimplementasikan *Runnable*, *MouseListener*, dan *ActionListener* dari paket Java. Implementasi *Runnable* digunakan untuk *thread* gambar pertempuran. Implementasi *MouseListener* digunakan untuk pemilihan target dalam pertempuran dengan klik mouse. Implementasi *ActionListener* digunakan untuk menjalankan aksi dari tombol-tombol pada panel tempur.

Pada paket *Battlefield* terdapat dua bagian penting yakni Battle*Thread* dan *FrmAction*.

Gambar 3.36 Class Diagram : *battlefield* / Package Overview

Battle*Thread* adalah *thread* untuk menyiapkan aksi *hero* atau *monster* berdasarkan urutan pada antrian tempur. Urutan pada antrian tempur ditentukan oleh *speed* dari *hero* atau *monster*.

FrmAction adalah menu yang digunakan untuk menggunakan sihir atau item *hero* dalam pertempuran. Pada menu ini terdapat dua daftar yakni daftar sihir *hero* yang akan digunakan dalam pertempuran dan daftar item yang akan digunakan

dalam pertempuran. Terdapat pula cek pilihan yang digunakan untuk memilih sifat sihir yang akan ditampilkan pada daftar sihir.

Metode-metode penting yang terdapat pada kelas *BattleField* akan dijelaskan di bawah.

3.4.3.8.1 Metode *doBattle()*

Metode ini digunakan untuk mempersiapkan Battle*Thread* dan *thread* masing-masing *hero* serta *monster* sebagai persiapan pertempuran

3.4.3.8.2 Metode *drawHero()*

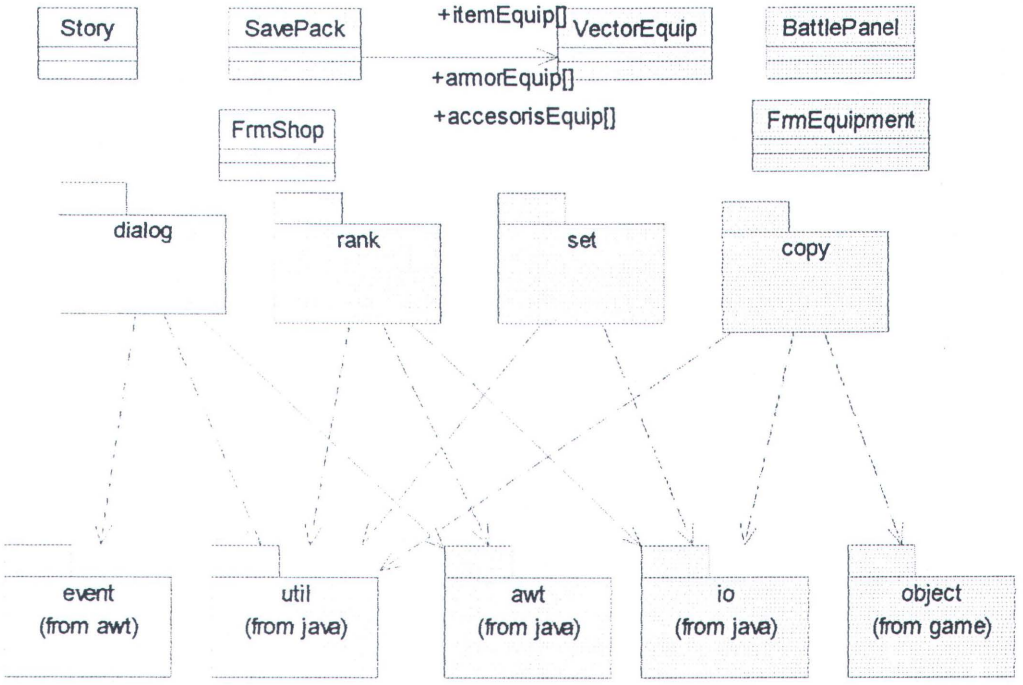
Metode ini digunakan untuk mempersiapkan gambar *hero* yang akan digambar oleh *thread* gambar berdasarkan aksi dari *hero* tersebut.

3.4.3.8.3 Metode *drawMonster()*

Metode ini digunakan untuk mempersiapkan gambar *monster* yang akan digambar oleh *thread* gambar berdasarkan aksi dari *monster* tersebut.

3.4.4 Paket Tool

Tool adalah sarana yang digunakan dalam game untuk berbagai keperluan di dalamnya. Terdapat enam kelas serta empat paket khusus. Kelas-kelas tersebut adalah *BattlePanel*, FrmShop, Frm*Equipment*, *Story*, *VectorEquip*, dan *SavePack*. Sedangkan keempat paket tersebut adalah *dialog*, *copy*, *rank*, dan *set*.



Gambar 3.37 Class Diagram : tool / Package Overview

Semua kelas serta paket di atas akan dijelaskan di bawah ini.

3.4.4.1 Kelas *BattlePanel*

BattlePanel adalah panel tempur yang digunakan pemain untuk menentukan aksi *hero* dalam pertempuran. Kelas ini merupakan turunan dari kelas Panel dari paket Java. Pada panel tempur ini terdapat tombol *"attack"*, *"magic"*, *"item"*, *"guard"*, *"run"*, dan *"SP"*.

3.4.4.2 Kelas FrmShop

FrmShop adalah bingkai yang digunakan untuk menampilkan menu belanja dari tiap toko. Kelas ini merupakan turunan dari kelas Frame dari paket Java.

3.4.4.3 Kelas Frm*Equipment*

FrmShop adalah bingkai yang digunakan untuk menampilkan menu *equipment*. Kelas ini merupakan turunan dari kelas Frame dari paket Java.

3.4.4.4 Kelas *Story*

Story adalah sarana yang digunakan untuk menampilkan cerita game. Kelas ini merupakan turunan dari kelas Label dari paket Java. Cerita game ditampilkan dalam gambar dan dialog.

3.4.4.5 Kelas *VectorEquip*

VectorEquip adalah sarana yang digunakan sebagai tempat untuk meletakkan perlengkapan serta item pada perbekalan *hero*. Kelas ini merupakan turunan dari kelas Vector dari paket Java. Perlengkapan serta item diletakkan pada *VectorEquip* berdasarkan index yang dimiliki oleh masing-masing tipe perlengkapan atau item tersebut.

3.4.4.6 Kelas *SavePack*

SavePack adalah obyek yang berisi segala informasi tentang tim *hero*, perbekalan perlengkapannya, serta *state* tempat pemain terakhir menyimpan status permainannya. Obyek ini digunakan pada proses save game dan load game.

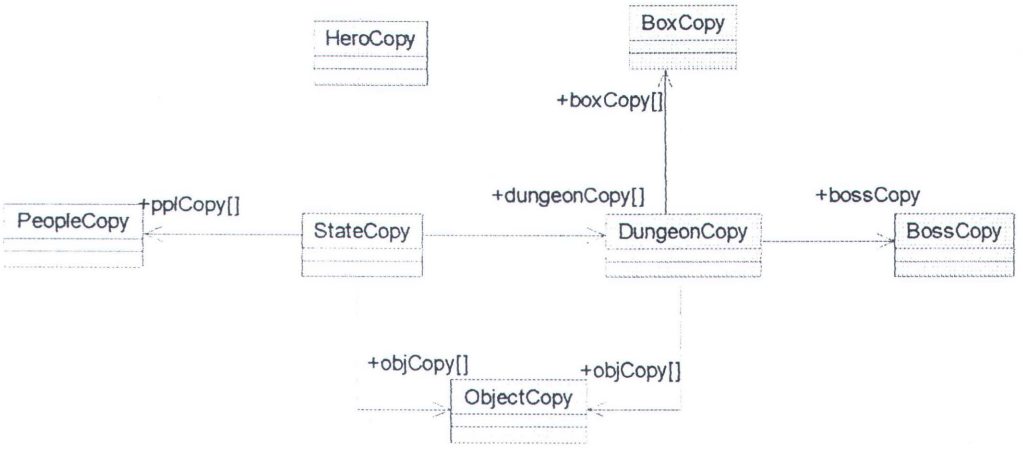
3.4.4.7 Paket *Dialog*

Paket ini berisi sarana yang diperlukan untuk menampilkan dialog percakapan dalam game. Terdapat dua kelas dalam paket dialog yakni *DlgPlace* dan *FrmDialog*.

DlgPlace digunakan untuk mengatur isi serta tampilan teks yang akan ditampilkan pada dialog. *FrmDialog* adalah bingkai yang digunakan untuk menampilkan dialog.

3.4.4.8 Paket *Copy*

Paket ini berisi sarana yang diperlukan untuk membuat salinan *hero* serta seluruh informasi yang ada dalam *state*. Salinan ini nantinya digunakan dalam obyek *SavePack*. Terdapat tujuh kelas dalam paket copy yakni *HeroCopy*, *StateCopy*, *DungeonCopy*, *PeopleCopy*, *BoxCopy*, *ObjectCopy*, dan *BossCopy*.



Gambar 3.38 Class Diagram : copy / Package Overview

HeroCopy merupakan salinan dari *hero*. *StateCopy* merupakan salinan *state*. *DungeonCopy* merupakan salinan dari *subDungeon*. *BoxCopy* merupakan salinan dari kotak barang. *ObjectCopy* merupakan salinan dari obyek *town* atau *dungeon*. *BossCopy* merupakan salinan dari bos *monster*.

3.4.4.9 Paket *Rank*

Paket ini berisi sarana yang diperlukan untuk melihat ranking dari seluruh pemain game ini yang telah terdaftar pada basis data server game. Urutan ranking

didasarkan pada status dari tim hero yang dimiliki oleh pemain yang telah disimpan di server game.

Terdapat dua kelas dalam paket rank ini yakni *FrmRank* dan *Ranking*. *FrmRank* digunakan sebagai bingkai untuk menampilkan daftar ranking dari seluruh pemain. *Ranking* merupakan obyek yang menyimpan informasi tim hero pemain yang nantinya digunakan sebagai acuan urutan ranking.

3.4.4.10 Paket *Set*

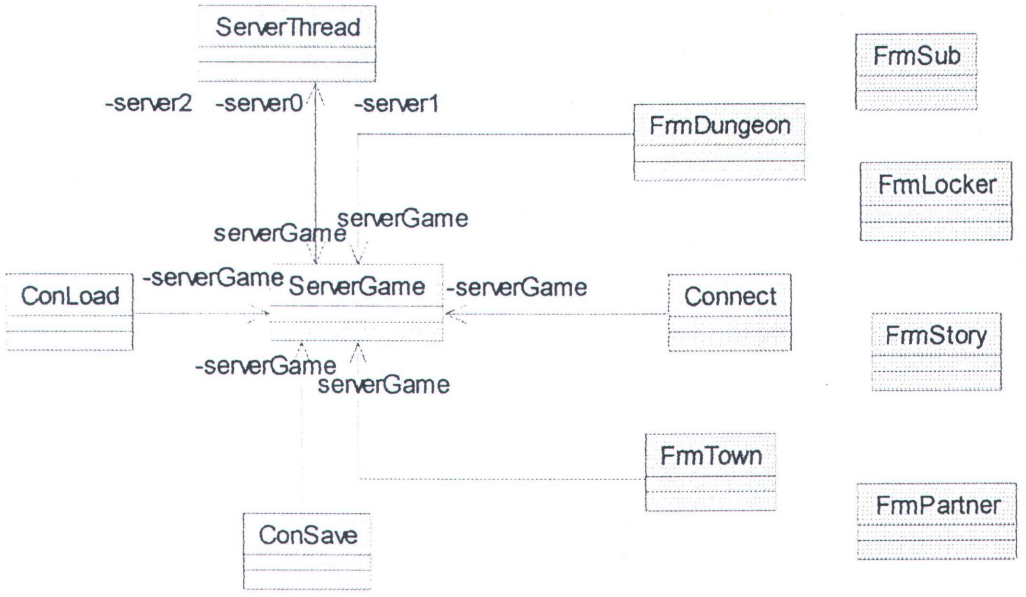
Paket ini berisi sarana yang diperlukan untuk menyimpan informasi rancangan pengaturan segala aspek yang ada dalam state

Terdapat lima kelas dalam paket set ini yang diperlukan dalam pengaturan rancangan *state*. *SetState* digunakan untuk menyimpan informasi pengaturan segala aspek dalam *state* baik sebagai *town* atau *dungeon*. *SetSub* digunakan untuk menyimpan informasi pengaturan *subDungeon* yang ada pada *dungeon*. *SetPartner* digunakan untuk menyimpan informasi dialog pembicaraan yang terjadi serta tipe hero partner pada partner baru yang dijumpai di *town*. *SetLocker* digunakan untuk menyimpan informasi dialog pembicaraan yang terjadi ketika berjumpa seorang people yang merupakan pemegang kunci *state* di *town*. *SetStory* digunakan untuk menyimpan informasi dialog pembicaraan pada akhir pertarungan melawan bos monster.

3.5 Perancangan Server Game

Server game berfungsi untuk melayani proses login game, permintaan informasi *state*, serta penyimpanan dan load status permainan. Bagian ini

merupakan aplikasi yang berjalan pada sisi server. Kelas *ServerGame* sendiri merupakan turunan dari kelas Frame dari paket Java.



Gambar 3.39 Class Diagram : servergame / Package Overview

Pada server game terdapat empat kelas yang merupakan turunan dari kelas *Thread* dari paket Java yakni Server*Thread*, *Connect*, *ConSave*, dan *ConLoad*, enam kelas yang berupa turunan kelas Frame dari paket Java yakni Frm*Town*, Frm*Dungeon*, FrmSub, FrmPartner, FrmLocker, dan FrmStory serta sebuah metode *getRank()*. Masing-masing dari kelas tersebut akan dijelaskan di bawah.

3.5.1 Kelas *ServerThread*

Kelas ini merupakan *thread* yang digunakan untuk mendengar permintaan dari klien game berdasarkan port *socket*-nya. Bila menerima *socket* pada port 9999 maka akan diciptakan *thread* Connect. Bila menerima *socket* pada port 10000 maka akan diciptakan *thread* ConSave. Bila menerima *socket* pada port 10001 maka akan diciptakan *thread* ConLoad.

3.5.2 Kelas *Connect*

Kelas ini merupakan *thread* yang digunakan untuk melayani proses login game.

Pada proses login game, login dan password yang diterima oleh server akan dibandingkan dengan basis data server. Konfirmasi ditemukannya atau tidak login tersebut pada basis data akan dikirimkan kembali kepada klien game.

3.5.3 Kelas *ConSave*

Kelas ini merupakan *thread* yang digunakan untuk melayani proses penyimpanan status permainan klien game. Ketika server mendapat kiriman obyek *SavePack* maka obyek tersebut disimpan pada file di harddisk server dengan nama sesuai dengan loginnya.

3.5.4 Kelas *ConLoad*

Kelas ini merupakan *thread* yang digunakan untuk melayani proses load status permainan klien game, proses mendapatkan urutan ranking pemain, serta melayani permintaan informasi *state* game.

Ketika server mendapat permintaan load, maka dikirimkanlah obyek *SavePack* yang diperoleh dari file pada harddisk server dengan nama sesuai loginnya kepada klien game tersebut.

Jika server mendapat permintaan rank, maka dikirimkanlah obyek yang berisi urutan ranking pemain kepada klien peminta.

Pada proses permintaan informasi *state* game, server akan membaca file *"vState.sta"* yang merupakan file yang berisi informasi seluruh *state* yang ada pada game. Kemudian hasil pembacaan tersebut dikirimkan kepada klien game.

3.5.5 Kelas *FrmTown*

Kelas ini merupakan frame antarmuka yang digunakan untuk mengatur segala aspek yang ada dalam *town*.

3.5.6 Kelas *FrmDungeon*

Kelas ini merupakan frame antarmuka yang digunakan untuk mengatur segala aspek yang ada dalam *dungeon*.

3.5.7 Kelas *FrmSub*

Kelas ini merupakan frame antarmuka yang digunakan untuk mengatur segala aspek yang ada dalam *subDungeon* pada *dungeon*.

3.5.8 Kelas *FrmPartner*

Kelas ini merupakan frame antarmuka yang digunakan untuk mengatur dialog pembicaraan yang terjadi serta tipe hero partner pada partner baru yang dijumpai di *town*.

3.5.9 Kelas *FrmLocker*

Kelas ini merupakan frame antarmuka yang digunakan untuk mengatur dialog pembicaraan yang terjadi ketika berjumpa seorang people yang merupakan pemegang kunci *state* di *town*.

3.5.10 Kelas *FrmStory*

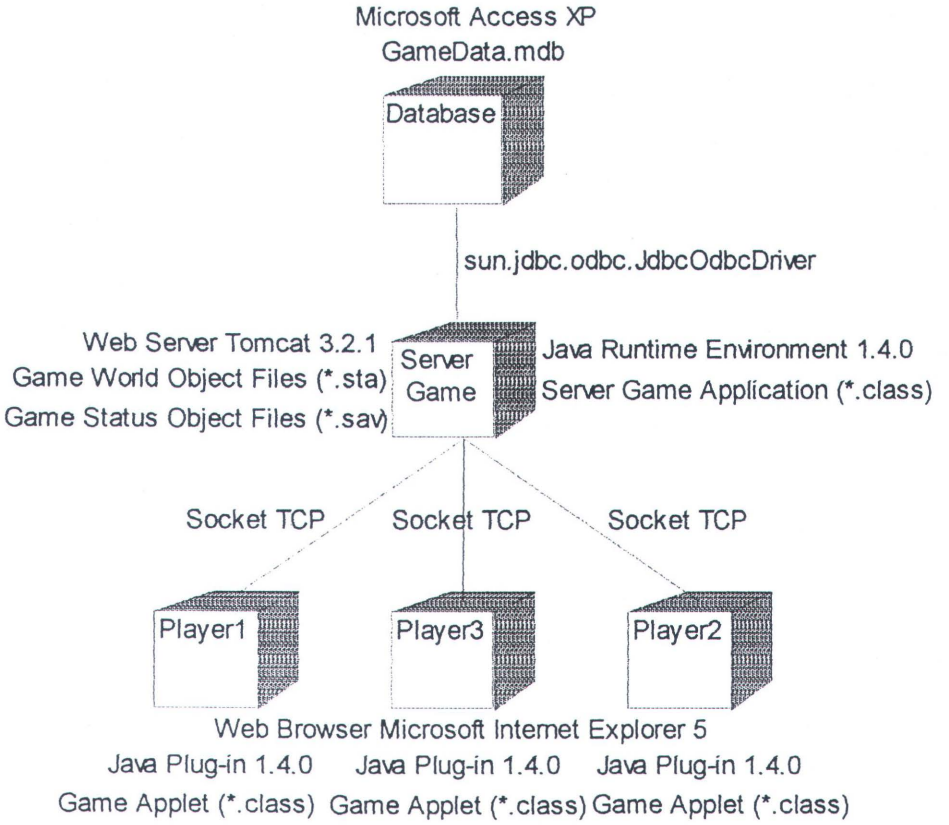
Kelas ini merupakan frame antarmuka yang digunakan untuk mengatur dialog pembicaraan pada akhir pertarungan melawan bos monster.

3.5.11 Metode *getRank()*

Metode ini digunakan untuk mendapatkan obyek yang berisi urutan ranking seluruh pemain yang telah terdaftar pada basis data server game.

3.6 Perancangan *Deployment*

Berdasarkan seluruh hasil rancangan di atas maka untuk menggambarkan sistem ini dibutuhkan lima prosesor. Berikut ini adalah Deployment Diagram untuk sistem *Role Playing Game* berbasis web ini.



Gambar 3.40 Deployment Diagram

object	structure	system				tool				
			battlefield	equipment	inn		copy	dialog	rank	set
Accesoris	Boss	AccesorisShop	BattleField	Equip	Inn	BattlePanel	BossCopy	DlgPlace	FrmRank	SetLocker
Armor	Dungeon	ArmorShop	BattleThread	EquipItem	LabelInn	FrmEquipment	BoxCopy	FrmDialog	Ranking	SetPartner
EquipUp	Game	BlackSmith	FrmAction	EquipMagic		FrmShop	DungeonCopy			SetState
Hero	State	ItemShop		Equipment		SavePack	HeroCopy			SetStory
Item	SubDungeon	WeaponShop		EquipStatus		Story	ObjectCopy			SetSub
ItemBox	Town			HeroSelect		VectorEquip	PeopleCopy			
Magic							StateCopy			
Monster										
ObjectMap										
People										
Weapon										

Gambar 3.41 Tabel Kelas pada Applet Game

server	editor
ConLoad	FrmDungeon
Connect	FrmLocker
ConSave	FrmPartner
ServerGame	FrmStory
ServerThread	FrmSub
	FrmTown

Gambar 3.42 Tabel Kelas pada Aplikasi Server Game

Satu prosesor digunakan sebagai server game. Pada prosesor tersebut terdapat Java Runtime Environment 1.4.0 yang digunakan untuk menjalankan aplikasi ServerGame. Pada prosesor ini juga terdapat web server Tomcat 3.2.1 yang digunakan untuk mempublikasikan folder tempat kelas-kelas aplikasi applet Game berada.

Satu prosesor yang lain digunakan sebagai basis data game dalam hal ini digunakan Microsoft Access XP dengan file GameData.mdb. Komunikasi antara prosesor server game dan prosesor basis data dihubungkan melalui JdbcOdbcDriver.

Tiga prosesor digunakan untuk mewakili tiga pemain. Pada masing-masing prosesor tersebut terdapat web browser Microsoft Internet Explorer 5 yang telah ter-install Java Plug-in 1.4.0 yang digunakan untuk menjalankan applet Game. Komunikasi antara prosesor server game dengan prosesor para pemain dihubungkan melalui *socket TCP*.

BAB IV

IMPLEMENTASI PERANGKAT LUNAK

BAB IV

IMPLEMENTASI PERANGKAT LUNAK

Pada Bab 3 telah dilakukan perancangan sistem dari aplikasi dengan menggunakan notasi UML pada Rational Rose. Selanjutnya pada bab ini akan dilakukan implementasi dari sistem tersebut. Seluruh aplikasi yang ada dibawah ini dibangun dengan Java 2 SDK 1.4.0.

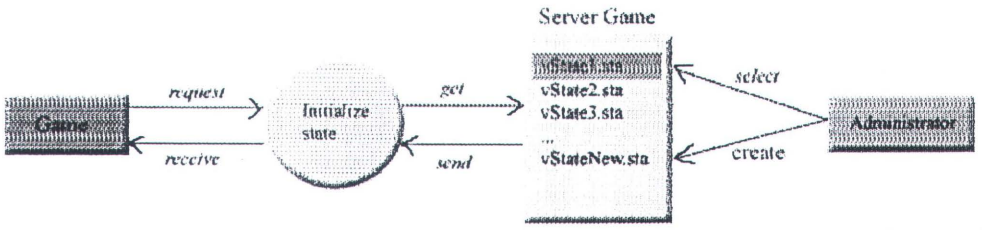
4.1 Pembuatan Aplikasi Game

Aplikasi Game merupakan aplikasi yang digunakan oleh pemain untuk bermain game. Aplikasi ini merupakan turunan kelas Applet dari paket Java oleh karena itu game ini dijalankan melalui web browser.

Berikut di bawah ini akan dijelaskan proses-proses utama yang terjadi dalam aplikasi game.

4.1.1 Inisialisasi Dunia Permainan

Sebelum memulai permainan, seluruh state dalam dunia permainan termasuk segala aspek yang ada di dalamnya baik state tersebut berupa *town* atau *dungeon* akan diinisialisasikan terlebih dahulu. Inisialisasi diperoleh aplikasi game dari server game dalam bentuk obyek vector yang sebelumnya dikirimkan melalui *socket* port 10001.



Gambar 4.1 **Gambar 4.1 Proses Inisialisasi State**

Proses permintaan inisilisasi state dilakukan dengan menyertakan pula login beserta password pemain sehingga hanya pemain yang telah terdaftar pada basis data server game yang bisa menginisialisasi seluruh state pada game. Berikut ini adalah pernyataan yang ditulis untuk melakukan permintaan inisialisasi state kepada server game.

```
try{
      socket=new Socket(getDocumentBase().getHost(),10001);
      ObjectInput objIn=new ObjectInputStream(socket.getInputStream());
      request=new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));
      request.write("state "+tLogin.getText()+" "+tPass.getText());
      request.newLine();
      request.flush();
      vState=(Vector)objIn.readObject();
      objIn.close();
      request.close();
      socket.close();
}
catch(Exception ioe){
      System.out.println(e.toString());
}
```

Mula-mula *socket* diinisialisasi dengan host tempat server game berada dan port 10001. Kemudian diinisialisikan pula ObjectInput yang digunakan untuk menerima obyek kiriman dari server beserta BufferedWriter yang digunakan untuk menulis string permintaan. String permintaan yang ditulis adalah "state login password". Obyek vector kiriman dari server game diletakkan pada obyek vState. Object vector itu sendiri berisi obyek-obyek SetState.

Kemudian obyek vState tersebut digunakan untuk menginisialisasi seluruh state game pada saat memulai permainan baru atau melanjutkan permainan.

```
state=new State[vState.size()];
for(int i=0;i<state.length;i++){
      state[i]=new State((SetState)vState.elementAt(i),i,this);
```

```
        state[i].setLocation(xLegend+state[i].px,yLegend+state[i].py);
        if(i>0){
            state[i].setPrev(state[i-1]);
            state[i-1].setNext(state[i]);
        }
        if(state[i].show){
            add(state[i]);
        }
}
```

State-state tersebut kemudian diurutkan berdasarkan urutannya dalam vector dan state yang memiliki status show = true akan ditampilkan pada peta dunia game. Masing-masing dari state tersebut diinisialisasi sebagai *town* atau *dungeon* tergantung dari status is*Town* yang dimilikinya.

```
if(setState.isTown){
    isTown=true;
    town=new Town(this);
}
else{
    isTown=false;
    dungeon=new Dungeon(this);
}
```

Obyek-obyek SetState yang ada dalam vState tersebut digunakan untuk menginisialisasi *town*, *dungeon*, sub*Dungeon*, pertempuran, seluruh toko, serta blackSmith.

4.1.2 Penjelajahan Area

Penjelajahan Hero dilakukan di area *town* atau *dungeon*. Pada *town* atau *dungeon* itu sendiri telah diinisialisasikan suatu peta area yang akan menentukan pergerakan hero dalam area tersebut.

Pemetaan *area* tersebut menggunakan metode 2D *tile based* engine yakni membagi *area* menjadi petak-petak dengan ukuran pixel tertentu yang berisi suatu nilai yang mewakili kondisi tertentu. Peta *area* tersebut dibentuk dalam format array 2D. Berikut ini keterangan dari tiap nilai tersebut :

0 : Area yang tidak bisa dilalui *hero*

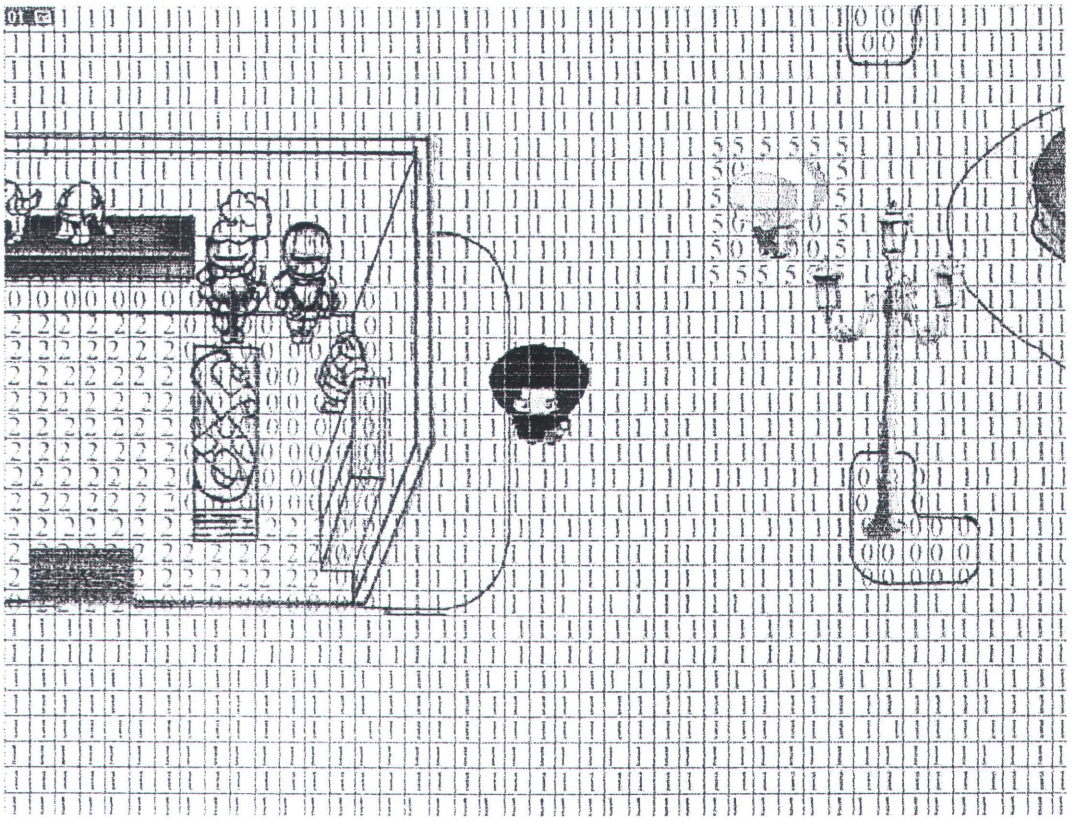
1 : Area yang bisa dilalui *hero* yang bersifat netral

2 s/d (total obyek+1) : Area dalam obyek, tiap nilai mewakili tiap obyek

(total obyek+ 1) s/d (total obyek+ 1 + total *people*) : Area reaksi pekerjaan dari *people*

(total obyek+ 2 + total *people*) : Pintu gerbang keluar *area*

Pemetaan *area* diatas didasarkan pada letak obyek-obyek *area* yang ada di dalamnya. Nilai 0 diletakkan pada dinding rumah, tanaman, pohon, tiang, atau benda-benda yang tidak mungkin dilalui *hero*. Nilai diatas 0 bisa dilalui *hero* namun masing-masing nilai memiliki karakteristik seperti yang dijelaskan pada keterangan di atas. Nilai 1 diletakkan pada jalan atau segala sesuatu yang bisa dilalui *hero* namun tidak mempunyai efek apapun. Nilai 2 s/d (total obyek+1) yang berupa rumah diletakkan pada area bagian dalam rumah. Tiap nilai di atas mewakili tiap-tiap rumah jadi jika *hero* melewatinya akan menghilangkan atap rumah yang diwakilinya. Nilai (total obyek+ 1) s/d (total obyek+ 1 + total *people*) diletakkan pada area di sekitar *people* yang tiap nilainya juga mewakili tiap *people* tersebut. Jika *hero* berada pada area tersebut kemudian pemain menekan key "enter" pada keyboard maka akan dijalankan metode setJob() dari *people* yang diwakili nilai tersebut. Nilai (total obyek+ 2 + total *people*) diletakkan pada pintu gerbang keluar *area* jadi jika *hero* melewatinya maka ia akan keluar dari *area* tersebut.



Gambar 4.2 Contoh pemetaan *area*

Thread gambar yang memanggil metode paint() akan aktif ketika pemain menekan key arah pada keyboard. Dalam metode paint() tersebut metode drawHero() akan dijalankan. Seperti telah dijelaskan sebelumnya bahwa metode drawHero() pada *town* atau *dungeon* digunakan untuk mempersiapkan gambar *hero* yang akan digambar serta mengubah koordinat dari gambar wilayah area serta seluruh obyek dalam area berdasarkan perubahan posisi *hero*. Perubahan posisi *hero* ditentukan oleh key arah pada keyboard yang ditekan oleh pemain. Jadi pergeseran gambar wilayah area beserta seluruh koordinat obyek yang ada di dalamnya ditentukan oleh arah yang ditekan pemain. Berikut ini penggalan pernyataan dalam metode drawHero() untuk arah atas.

```
if(yHeroMap-1>0){
      for(int x=xHeroMap;x<=xHeroMap+hWidth;x++){
            if(map[x/5][(yHeroMap-1)/5]==0){
                  walk=false;
            }
      }
      if(walk==true){
```

```
            yHeroMap-=1;
            yMap+=1;
            for(int i=0;i<objPack.size();i++){
      if(map[xHeroMap/5][(yHeroMap+hHeight)/5]==1&&map[(xHeroMap+hWidth)/5][(yHeroMa
p+hHeight)/5]==1){
                        object[i].show=true;
                  }
                  object[i].yPanel+=1;
      if(map[xHeroMap/5][yHeroMap/5]==i+2&&map[(xHeroMap+hWidth)/5][yHeroMap/5]==i+2
){
                        object[i].show=false;
                  }
            }
            for(int i=0;i<pplPack.size();i++){
                  people[i].yPanel+=1;
            }
      if(map[xHeroMap/5][(yHeroMap1)/5]==objPack.size()+2+pplPack.size()&&map[(xHero
Map+hWidth)/5][(yHeroMap-1)/5]==objPack.size()+2+pplPack.size()){
                  drawSuspended=true;
                  onWalk=false;
                  cntUp=0;
                  faceHero=1;
                  state.game.remove(this);
            }
      }
}
```

Bisa dilihat diatas bahwa untuk bisa bergerak posisi hero harus pada area yang memiliki nilai peta > 0. Arah atas menyebabkan seluruh ordinat obyek harus ditambah 1. Bisa dilihat pula hero akan keluar dari area bila melalui nilai (total obyek+ 2 + total *people*).

Akibat dari perubahan koordinat di atas menyebabkan perubahan letak gambar dari wilayah area serta seluruh obyek. Berikut ini adalah pernyataan yang ada dalam metode paint() *town*.

```
public void paint(Graphics g){
      drawHero();
      setFocusable(true);
      requestFocus();
      if(offscreenImage==null){
            offscreenImage=createImage(getWidth(),getHeight());
            offscreenGraphics=offscreenImage.getGraphics();
      }
      if(!tracker.checkAll()){
            System.out.println("loading image...");
      }
      else{
            offscreenGraphics.setColor(Color.black);
            offscreenGraphics.fillRect(0,0,getWidth(),getHeight());
            offscreenGraphics.drawImage(imgMap,xMap,yMap,this);
            offscreenGraphics.drawImage(imgHero,xHero,yHero,this);
            for(int i=0;i<people.length;i++){
                  if(people[i].show==true){
      offscreenGraphics.drawImage(people[i].imgActive,people[i].xPanel,people[i].yPa
nel,this);
                  }
            }
            for(int i=object.length-1;i>=0;i--){
                  if(object[i].show==true){
      offscreenGraphics.drawImage(object[i].image,object[i].xPanel,object[i].yPanel,
this);
```

```
                }
            }
            g.drawImage(offscreenImage,0,0,this);
        }
}
```

Sebelum memulai menggambar, diciptakanlah terlebih dahulu obyek offscreenGraphics yang merupakan kanvas di balik layar yang digunakan untuk menggambar seluruh obyek yang ada dalam *town* beserta image background *town*. Cara ini merupakan implementasi dari metode double buffering. Sebelumnya pula mediaTracker mencek seluruh image yang akan digambar apakah telah seluruhnya ter-load dengan sempurna. Setelah seluruh image obyek digambar berdasarkan hasil koordinat yang diperoleh dari metode drawHero(), kanvas yang ada di balik layar tersebut ditampilkan pada layar yang sesungguhnya.

4.1.3 Sistem Transaksi

Transaksi yang dilakukan hero di *town* antara lain dijelaskan sebagai berikut.

4.1.3.1 Jual-Beli Perlengkapan

Ada empat macam toko yakni *WeaponShop*(toko senjata), *ArmorShop*(toko baju), *AccesorisShop*(toko aksesoris), dan *ItemShop*(toko item). Ketika form belanja dari toko tersebut telah muncul maka ditampilkan daftar perlengkapan yang ditawarkan oleh penjual yang akan dibeli oleh pemain, daftar perlengkapan yang ingin dijual oleh pemain, dan daftar *hero* yang bisa mengenakan perlengkapan tersebut.

Untuk perlengkapan senjata dan baju, sebelum membelinya pemain harus membandingkan terlebih dahulu status yang akan berubah pada hero bila mengenakannya. Status hero yang berpengaruh tersebut meliputi attack, defend,

agility, dan speed. Penggalan kode untuk membandingkan status hero adalah sebagai berikut

```
tmpHero.setWeapon(weapon);
int range;
String attStatus;
String defStatus;
String spdStatus;
String aglStatus;
if(tmpHero.attack>=activeHero.attack){
      range=tmpHero.attack-activeHero.attack;
      attStatus="Attack : + "+range;
}
else{
      range=activeHero.attack-tmpHero.attack;
      attStatus="Attack : - "+range;
}
if(tmpHero.defend>=activeHero.defend){
      range=tmpHero.defend-activeHero.defend;
      defStatus="Defend : + "+range;
}
else{
      range=activeHero.defend-tmpHero.defend;
      defStatus="Defend : - "+range;
}
if(tmpHero.speed>=activeHero.speed){
      range=tmpHero.speed-activeHero.speed;
      spdStatus="Speed : + "+range;
}
else{
      range=activeHero.speed-tmpHero.speed;
      spdStatus="Speed : - "+range;
}
if(tmpHero.agility>=activeHero.agility){
      range=tmpHero.agility-activeHero.agility;
      aglStatus="Agility : + "+range;
}
else{
      range=activeHero.agility-tmpHero.agility;
      aglStatus="Agility : - "+range;
}
status.setText(attStatus+"\n"+defStatus+"\n"+spdStatus+"\n"+aglStatus);
```

Seperti terlihat di atas bahwa pembandingan status hero dilakukan dengan memasang perlengkapan tersebut pada hero sementara terlebih dahulu. Kemudian status hero sementara dibandingkan dengan hero sesungguhnya.

Setiap perlengkapan memiliki harga beli dan harga jual. Pemain bisa membeli perlengkapan bila uang yang ada melebihi harga beli perlengkapan tersebut.

```
public void buyWeapon(Weapon w){
      if(weaponEquip[w.index].size()<98){
            if(money>=w.buyPrice){
                  money-=w.buyPrice;
                  weaponEquip[w.index].addElement(w);
            }
            else{
                  System.out.println("Your money is not enough");
            }
      }
```

```
}
```

Begitu pula bila pemain menjual perlengkapan maka ia akan mendapat uang sebesar harga jual perlengkapan tersebut.

```
public void sellWeapon(int index){
      Weapon tmpWeapon=new Weapon();
      if(weaponEquip[index].size()>0){
            tmpWeapon=(Weapon)weaponEquip[index].firstElement();
            weaponEquip[index].removeElementAt(0);
            money+=tmpWeapon.sellPrice;
      }
      else{
            System.out.println("No weapon found");
      }
}
```

4.1.3.2 Upgrade Senjata

Upgrade senjata dilakukan di blacksmith(pandai besi). Setiap senjata hasil upgrade membutuhkan dua macam perlengkapan yakni senjata dan aksesoris. Untuk setiap senjata hasil upgrade memiliki kombinasi kebutuhan senjata dan aksesoris yang bisa diatur.

```
EquipUp equipUp=new EquipUp(new Weapon(result[0],town.state.game));
equipUp.setRequire(new Weapon(result[1],town.state.game));
equipUp.setRequire(new Accesoris(result[2]));
addUpgrade(equipUp);
```

Pemain harus memiliki senjata dan aksesoris apa saja yang dibutuhkan beserta biayanya untuk mendapatkan senjata hasil upgrade tersebut.

4.1.3.3 Penginapan

Penginapan berfungsi untuk memulihkan seluruh status yang ada pada tim hero. Pernyataan di bawah menggambarkan kondisi ini.

```
inn.town.state.game.money-=10;
      for(int i=0;i<inn.town.state.game.heroPack.size();i++){
            Hero hero=(Hero)inn.town.state.game.heroPack.elementAt(i);
            hero.hp=hero.maxHp;
            hero.mp=hero.maxMp;
            hero.poison=false;
            hero.berserk=false;
            hero.blind=false;
            if(hero.slow){
                  hero.slow=false;
                  hero.waitTime=hero.tmpWait;
                  }
            }
```

Seperti terlihat di atas bahwa hero yang diistirahatkan di penginapan membuat hp dan mp menjadi penuh serta seluruh status tidak normal menjadi hilang.

4.1.4 Sistem Pengaturan Hero

Untuk masuk ke menu pengaturan perlengkapan tim hero dilakukan dengan menekan tombol "esc" pada keyboard.

4.1.4.1 Status Hero

Hero memiliki status antara lain *hp*(*health* point), *mp*(*magic* point), *maxHp*, *maxMp*, *level*, *attack*, *defend*, *speed*, *agility*, *point*, serta *nextPoint*.

Hp adalah nilai yang menyatakan jumlah darah *hero*. Jika mp habis maka *hero* tersebut mati sehingga tidak bisa melakukan apa-apa. MaxHp adalah hp maksimum yang bisa diperoleh *hero* saat itu.

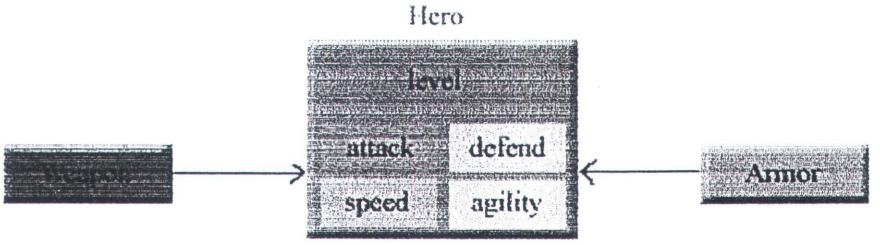
Mp adalah nilai yang menyatakan jumlah kemampuan sihir *hero*. Jika mp maka *hero* tersebut tidak bisa menggunakan sihirnya. MaxMp adalah mp maksimum yang bisa diperoleh *hero* saat itu.

Attack adalah nilai yang menyatakan daya rusak yang bisa dilakukan *hero* terhadap lawannya. Semakin tinggi nilai *attack hero* maka daya rusaknya terhadap *monster* yang diserangnya akan makin besar pula.

Defend adalah nilai yang menyatakan daya tahan *hero* terhadap serangan lawan. Semakin tinggi nilai *defend hero* maka kerusakan yang diperoleh *hero* ketika diserang *monster* lawan akan makin kecil.

Speed adalah nilai yang menyatakan kecepatan beraksi *hero* dalam pertempuran. Semakin tinggi nilai *speed hero* maka waktu tunggu pemain untuk mengeksekusi aksi *hero* tersebut akan makin cepat.

Agility adalah nilai yang menyatakan kelincahan *hero* dalam bertempur. Semakin tinggi nilai *agility hero* maka semakin besar kemungkinan *hero* tersebut untuk mengelak serangan *monster* lawan.



Gambar 4.3 Gambaran Status Hero

Point adalah nilai yang diperoleh *hero* setelah mengalahkan *monster*. NextPoint adalah poin yang diperlukan *hero* untuk naik level selanjutnya. Level adalah tingkatan status kemampuan *hero*. Semakin tinggi level *hero* maka maxHp, maxMp, *attack*, *defend*, *speed*, serta *agility hero* juga meningkat pula.

Hero juga memiliki status tidak normal antara lain *poison*, *blind*, *sleep*, *berserk*, dan *slow*. *Poison* adalah kondisi *hero* mengalami keracunan sehingga hp-nya akan selalu berkurang setiap beraksi dalam pertempuran. *Blind* adalah kondisi *hero* mengalami kebutaan sehingga serangannya akan selalu meleset. *Sleep* adalah kondisi *hero* mengalami ketiduran sehingga *hero* tidak bisa melakukan apa-apa dalam pertempuran. *Berserk* adalah kondisi *hero* mengamuk sehingga serangan *hero* tidak bisa dikendalikan. *Slow* adalah kondisi *hero* mengalami kelambatan sehingga reaksi *hero* dalam pertempuran menjadi dua kali lebih lambat dari kondisi normal.

Kenaikan level dari hero akan meningkatkan segala aspek status normal yang ada padanya serta pada level tertentu akan mempelajari suatu jenis sihir tertentu pula. Kenaikan level hero diperoleh bila point yang diperoleh melebihi

nextPoint-nya. Masing-masing daripada hero memiliki fungsi kenaikan status yang spesifik. Berikut di bawah ini adalah penggalan pernyataan pada metode cekLevel() hero untuk tipe swordman.

```
public void cekLevel(Applet game){
      while(point>=nextPoint){
            if(level<99){
                  level+=1;
                  String talk=name+" level naik menjadi "+level;
                  FrmDialog frmDialog=new FrmDialog(talk,people,"happy");
                  frmDialog.dialog.show();
                  if(type.equalsIgnoreCase("swordman")){
                        maxHp=(10*level)+200;
                        maxMp=level;
                        hp=maxHp;
                        mp=maxMp;
                        attack=(4*level)/5;
                        defend=(3*level)/4;
                        agility=(7*level)/10;
                        speed=(3*level)/5;
                        if(level==5){
                              Magic magic=new Magic("blind",game);
                              putMagic(magic);
                              talk=name+" telah mempelajari sihir "+magic.type;
                              frmDialog=new FrmDialog(talk,people,"happy");
                              frmDialog.dialog.show();
                        }
```

Berikut ini adalah fungsi kenaikan status level pada masing-masing tipe hero.

Swordman :

```
maxHp=(10*level)+200;
maxMp=level;
hp=maxHp;
mp=maxMp;
attack=(4*level)/5;
defend=(3*level)/4;
agility=(7*level)/10;
speed=(3*level)/5;
```

Sorcerer :

```
maxHp=((level*level)/25+6*level)+180;
maxMp=level;
hp=maxHp;
mp=maxMp;
attack=(level*level)/500+(2*level)/5;
defend=(level*level)/500+(3*level)/10;
agility=(level*level)/500+(3*level)/5;
speed=(3*(level*level))/1000+(9*level)/20;
```

Brawler :

```
maxHp=((-1*(level*level))/25+14*level)+220;
maxMp=level;
```

```
hp=maxHp;
mp=maxMp;
attack=(-3*(level*level))/1000+(21*level)/20;
defend=(-1*(level*level))/500+level;
agility=(-1*(level*level))/500+(4*level)/5;
speed=(-1*(level*level))/500+(7*level)/10;
```

Shooter :

```
maxHp=(10*level)+200;
maxMp=level;
hp=maxHp;
mp=maxMp;
attack=(7*level)/10;
defend=(3*level)/5;
agility=(3*level)/4;
speed=(4*level)/5;
```

4.1.4.2 Perlengkapan Hero

Hero dapat melengkapi dirinya dengan senjata, baju atau aksesoris yang sesuai dengan tipenya. Pemakaian senjata dan baju dapat mempengaruhi status hero dalam hal ini adalah attack, defend, speed, dan agility. Aksesoris dapat memberikan kemampuan tambahan bagi *hero*. Pernyataan pada metode di bawah ini akan menunjukkan proses tersebut.

```
public void setWeapon(Weapon w){
      weapon=w;
      attack+=weapon.plusAttack;
      defend+=weapon.plusDefend;
      speed+=weapon.plusSpeed;
      agility+=weapon.plusAgile;
      if(attack<0)attack=0;
      if(defend<0)defend=0;
      if(speed<0)speed=0;
      if(agility<0)agility=0;
}
```

Dalam memilih perlengkapan yang akan dikenakan oleh hero, pemain juga membandingkan perubahan status pada hero bila mengenakannya seperti halnya pada toko perlengkapan.

4.1.4.3 Penggunaan Sihir

Sihir yang bisa digunakan adalah sihir yang telah dipelajari oleh hero berdasarkan kenaikan levelnya. Pada menu pengaturan perlengkapan hero, sihir

yang bisa digunakan hanya sihir yang bersifat penyembuh(white magic). Sihir bisa digunakan oleh hero bila mp mencukupi untuk menggunakan sihir tersebut.

Setiap macam sihir memiliki efek yang berbeda-beda. Untuk menggunakan sihir yang bersifat penyembuh, pemain harus memilih hero yang dijadikan target sihir terlebih dahulu. Di bawah ini adalah penggalan pernyataan untuk menggunakan sihir penyembuh dengan target hero.

```
public void useMagic(Hero user,Hero target){
      if(type.equalsIgnoreCase("heal")){
            if(target.live){
                  target.effect=100;
                  target.hp+=100;
                  if(target.hp>target.maxHp){
                        target.hp=target.maxHp;
                  }
            }
      }
```

4.1.4.4 Penggunaan Item

Semua item bersifat sebagai penyembuh bila digunakan oleh hero. Setiap tipe item memiliki efek yang berbeda-beda. Untuk menggunakan item, pemain terlebih dahulu harus memilih hero yang akan menggunakan item tersebut. Berikut ini adalah penggalan pernyataan penggunaan item.

```
public void useItem(Hero hero){
      if(type.equalsIgnoreCase("potion")){
            if(hero.live){
                  hero.effect=100;
                  hero.hp+=100;
                  if(hero.hp>hero.maxHp){
                        hero.hp=hero.maxHp;
                  }
            }
      }
```

4.1.5 Sistem Pertempuran

Susunan monster yang akan dilawan oleh tim hero ditentukan oleh hasil yang diperoleh dari nilai acak yang mewakili ID daripada susunan monster tersebut. Susunan monster disimpan pada sebuah array dari hasil pengaturan *dungeon* yang

diperoleh dari obyek SetState kiriman dari server. Berikut ini adalah pernyataan inisialisasi susunan monster dalam pertempuran.

```
String[] type=(String[])dungeon.state.setState.vMonster.elementAt(idBattle);
monster=new Monster[type.length];
for(int i=0;i<monster.length;i++){
      monster[i]=new Monster(type[i],dungeon.state.game);
      monster[i].monsThread=new Thread(monster[i]);
      monster[i].setBattle(this,i);
}
```

Pertempuran diakhiri bila semua monster yang ada berhasil dikalahkan atau semua hero dalam tim tempur tewas.

4.1.5.1 Proses Antrian

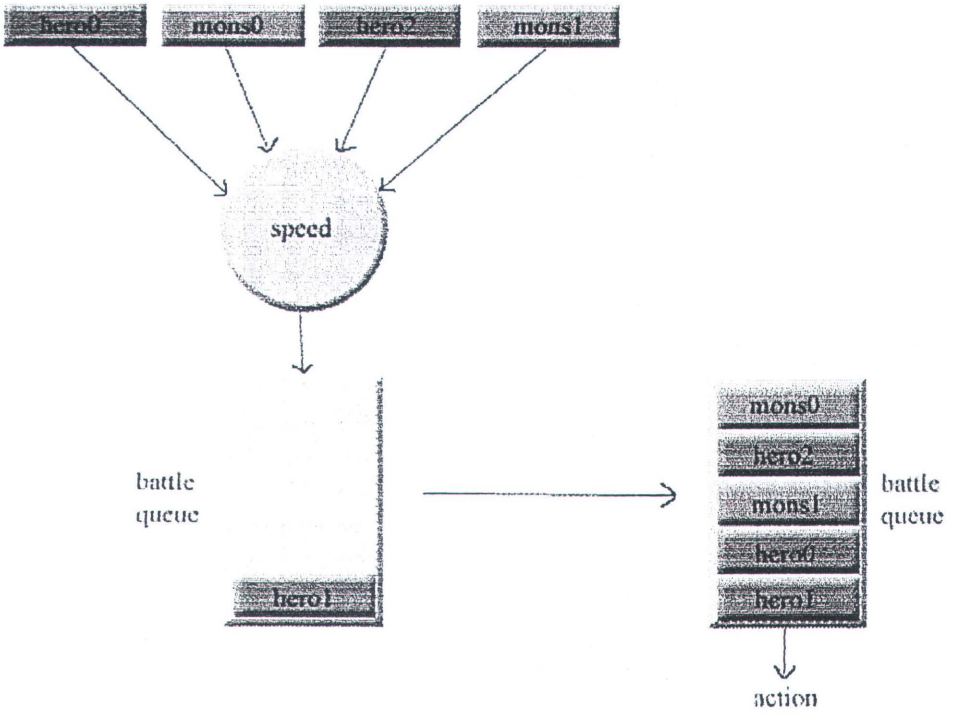
Urutan siapa yang akan melakukan aksi terlebih dahulu dalam pertempuran ditentukan oleh urutannya dalam antrian tempur(vector battleQueue). Baik hero maupun monster memiliki status speed yang menentukan lama waktu tunggu thread untuk masuk ke dalam elemen vector battleQueue. Berikut ini adalah pernyataan pada metode run() pada hero.

```
public void run(){
      Thread thisThread=Thread.currentThread();
      while(heroThread==thisThread){
            try{
                  heroThread.sleep(waitTime-(waitTime*speed/100));
            }catch(Exception e){};
            String nameQueue="hero"+idQueue;
            battleQueue.addElement(nameQueue);
            heroSuspend();
            try{
                  synchronized(this) {
                        while (heroSuspended&&heroThread==thisThread){
                              wait();
                        }
                  }
            }
            catch (InterruptedException e){break;}
      }
}
```

BattleThread menunggu untuk berjalan kembali sampai total isi vector battleQueue telah mencapai total hero dan monster yang masih hidup dalam pertempuran. Battle*Thread* sendiri adalah *thread* untuk menyiapkan aksi *hero* atau

monster berdasarkan urutan pada antrian tempur. Pengecekan total isi vector battleQueue dilakukan di dalam thread gambar battleField.

```
public void run(){
      Thread thisThread=Thread.currentThread();
      while(draw==thisThread){
            try{
                  draw.sleep(100);
            }
            catch (InterruptedException e){break;}
            repaint();
            if(battle.queueSuspended&&battleQueue.size()>=totalLive){
                  battle.queueResume();
            }
      }
}
```



Gambar 4.4 Proses Antrian Tempur

4.1.5.2 Aktifitas Hero

Dalam pertempuran, hero dapat melakukan serangan, menggunakan sihir, menggunakan item, bertahan, menggunakan jurus spesial, atau lari dari pertempuran. Aksi yang dilakukan oleh hero ditentukan pemain pada saat giliran hero tersebut dalam antrian tempur melalui tombol-tombol yang ada pada panel tempur(BattlePanel).

Serangan hero tidak akan selalu mengenai sasaran karena monster memiliki status agility. Bila status agility monster target besar maka kesempatan serangan hero untuk kena sasaran menjadi semakin kecil. Serangan hero tidak akan pernah sasaran bila status hero dalam keadaan buta(blind). Suatu serangan mengakibatkan pula target yang sedang tidur menjadi terjaga. Berikut ini pernyataan dalam metode doAttack() hero yang di dalamnya terdapat pula fungsi pengurangan hp monster target bila serangan hero mengenai sasaran.

```
public void doAttack(Monster target){
      if(blind){
            target.effect=0;
      }
      else{
            Random random=new Random();
            int chance=random.nextInt(100);
            if(chance<target.agility&&!target.sleep){
                  target.evade=true;
                  target.effect=0;
            }
            else{
                  target.effect=(((1000*attack)/100)-
((((1000*attack)/100)*target.defend)/100));
                  target.hp-=(((1000*attack)/100)-
((((1000*attack)/100)*target.defend)/100));
            }
      }
      target.sleep=false;
      onAttack=true;
}
```

Seperti terlihat di atas fungsi efek serangan terhadap lawan adalah sebagai berikut:

*target.effect=((((1000*attack)/100)-((((1000*attack)/100)*target.defend)/100));*

Serangan maksimum adalah 1000. Status attack merupakan persentase serangan yang bisa diperoleh. Serangan yang diperoleh masih harus dikurangi dengan serangan dikalikan dengan persentase defend lawan. Status defend merupakan persentase yang mengurangi serangan.

Dalam pertempuran, hero bisa menggunakan kedua macam sihir baik itu sihir penyembuh(white magic) maupun sihir penyerang(black magic). Selain status defend, status elemen monster juga berpengaruh dalam hal pertahanan terhadap serangan sihir. Ada elemen monster yang mengurangi efek sihir tertentu bahkan ada

yang kebal sama sekali. Berikut ini adalah penggalan pernyataan penggunaan sihir api.

```
public void useMagic(Hero user,Monster target){
      if(type.equalsIgnoreCase("fire")||type.equalsIgnoreCase("flame")||type.equalsI
gnoreCase("hellfire")){
            if(target.element.equalsIgnoreCase("fire")){
                  attack=0;
            }
            if(target.element.equalsIgnoreCase("water")){
                  attack=power+((power*3)/10);
            }
            if(target.element.equalsIgnoreCase("undead")){
                  attack=power-((power*2)/10);
            }
      }
```

Seperti terlihat di atas bahwa monster elemen api kebal terhadap sihir api. Sedangkan monster elemen air mendapat efek serangan lebih besar.

Terdapat pula sihir yang mengakibatkan status tidak normal terhadap sasaran. Sebagai contoh pernyataan berikut adalah sihir racun.

```
if(type.equalsIgnoreCase("poison")){
      attack=power;
      if(!target.boss){
            target.poison=true;
      }
}
```

Bertahan dilakukan untuk meningkatkan status defend hero menjadi dua kali lipat. Untuk hero tipe brawler, bertahan akan menambah nilai spesialnya. Berikut ini adalah pernyataan pada metode doGuard() hero.

```
public void doGuard(BattleField bField){
      defend*=2;
      onGuard=true;
}
```

Jurus spesial hero baru bisa digunakan bila nilai spesial hero telah mencapai 5. Setiap tipe hero memiliki jurus spesial yang spesifik. Berikut ini pernyataan pada metode doSpecial() hero untuk semua tipe hero.

```
public void doSpecial(BattleField bField){
      if(type.equalsIgnoreCase("swordman")||type.equalsIgnoreCase("shooter")){
            bField.monster[bField.monsTarget].effect=1000;
            bField.monster[bField.monsTarget].hp-=1000;
            bField.monster[bField.monsTarget].sleep=false;
      }
      if(type.equalsIgnoreCase("brawler")){
            for(int i=0;i<bField.monster.length;i++){
                  bField.monster[i].effect=500;
                  bField.monster[i].hp-=500;
                  bField.monster[i].sleep=false;
            }
```

```
        }
        if(type.equalsIgnoreCase("sorcerer")){
            for(int i=0;i<bField.hero.length;i++){
                if(!bField.hero[i].live){
                    bField.hero[i].live=true;
                }
                if(bField.hero[i].poison){
                    bField.hero[i].poison=false;
                }
                if(bField.hero[i].sleep){
                    bField.hero[i].sleep=false;
                }
                if(bField.hero[i].slow){
                    bField.hero[i].slow=false;
                }
                if(bField.hero[i].berserk){
                    bField.hero[i].berserk=false;
                }
                if(bField.hero[i].blind){
                    bField.hero[i].blind=false;
                }
                bField.hero[i].effect=bField.hero[i].maxHp;
                bField.hero[i].hp=bField.hero[i].maxHp;
                bField.hero[i].mp=bField.hero[i].maxMp;
            }
        }
        special=0;
        onSpecial=true;
}
```

Seperti terlihat di atas bahwa jurus spesial hero tipe swordman dan shooter menyerang pada satu target dengan efek serang 1000. Jurus spesial hero tipe brawler menyerang pada semua lawan dengan efek serang 500. Sedangkan jurus spesial hero tipe sorcerer menyembuhkan semua status tidak normal bahkan membangkitkan hero yang mati serta mengembalikan hp dan mp menjadi penuh pada semua hero dalam tim tempur.

Kesempatan untuk bisa lari dari pertempuran adalah 50:50. Bila hero gagal melarikan diri maka pertempuran masih terus berlanjut dan itu berarti ia telah melewatkan kesempatan untuk melakukan hal yang lain seperti menyerang atau menyembuhkan diri. Ketika melawan bos monster, tombol "run" tidak akan diaktifkan. Berikut ini adalah penggalan pernyataan untuk melarikan diri.

```
boolean run=random.nextBoolean();
if(run){
        draw=null;
        dungeon.state.game.bPanel.removeBattle(this);
        for(int i=0;i<hero.length;i++){
            hero[i].heroStop();
            hero[i].sleep=false;
        }
        for(int i=0;i<monster.length;i++){
            monster[i].monsStop();
        }
```

```
        dungeon.state.game.mscBattle.stop();
        dungeon.state.game.bPanel.onBattle=false;
        dungeon.state.game.bPanel.repaint();
        dungeon.state.game.remove(this);
        dungeon.state.game.add(dungeon);
        dungeon.state.game.mscDungeon.loop();
}
else{
        battle.battleResume();
}
```

Terdapat pula suatu reaksi khusus yang merupakan respon dari serangan lawan yakni mengelak(evade). Keberhasilan untuk mengelak dari serangan ditentukan dengan membandingkan nilai status agility dengan suatu nilai random 100. Jika nilai random yang dihasilkan kurang dari nilai status agility maka hero akan mengelak dari serangan. Jadi semakin tinggi nilai status agility menyebabkan kemungkinan untuk mengelak menjadi semakin besar. Berikut ini adalah penggalan pernyataan serangan monster terhadap hero.

```
int chance=random.nextInt(100);
if(target<2){
        if(chance<bField.hero[target].agility&&!bField.hero[target].sleep){
                bField.hero[target].evade=true;
                bField.hero[target].effect=0;
        }
        else{
                bField.hero[target].effect=(((1000*attack)/100)-
((((1000*attack)/100)*bField.hero[target].defend)/100));
                bField.hero[target].hp-=(((1000*attack)/100)-
((((1000*attack)/100)*bField.hero[target].defend)/100));
                if(bField.hero[target].type.equalsIgnoreCase("sorcerer")){
                        bField.hero[target].special++;
                }
        }
}
else{
        if(chance<bField.hero[target].agility*2&&!bField.hero[target].sleep){
                bField.hero[target].evade=true;
                bField.hero[target].effect=0;
        }
        else{
                bField.hero[target].effect=(((1000*attack)/100)-
((((1000*attack)/100)*bField.hero[target].defend)/100));
                bField.hero[target].hp-=(((1000*attack)/100)-
((((1000*attack)/100)*bField.hero[target].defend)/100));
                if(bField.hero[target].type.equalsIgnoreCase("sorcerer")){
                        bField.hero[target].special++;
                }
        }
}
```

Seperti terlihat di atas bahwa posisi hero pada barisan tempur menentukan keberhasilan untuk mengelak dari serangan. Pada posisi 2(belakang), status agility hero menjadi dua kali lebih tinggi daripada hero pada posisi 0 dan 1(depan).

4.1.5.3 Reaksi Monster

Aksi monster ketika telah tiba gilirannya dari antrian tempur ditentukan oleh nilai random. Nilai random tersebut mewakili aksi yang akan dilakukan oleh monster. Setiap tipe monster memiliki kombinasi random aksi yang spesifik. Berikut ini adalah penggalan pernyataan pada metode doAction() untuk monster tipe dragon dan abomination.

```
public void doAction(int t,BattleField bField){
      target=t;
      int action=random.nextInt(5);
      if(type.equalsIgnoreCase("dragon")||type.equalsIgnoreCase("abomination")){
            if(action==0||action==1||action==2){
                  doAttack(bField);
            }
            else if(action==3){
                  doGuard();
            }
            else if(action==4){
                  doMagic(bField);
            }
      }
```

Monster juga memiliki respon mengelak dari serangan seperti halnya pada hero yang juga ditentukan oleh status agility.

4.1.5.4 Animasi Tempur

Pada battleField terdapat thread gambar yang akan menggambar semua hero maupun monster yang terlibat dalam pertempuran beserta animasi efek serta background.

Sebelum menggambar, dijalankan terlebih dahulu masing-masing metode drawHero() serta drawMonster() untuk mempersiapkan image yang akan digambar beserta posisinya. Kedua metode tersebut digunakan untuk mempersiapkan gambar *hero* atau monster yang akan digambar oleh *thread* gambar berdasarkan aksi dari *hero* atau monster tersebut. Berikut ini penggalan metode drawHero() untuk melakukan serangan.

```
private void drawHero(Hero hero){
      if(hero.live){
```

```
        if(hero.onAttack){
            if(hero.cntAttack>=hero.imgAttack.length+hero.imgEffect.length){
                onEffect=false;
                blkToMons=false;
                monster[monsTarget].evade=false;
                monster[monsTarget].onEvade=false;
                monster[monsTarget].onDamage=false;
                cntEffect=0;
                hero.onAttack=false;
                hero.cntAttack=0;
                hero.x=hero.posx;
                hero.y=hero.posy;
                hero.imgActive=hero.imgStand[hero.cntStand];
                battle.battleResume();
            }
            else{
                if(hero.cntAttack<hero.imgAttack.length){
                    if(hero.cntAttack==0){
                        hero.sndAttack.play();
                        hero.sndWeapon.play();
                    }
                    if(hero.type.equalsIgnoreCase("swordman")||hero.type.equ
alsIgnoreCase("brawler")){
                        hero.x=monster[monsTarget].x+10;
                        if(hero.height>=monster[monsTarget].height){
                            hero.y=monster[monsTarget].y-((hero.height-
monster[monsTarget].height)/2);
                        }
                        else{
                            hero.y=monster[monsTarget].y+((monster[monsT
arget].height-hero.height)/2);
                        }
                        hero.imgActive=hero.imgAttack[hero.cntAttack];
                    }
                    if(hero.type.equalsIgnoreCase("shooter")||hero.type.equa
lsIgnoreCase("sorcerer")){
                        hero.imgActive=hero.imgAttack[hero.cntAttack];
                    }
                    hero.onWeapon=true;
                }
                else{
                    hero.onWeapon=false;
                    if(monster[monsTarget].evade){
                        monster[monsTarget].onEvade=true;
                    }
                    else{
                        onEffect=true;
                        blkToMons=true;
                        monster[monsTarget].onDamage=true;
                        imgEffect=hero.imgEffect[cntEffect];
                        xEffect=monster[monsTarget].x;
                        yEffect=monster[monsTarget].y;
                        if(cntEffect==0){
                            hero.sndHit.play();
                        }
                    }
                    cntEffect++;
                }
                hero.cntAttack++;
            }
        }
```

Seperti terlihat di atas bahwa animasi serangan hero berlangsung sampai nilai cntAttack telah mencapai total image serangan hero + total image efek serangan hero. Tiap tipe hero menempati posisi serangan yang berbeda-beda. Animasi efek

serangan berlangsung setelah nilai cntAttack telah melewati total image serangan hero.

Proses animasi yang dilakukan berdasarkan aksi hero baik dipilih melalui panel tempur atau aksi yang merupakan respon suatu respon serangan seperti mengelak(evade). Bila tidak ada suatu aksi yang terjadi maka hero akan melakukan animasi berdiri. Berikut ini adalah penggalan metode drawHero() untuk animasi berdiri.

```
else{
      hero.cntStand++;
      if(hero.cntStand>=hero.imgStand.length){
            hero.cntStand=0;
      }
      hero.imgActive=hero.imgStand[hero.cntStand];
}
```

Berikut ini adalah penggalan pernyataan dalam metode paint() battleField untuk menggambar animasi hero.

```
for(int i=0;i<hero.length;i++){
      drawHero(hero[i]);
      offscreenGraphics.drawImage(hero[i].imgActive,hero[i].x,hero[i].y,this);
      if(hero[i].onWeapon){
            if(hero[i].onAttack){
            offscreenGraphics.drawImage(hero[i].weapon.imgWeapon[hero[i].cntAttack-
1],hero[i].x,hero[i].y,this);
            }
            if(hero[i].onSpecial){
                  offscreenGraphics.drawImage(hero[i].weapon.imgWeapon[hero[i].cntSpe
cial-1],hero[i].x,hero[i].y,this);
            }
      }
```

Seperti terlihat di atas bahwa metode drawHero() yang menentukan image beserta posisinya yang akan digambar oleh metode paint().

4.1.6 Penyimpanan dan Loading

Ketika akan melakukan proses penyimpanan, terlebih dahulu diciptakan obyek SavePack untuk menyimpan seluruh informasi yang ada dalam game. Berikut ini adalah pernyataan inisialisasi obyek SavePack.

```
public SavePack(int id,Game game){
      idState=id;
      heroPack=new Vector();
      for(int i=0;i<game.heroPack.size();i++){
```

saat itu melalui *socket* port 10001. Berikut ini penggalan pernyataan untuk meminta kiriman obyek SavePack pada metode loadGame().

```
SavePack savePack=new SavePack();
try{
      Socket socket=new Socket(getDocumentBase().getHost(),10001);
      ObjectInput objIn=new ObjectInputStream(socket.getInputStream());
      BufferedWriter                  request=new                  BufferedWriter(new
OutputStreamWriter(socket.getOutputStream()));
      request.write("load "+login);
      request.newLine();
      request.flush();
      savePack=(SavePack)objIn.readObject();
      objIn.close();
      request.close();
      socket.close();
}
catch(Exception e){
      System.out.println(e.toString());
}
```

Setelah obyek SavePack diterima, kemudian diinisialisasikan seluruh hero, perlengkapan, uang, serta state berdasarkan informasi yang tersimpan dalam obyek SavePack tersebut. Berikut ini penggalan pernyataan dalam metode loadGame() untuk menginisialisasikan informasi yang terdapat dalam obyek SavePack.

```
money=savePack.money;
Hero[] hero=new Hero[savePack.heroPack.size()];
for(int i=0;i<savePack.heroPack.size();i++){
      hero[i]=new Hero();
      hero[i].setHero((HeroCopy)savePack.heroPack.elementAt(i),this);
      heroPack.addElement(hero[i]);
}
for(int i=0;i<savePack.battleTeam.length;i++){
      battleTeam.addElement(hero[savePack.battleTeam[i]]);
}
itemEquip=savePack.itemEquip;
armorEquip=savePack.armorEquip;
accesorisEquip=savePack.accesorisEquip;
for(int i=0;i<savePack.weaponEquip.size();i++){
      StringTokenizer                                                    token=new
StringTokenizer(savePack.weaponEquip.elementAt(i).toString());
      String[] weaponSet=new String[token.countTokens()];
      int j=0;
      while(token.hasMoreTokens()){
            weaponSet[j]=token.nextToken();
            j++;
      }
      Weapon weapon=new Weapon(weaponSet[0],this);
      for(int w=0;w<Integer.parseInt(weaponSet[1]);w++){
            putWeapon(weapon);
      }
}
state=new State[vState.size()];
for(int i=0;i<state.length;i++){
      if(i<savePack.statePack.size()){
            state[i]=new
State((SetState)vState.elementAt(i),(StateCopy)savePack.statePack.elementAt(i),this)
;
      }
      else{
            state[i]=new State((SetState)vState.elementAt(i),i,this);
      }
      state[i].setLocation(xLegend+state[i].px,yLegend+state[i].py);
```

```
        if(i>0){
            state[i].setPrev(state[i-1]);
            state[i-1].setNext(state[i]);
        }
}
```

4.1.7 Fasilitas Tambahan

Game ini menyediakan fasilitas tambahan untuk melihat ranking dari seluruh pemain yang telah terdaftar pada basis data server game. Urutan ranking didasarkan pada status dari tim hero yang dimiliki oleh pemain yang telah disimpan di server game.

Untuk mendapatkan urutan ranking pemain, mula-mula game meminta server game untuk mengirimkan obyek vector yang berisi urutan ranking pemain melalui *socket* port 10001. Setelah obyek tersebut diterima game, maka ditampilkanlah urutan ranking seluruh pemain pada bingkai ranking. Berikut ini pernyataan pada metode rankGame().

```
public void rankGame(){
      Vector vecRank=new Vector();
      try{
            Socket socket=new Socket(getDocumentBase().getHost(),10001);
            ObjectInput objIn=new ObjectInputStream(socket.getInputStream());
            BufferedWriter                request=new                BufferedWriter(new
OutputStreamWriter(socket.getOutputStream()));
            request.write("rank");
            request.newLine();
            request.flush();
            vecRank=(Vector)objIn.readObject();
            objIn.close();
            request.close();
            socket.close();
      }
      catch(Exception e){
            System.out.println(e.toString());
      }
      FrmRank frmRank=new FrmRank(vecRank);
      frmRank.dialog.show();
}
```

4. Posisi seluruh toko, penginapan, blackSmith, serta pencatat permainan.
5. Daftar perlengkapan yang dijual maupun yang disediakan oleh seluruh toko dan blackSmith.
6. Obyek *town* beserta posisinya.
7. Partner beserta tipe heronya, dialog, dan posisinya.
8. Pengunci state beserta dialog dan posisinya.
9. People beserta dialog dan posisinya.
10. Posisi pintu masuk/keluar *town*.
11. Dialog di saat kondisi state terkunci.

Pada form untuk menambah *dungeon*, hal-hal yang bisa ditambahkan oleh administrator ke dalam *dungeon* antara lain :

1. Nama *dungeon* serta statusnya apakah terkunci dan terlihat.
2. Daftar monster yang ada dalam *dungeon*.
3. Sub*Dungeon* beserta image dan pemetaan areanya.
4. Kotak harta karun beserta isi dan posisinya.
5. Posisi awal hero pada tiap sub*Dungeon*.
6. Posisi bos monster pada akhir sub*Dungeon*.
7. Obyek sub*Dungeon* beserta posisinya.
8. Posisi pintu masuk dan keluar masing-masing sub*Dungeon*.
9. Dialog pada saat bertemu bos monster.
10. Dialog setelah mengalahkan bos monster.
11. Image obyek bos monster pada akhir sub*Dungeon*.
12. Image background pertarungan.
13. Dialog di saat kondisi state terkunci.

Masing-masing dari pengaturan state tersebut kemudian ditambahkan sebagai obyek SetState ke dalam obyek vector vState yang sedang aktif. Berikut ini adalah penggalan pernyataan untuk menambahkan obyek SetState ke dalam vector vState.

```
try{
      ObjectInputStream                objIn=new                ObjectInputStream(new
FileInputStream(serverGame.dirState+serverGame.tVector.getText()));
      Vector vState=(Vector)objIn.readObject();
      vState.addElement(setState);
      serverGame.tState.setText(Integer.toString(vState.size()));
      File file=new File(serverGame.dirState+serverGame.tVector.getText());
      ObjectOutputStream objOut=new ObjectOutputStream(new FileOutputStream(file));
      objOut.writeObject(vState);
      objOut.flush();
      objOut.close();
}
catch(Exception ioe){}
```

Administrator juga bisa memilih obyek vState yang telah ada sebelumnya dalam bentuk file "*.sta" pada file dialog untuk di-load sehingga ia tidak perlu menciptakan skenario pengaturan statu baru dari awal. Berikut ini pernyataan ketika memilih obyek vState dari file.

```
FileDialog fDialog=new FileDialog(this,"Select File");
fDialog.setDirectory("state/");
fDialog.setFile("*.sta");
fDialog.show();
if(fDialog.getFile()!=null){
      tVector.setText(fDialog.getFile());
      dirState=fDialog.getDirectory();
      try{
            ObjectInputStream            objIn=new            ObjectInputStream(new
FileInputStream(dirState+tVector.getText()));
            Vector vState=(Vector)objIn.readObject();
            tState.setText(Integer.toString(vState.size()));
      }
      catch(Exception ioe){}
      btnAdd.setEnabled(true);
}
```

4.3 Penyediaan Aplikasi Pendukung

Selain aplikasi yang ada pada game dan server game yang berbasis Java, disertakan pula aplikasi lain yang mendukung jalannya sistem game ini.

4.3.1 Basis Data Pemain

Basis data pemain yakni GameData menggunakan Microsoft Access. Dipilihnya Microsoft Access karena selain field data yang dibutuhkan cukup sederhana juga dalam hal ini game ini masih dalam tahap riset.

Field-field yang digunakan dalam basis data GameData antara lain login, name, password, dan new. Field login digunakan untuk menyimpan login pemain, field name digunakan untuk menyimpan nama pemain, field password digunakan untuk menyimpan password pemain, dan field new digunakan untuk menyimpan status pemain apakah baru atau sudah pernah menyimpan permainannya.

4.3.2 Aplikasi Pemetaan Image

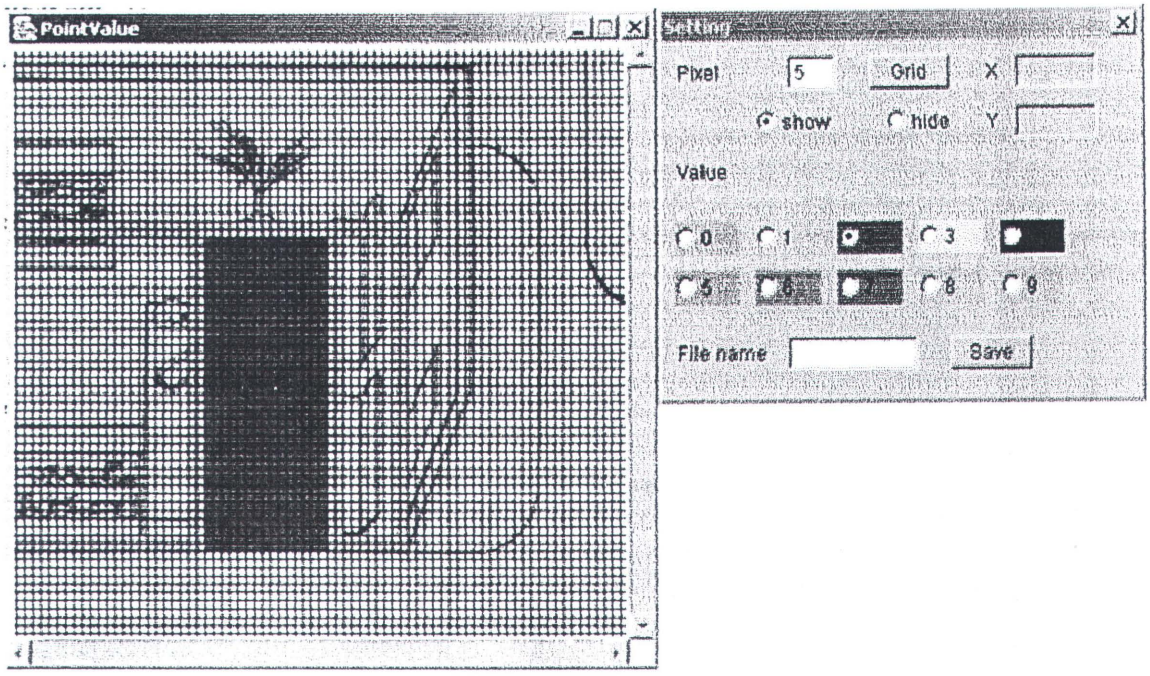
Untuk memetakan suatu image area, dibuat aplikasi berbasis Java yang dinamakan PointValue.

Dengan aplikasi tersebut, area image dapat dibagi menjadi berpetak-petak berdasarkan ukuran image dan ukuran petak. Kemudian suatu nilai tertentu bisa diisikan ke dalam petak-petak tersebut.

Fasilitas yang diberikan aplikasi ini antara lain :

1. Setiap nilai diwakili oleh suatu warna.
2. Nilai yang diisikan pada image secara langsung bisa dilihat.
3. Nilai bisa diisikan satu-persatu atau dengan men-drag mouse sehingga membentuk suatu area.
4. Kecepatan dalam mengisikan nilai pada petak image.
5. Hasil pemetaan disimpan dalam bentuk obyek Java yang langsung bisa dikenali oleh aplikasi game ini.

Berikut ini adalah tampilan dari aplikasi PointValue tersebut.



Gambar 4.5 Tampilan Aplikasi PointValue

BAB V

UJI COBA PERANGKAT LUNAK

BAB V

UJI COBA PERANGKAT LUNAK

Pada Bab 5 ini akan dibahas uji coba aplikasi Role Playing Game ini. Tujuan uji coba adalah untuk mengetahui apakah perangkat lunak yang dihasilkan dapat menjalankan fungsi-fungsinya dengan baik dan sesuai dengan tujuan Tugas Akhir ini, yaitu menghasilkan suatu perangkat lunak *Role Playing Game* yang dapat dijalankan dalam lingkungan internet. Uji coba dilakukan dalam lingkungan dan skenario yang telah ditentukan.

5.1 Lingkungan Uji Coba

Uji coba dilakukan pada komputer yang memiliki spesifikasi prosesor AMD Athlon 1 GHz dengan memori 256 MB. Sistem operasi yang dijalankan pada komputer tersebut adalah MS Windows 2000. Komputer tersebut digunakan sebagai server game dan sekaligus sebagai klien game.

Pada komputer tersebut terdapat web server Tomcat yang digunakan untuk mengakses game melalui web browser. Web browser haruslah memiliki fasilitas Java Virtual Machine dalam hal ini yang digunakan adalah MS Interner Explorer 5. Pada komputer tersebut telah ter-install pula Java Plug-in 1.4.0 supaya web browser yang digunakan men-support fungsi-fungsi yang ada game. Pada komputer tersebut terdapat pula MS Access yang digunakan untuk mengakses basis data game.

5.2 Skenario Uji Coba

Skenario yang akan diujikan adalah sebagai berikut :

1. Inisialisasi lingkungan permainan game. Dari skenario ini lingkungan permainan diinisialisasikan dari suatu objek rancangan pengaturan lingkungan permainan yang telah dipilih pada server game.
2. Memainkan game. Pada skenario ini akan diujikan segala aspek yang ada pada game ini apakah berjalan sesuai dengan semestinya.
3. Penyimpanan dan loading. Pada skenario ini akan diujikan apakah game dapat menyimpan serta me-load kembali status permainannya.
4. Menciptakan rancangan lingkungan permainan baru. Pada skenario ini akan dicoba untuk menciptakan suatu rancangan lingkungan permainan yang baru dan kemudian dicoba untuk memainkannya.

5.3 Pelaksanaan Uji Coba

Setelah semua sarana fisik dan perangkat lunak pendukung siap, maka dilakukan uji coba aplikasi yang telah dibuat. Sebelum uji coba dilakukan, dilakukan terlebih dulu penataan beberapa konfigurasi di sisi *server* dan *client*. Setelah penataan dilakukan dengan benar, barulah aplikasi siap dijalankan.

5.3.1 Pengaturan Konfigurasi dan Basis Data

Seluruh file class termasuk di dalamnya kelompok aplikasi game dan server game harus diletakkan pada folder yang telah dipublikasikan oleh web server Tomcat.

Pengaturan konfigurasi applet game dilakukan pada file game.htm. Pada file tersebut terdapat berbagai parameter yang digunakan dalam game. Berikut ini adalah kode html pada file game.htm.

```
<HTML>
```

```
<HEAD>
</HEAD>
<BODY BGCOLOR="000000">
<CENTER>
<APPLET
      code                = "game\structure\Game.class"
      width                           = "640"
      height                          = "580"
      >
      <param name=imgTitle value="image/title.jpg">
      <param name=imgLegend value="image/peta.jpg">
      <param name=wLegend value="660">
      <param name=hLegend value="572">
      <param name=mscOpening value="music/opening.au">
      <param name=mscStory1 value="music/story1.au">
      <param name=mscStory2 value="music/story2.au">
      <param name=msc Town value="music/town.au">
      <param name=msc Dungeon value="music/dungeon.au">
      <param name=mscBattle value="music/battle.au">
      <param name=mscBoss value="music/boss.au">
      <param name=mscWin value="music/win.au">
      <param name=mscMap value="music/map.au">
      <param name=heroLevel value="1">
      <param name=money value="1000">

      <param name=ttlStory value="3">
      <param name=storyMap0 value="image/peta.jpg">
      <param name=ttlDlgStory0 value="2">
      <param name=talk00 value="prolog;normal;Terdapatlah suatu legenda tentang tiga
mustika.">
      <param name=talk01 value="prolog;normal;Bila ketiga mustika tersebut telah
terkumpul maka jin yang bersemayam di dalamnya akan mengabulkan segala keinginan
kita.">
      <param name=storyMap1 value="image/title.jpg">
      <param name=ttlDlgStory1 value="2">
      <param name=talk10 value="prolog;normal;Pada suatu hari datanglah seorang
pemuda di suatu desa.">
      <param name=talk11 value="prolog;normal;Dia adalah seorang pembasmi monster
yang mengemban misi untuk membasmi tiga monster legenda.">
      <param name=storyMap2 value="image/peta.jpg">
      <param name=ttlDlgStory2 value="1">
      <param name=talk20 value="hero;normal;Aku akan mencari informasi tentang
keberadaan ketiga monster itu.">

</APPLET>
</CENTER>
</BODY>
</HTML>
```

Pada kode di atas terlihat applet diatur memiliki ukuran lebar 640 pixel serta tinggi 580 pixel. Terdapat dua kelompok parameter yakni konfigurasi awal game dan dialog pembuka game. Berikut ini akan dijelaskan masing-masing parameter di atas.

Kelompok parameter konfigurasi game :

1. imgTitle : letak file image judul game
2. imgLegend : letak file image peta dunia game
3. wLegend : lebar image peta dunia
4. hLegend : tinggi image peta dunia

5. mscOpening : letak file musik pembuka
6. mscStory1 : letak file musik cerita 1
7. mscStory2 : letak file musik cerita 2
8. msc*Town* : letak file musik *town*
9. msc*Dungeon* : letak file musik *dungeon*
10. mscBattle : letak file musik pertempuran
11. mscBoss : letak file musik bos monster
12. mscWin : letak file musik kemenangan
13. mscMap : letak file musik peta dunia
14. heroLevel : level hero pada awal game
15. money : uang hero pada awal game

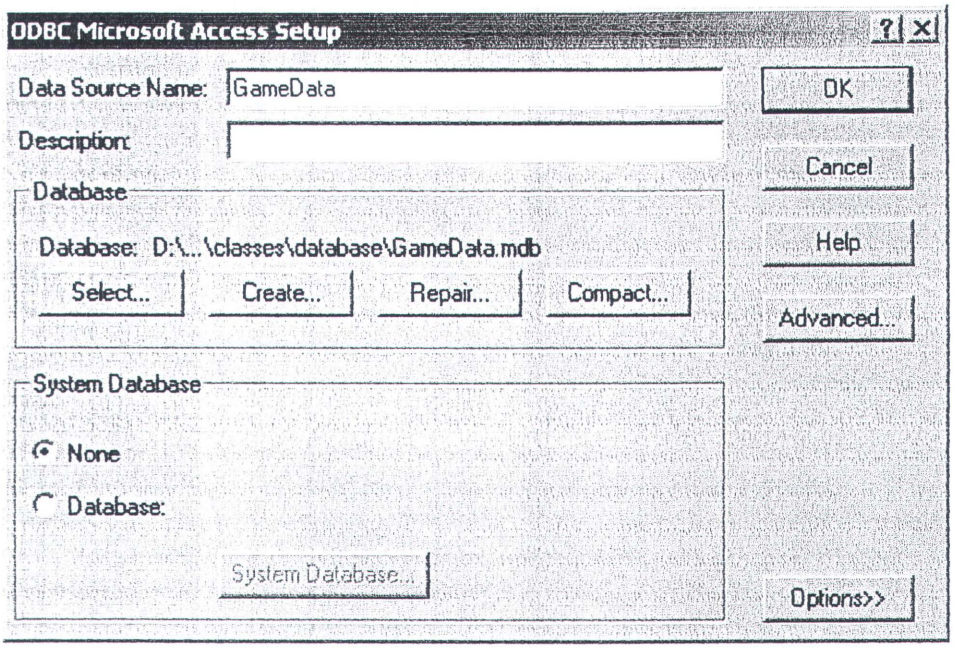
Format file musik yang mendukung aplikasi Java adalah Next/Sun au format(*.au) dengan setting mu-Law 8-bit.

Kelompok parameter dialog pembuka :

1. ttlStory : total cerita pembuka
2. storyMap*n* : letak file image background cerita ke-n
3. ttlDlgStory*n* : total dialog pada cerita ke-n
4. talk*nm* : pembicaraan pada cerita ke-n serta dialog ke-m

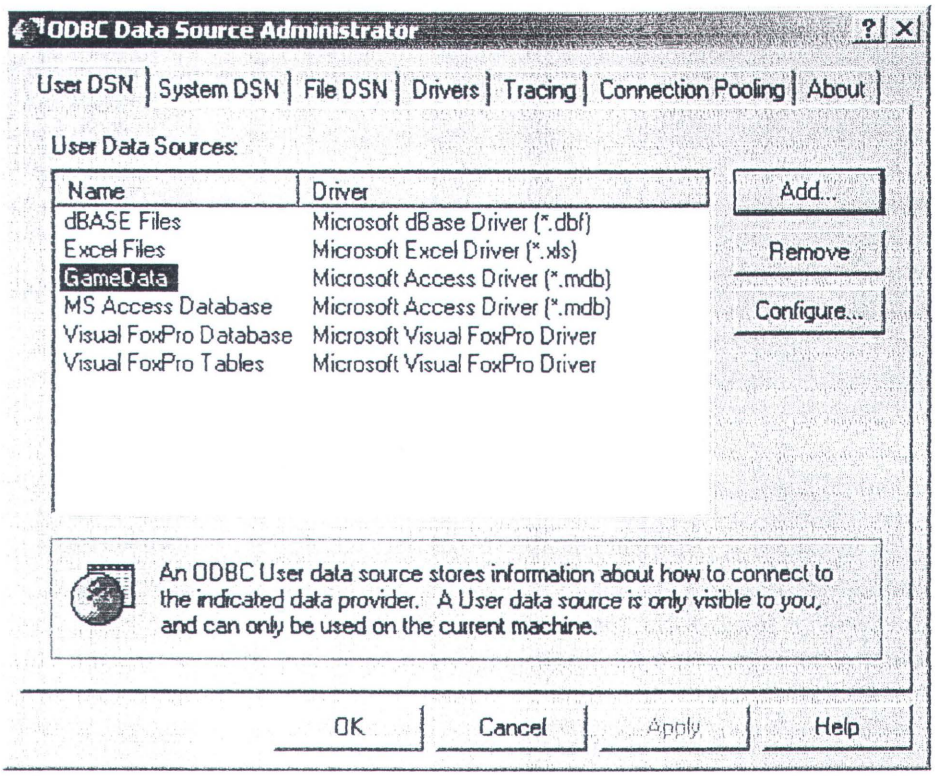
Pada parameter talk, kita bisa menentukan apakah pembicaraan tersebut dilakukan oleh prolog game atau hero. Kita juga bisa menentukan kondisi hero saat itu melalui parameter talk tersebut.

Basis data GameData harus dipasangkan terlebih dahulu pada ODBC komputer server. Berikut ini adalah tampilan setup GameData pada ODBC.



Gambar 5.1 **Gambar 5.1 ODBC Microsoft Access Setup**

Setelah GameData dipasang pada ODBC, tampilannya adalah sebagai berikut ini.



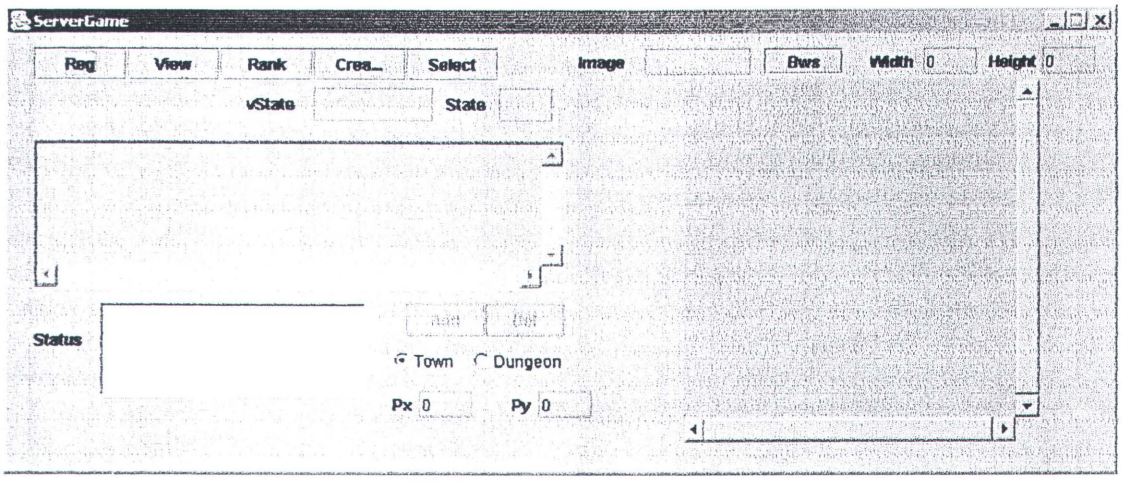
Gambar 5.2 **ODBC Data Source Administrator**

5.3.2 Pengaktifan Server Game

Menjalankan server game dilakukan dengan mengeksekusi file batch ServerGame.bat. Berikut ini adalah isi dari file batch tersebut.

```
@echo off
c:\j2sdk1.4.0\bin\java -cp D:\ProjectGame\Game\classes ServerGame
```

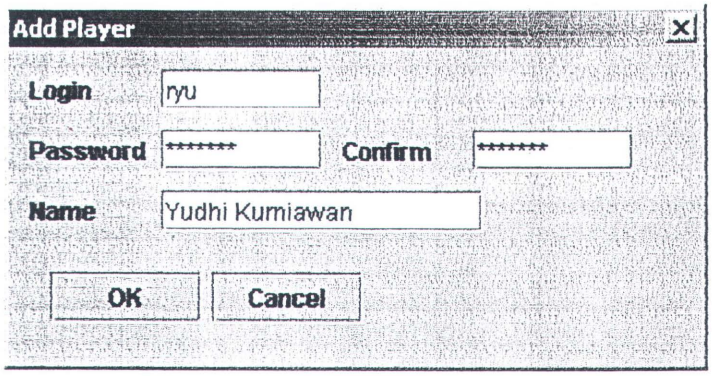
Pada file batch tersebut ditentukan folder letak file java.exe dan letak file ServerGame.class. Berikut ini adalah tampilan server game setelah dijalankan.



Gambar 5.3 Tampilan Server Game

5.3.3 Pendaftaran Pemain Baru

Pendaftaran pemain baru dilakukan pada server game dengan menekan tombol "Reg" yang kemudian akan muncul form pendaftaran login yang berisikan login, password, dan nama pemain. Berikut ini adalah tampilan form pendaftaran pemain baru.



Gambar 5.4 Tampilan Form Pendaftaran Pemain

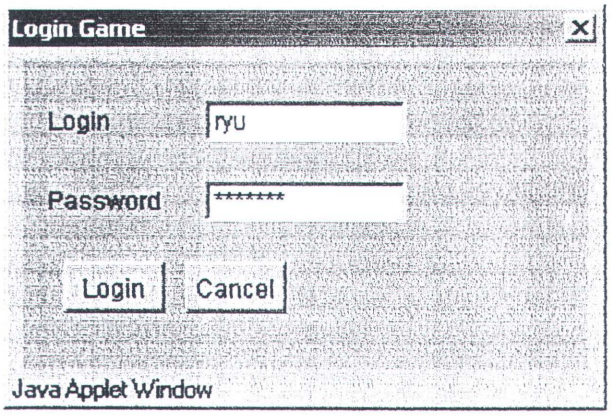
5.3.4 Menyiapkan Rancangan Lingkungan Permainan yang Telah Ada

Selanjutnya administrator memilih file objek rancangan lingkungan permainan yang telah ada pada subfolder "\state". Berikut ini tampilan pada server game ketika memilih file tersebut.

Gambar 5.5 Tampilan Memilih Rancangan

5.3.5 Login Pemain

Untuk menjalankan game, kita mengakses alamat http://ryu/game/game.htm. Ketika pertama kali game dijalankan kita akan disuguhkan form login pemain. Berikut ini adalah tampilan form tersebut.



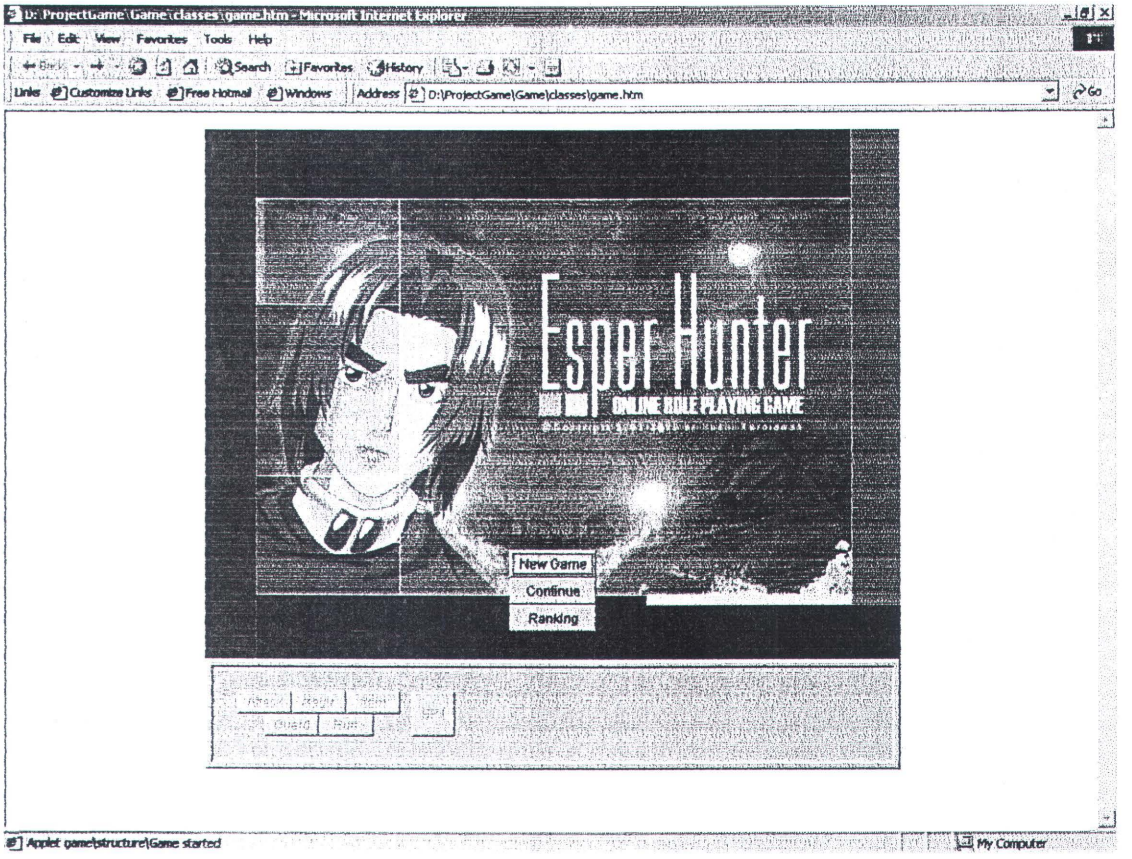
Gambar 5.6 Form Login Pemain

Kemudian kita memasukkan login beserta password yang sesuai pada basis data GameData. Bila login serta password kita tidak sesuai, kita tidak bisa masuk ke tampilan judul game. Bila kita menekan tombol "Cancel", kita akan menuju tampilan judul game namun tidak bisa mengakses tombol untuk memulai game.

Pada server game akan memberikan tampilan respon login pemain yang sesuai sebagai berikut.

5.3.6 Memainkan Game

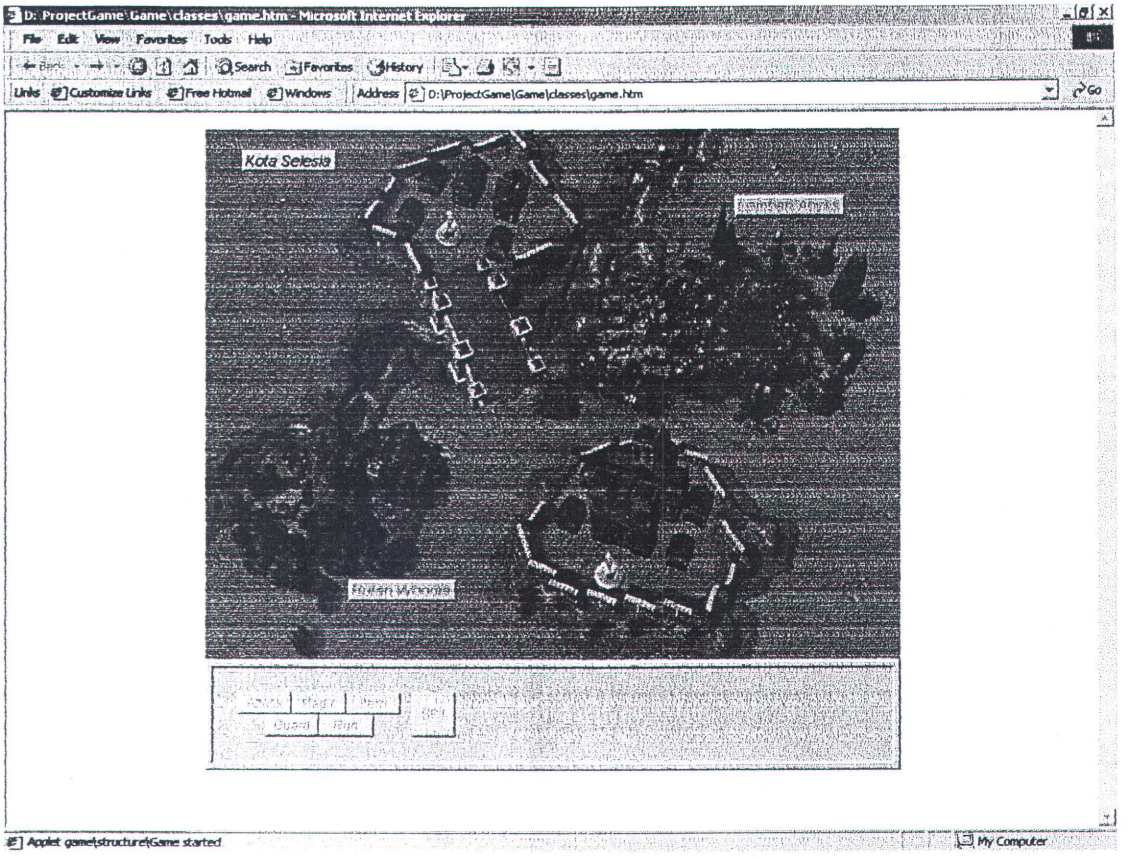
Bila login beserta password yang kita isikan sesuai kemudian menuju tampilan judul game. Berikut ini adalah tampilan judul game tersebut.



Gambar 5.7 Tampilan Judul Game

Seperti terlihat pada gambar, terdapat tiga tombol pada tampilan tersebut yakni memulai permainan baru, melanjutkan permainan, serta melihat ranking para pemain. Kita kali ini mencoba untuk memulai permainan baru.

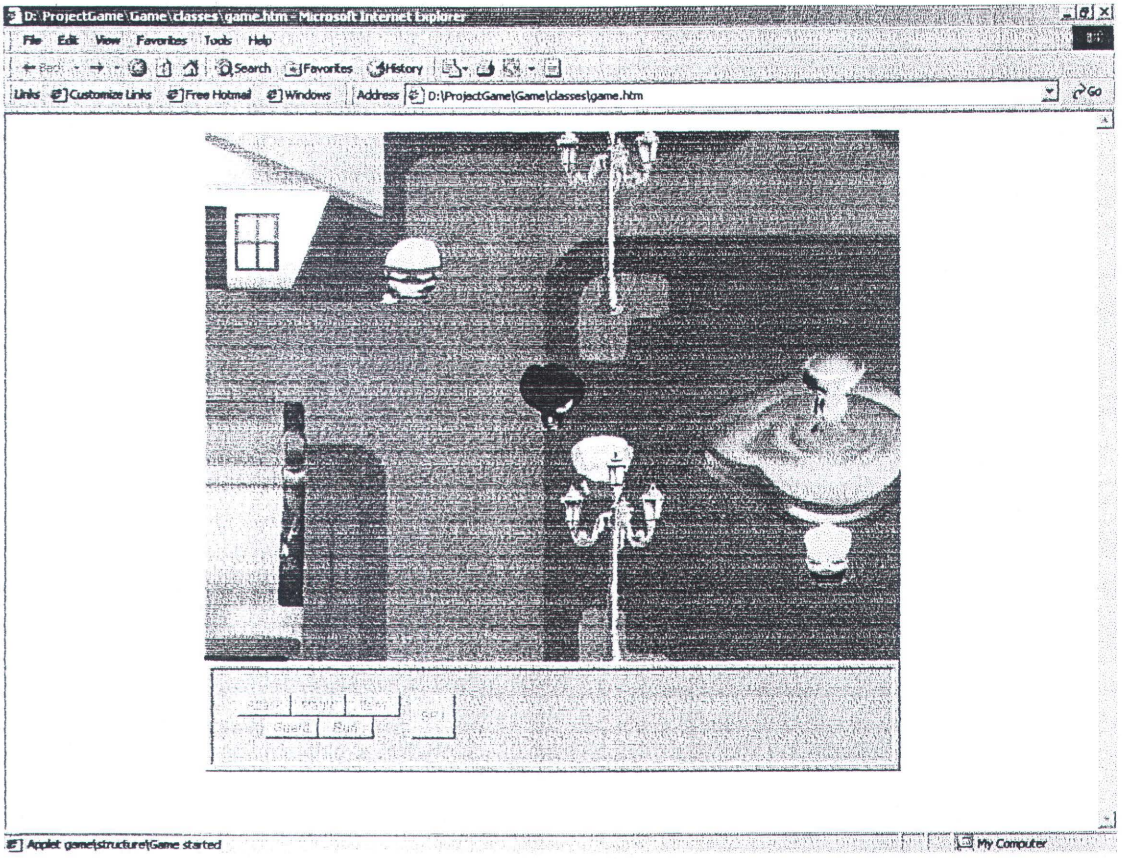
Setelah kita menekan tombol "New Game", mula-mula ditampilkanlah cerita pembuka game yang sebelumnya telah di-setting pada file game.htm. Kemudian kita akan ditampilkan peta dunia game yang di dalamnya terdapat state-state yang sebelumnya telah dirancangkan pada file vState.sta. Berikut ini adalah tampilan peta dunia game.



Gambar 5.8 **Tampilan Peta Dunia Game**

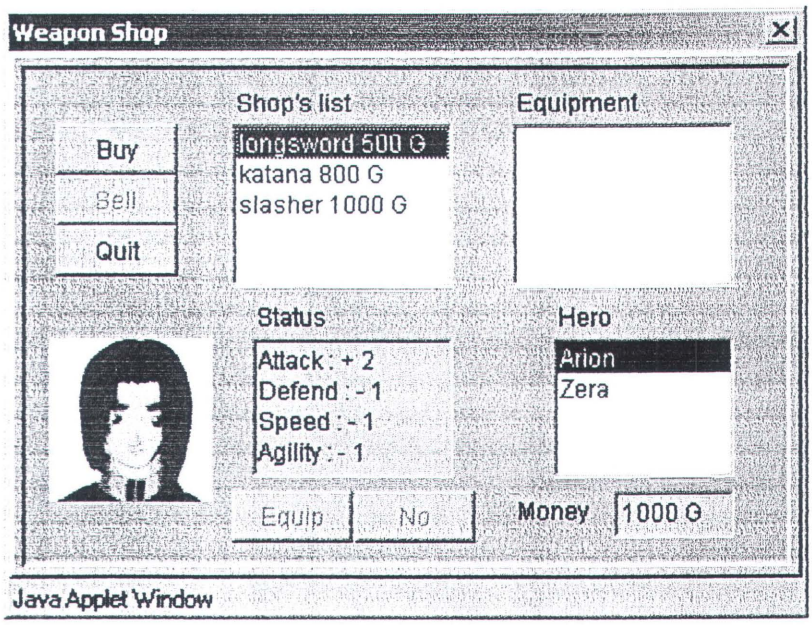
Kita bisa masuk ke state-state tersebut dengan mengkliknya. Hanya state-state yang telah terbuka kuncinya yang bisa kita akses.

Ketika kita memasuki state yang berjenis *town*, kita bisa mencoba untuk menjelajahi *town* tersebut. Keluar-masuk ke tiap-tiap rumah, berbicara dengan penduduk *town*, menginap di penginapan, berbelanja perlengkapan hero, mencari partner tim hero, dan sebagainya. Berikut ini adalah tampilan dalam suatu *town*.

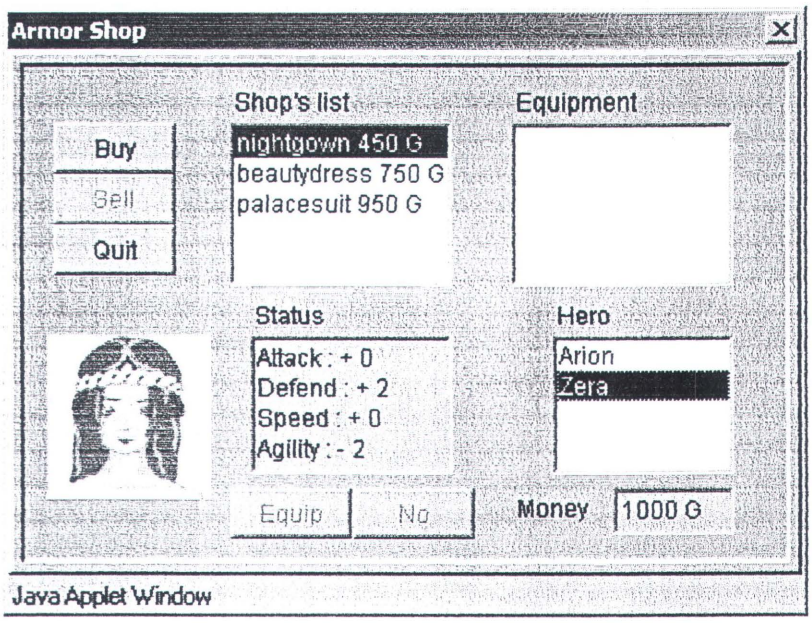


Gambar 5.9 Tampilan Dalam *Town*

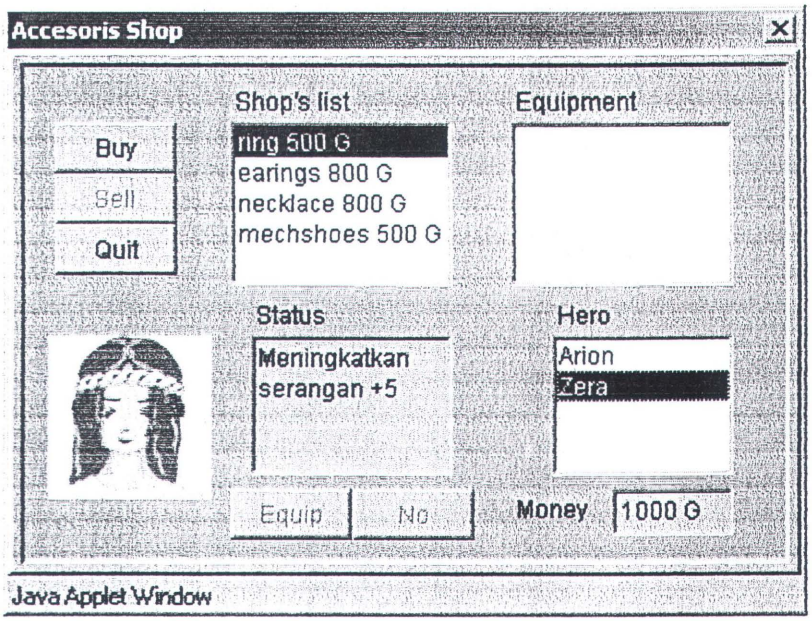
Transaksi perlengkapan hero dilakukan bila kita menemui para penjual yang ada di *town*. Kita bisa membeli dan menjual perlengkapan pada toko tersebut. Untuk perlengkapan senjata dan baju, kita bisa membandingkan perubahan status pada hero bila mengenakannya sebelum membeli atau menjual. Untuk meng-upgrade senjata, kita bisa melihat barang apa saja yang diperlukannya. Berikut ini adalah tampilan dari masing-masing toko yang ada pada *town*.



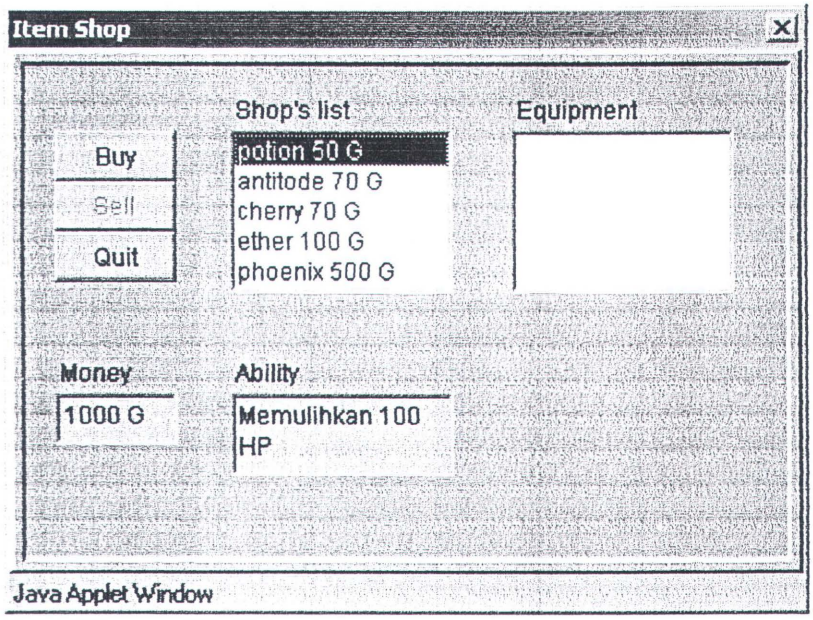
Gambar 5.10 Tampilan Toko Senjata



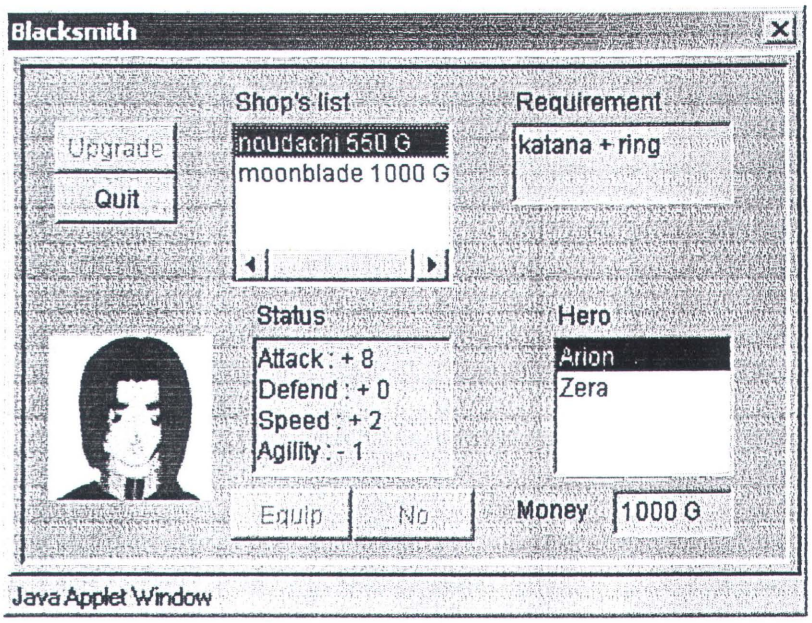
Gambar 5.11 Tampilan Toko Baju



Gambar 5.12 Tampilan Toko Aksesoris

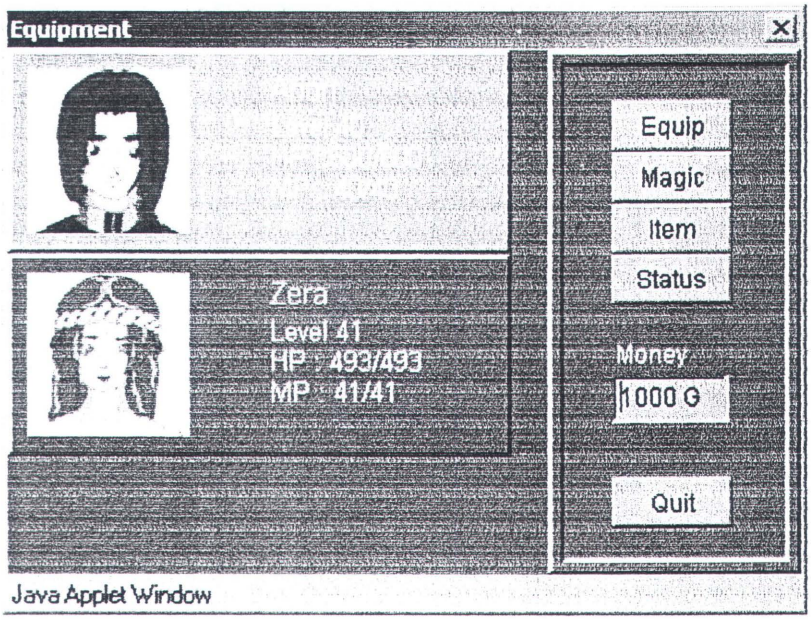


Gambar 5.13 Tampilan Toko Item



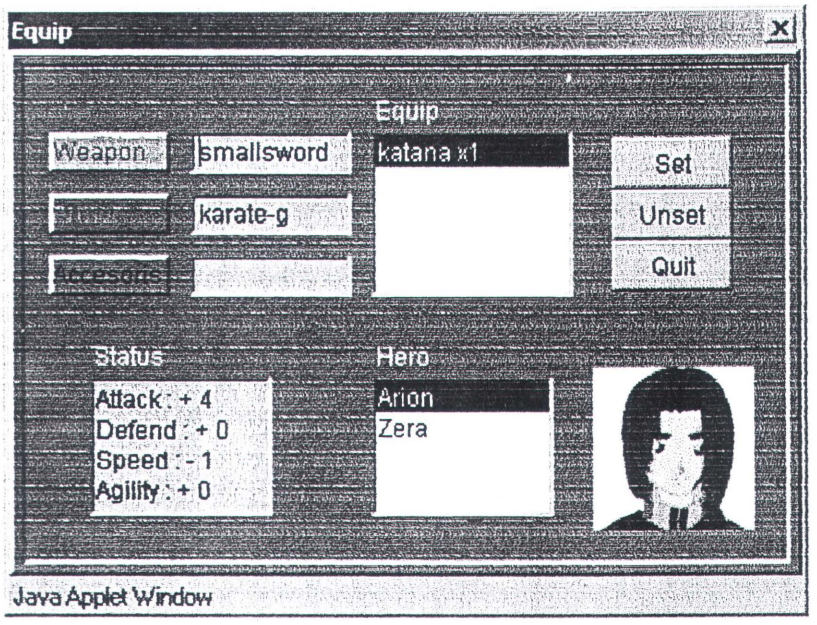
Gambar 5.14 Tampilan Blacksmith

Kita bisa mengatur perlengkapan yang ada pada tim hero dengan menekan tombol "esc". Pada menu pengaturan perlengkapan, kita bisa mengatur senjata, baju, serta aksesoris para hero, menggunakan sihir, menggunakan item, dan melihat status para hero. Berikut ini adalah tampilan menu pengaturan perlengkapan tersebut.

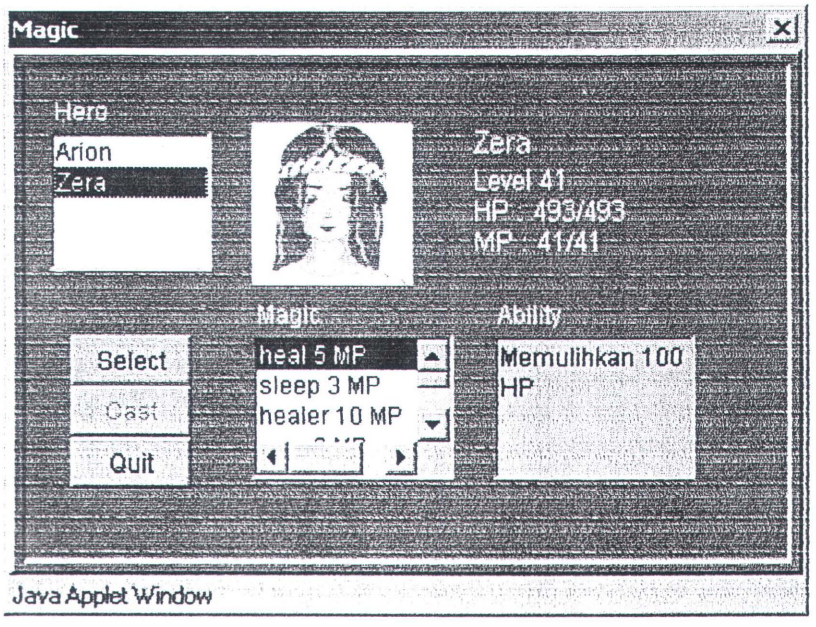


Gambar 5.15 Tampilan Menu Pengaturan Perlengkapan

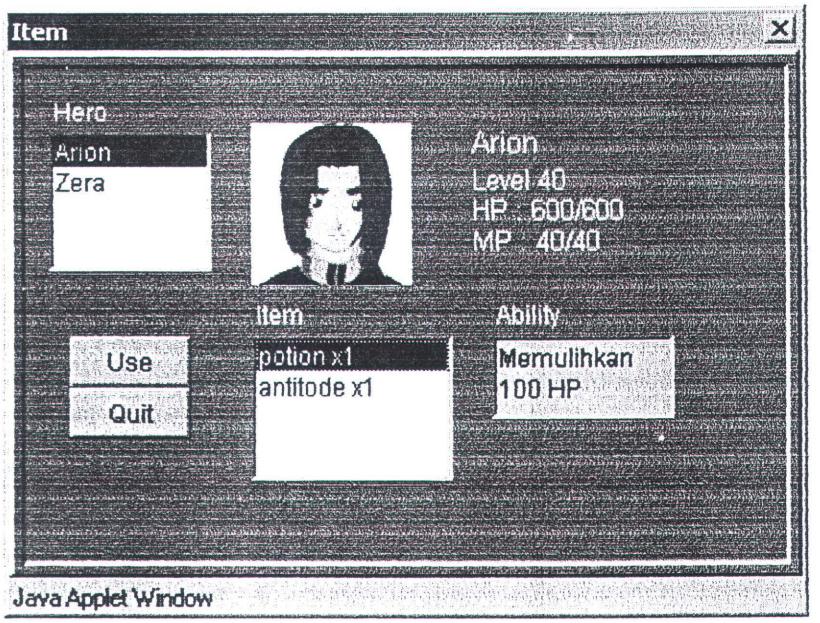
Berikut ini pula adalah tampilan dari masing-masing submenu pada menu pengaturan perlengkapan tim hero.



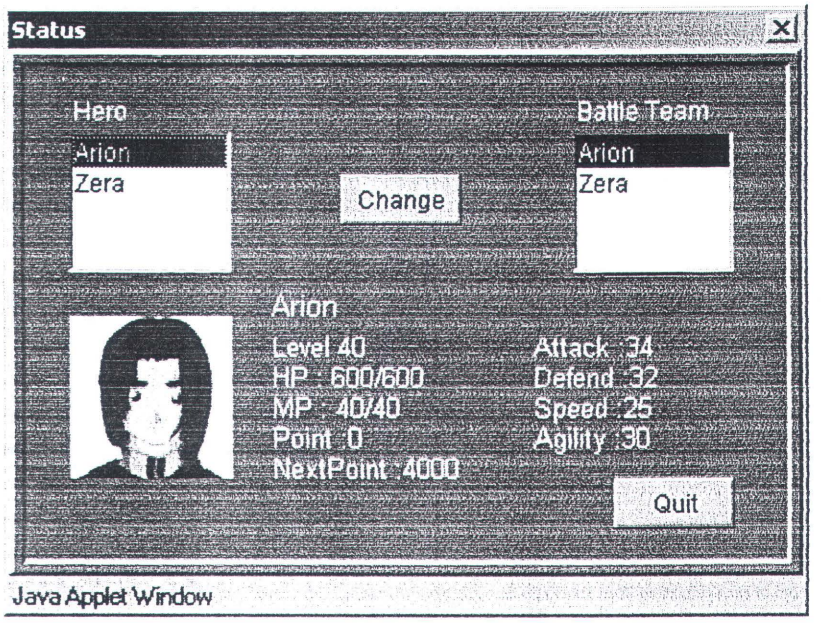
Gambar 5.16 Tampilan Menu Senjata, Baju, dan Aksesoris



Gambar 5.17 Tampilan Menu Sihir

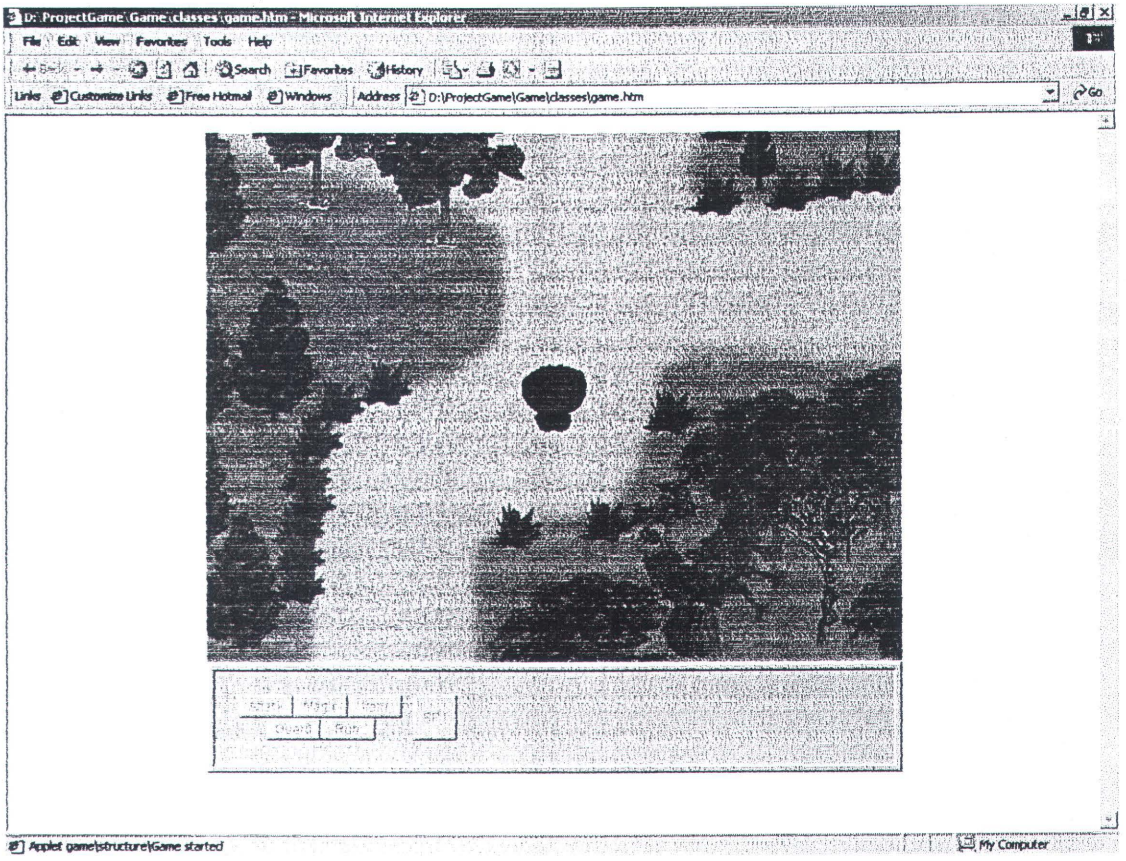


Gambar 5.18 **Tampilan Menu Item**



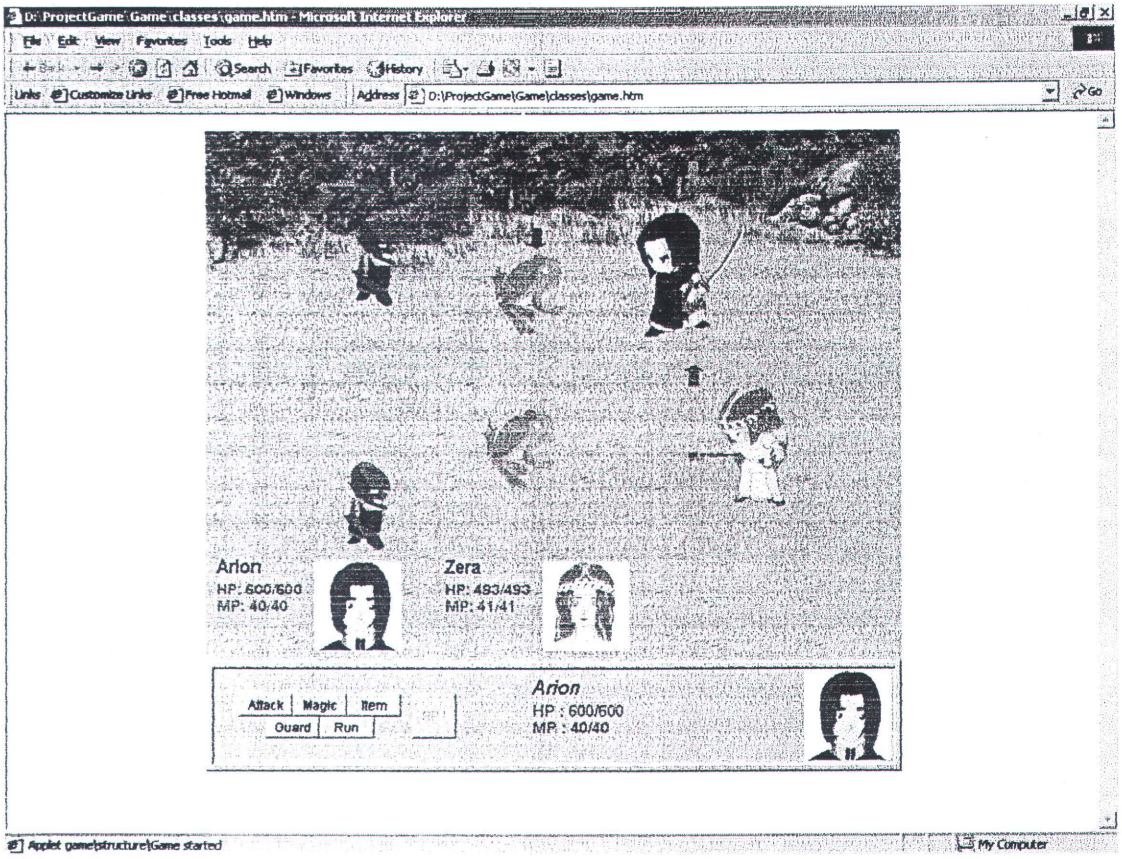
Gambar 5.19 **Tampilan Menu Status**

Bila kita memasuki state yang berjenis *dungeon*, kita bisa mencoba menjelahi *dungeon* tersebut sambil menemukan kotak harta karun yang ada di dalamnya. Berikut ini adalah tampilan dalam *dungeon*.



Gambar 5.20 Tampilan Dalam *Dungeon*

Ketika langkah hero pada *dungeon* telah mencapai nilai encounter maka arena pertempuran dibuka. Berikut ini adalah tampilan arena pertempuran antara monster melawan tim hero.

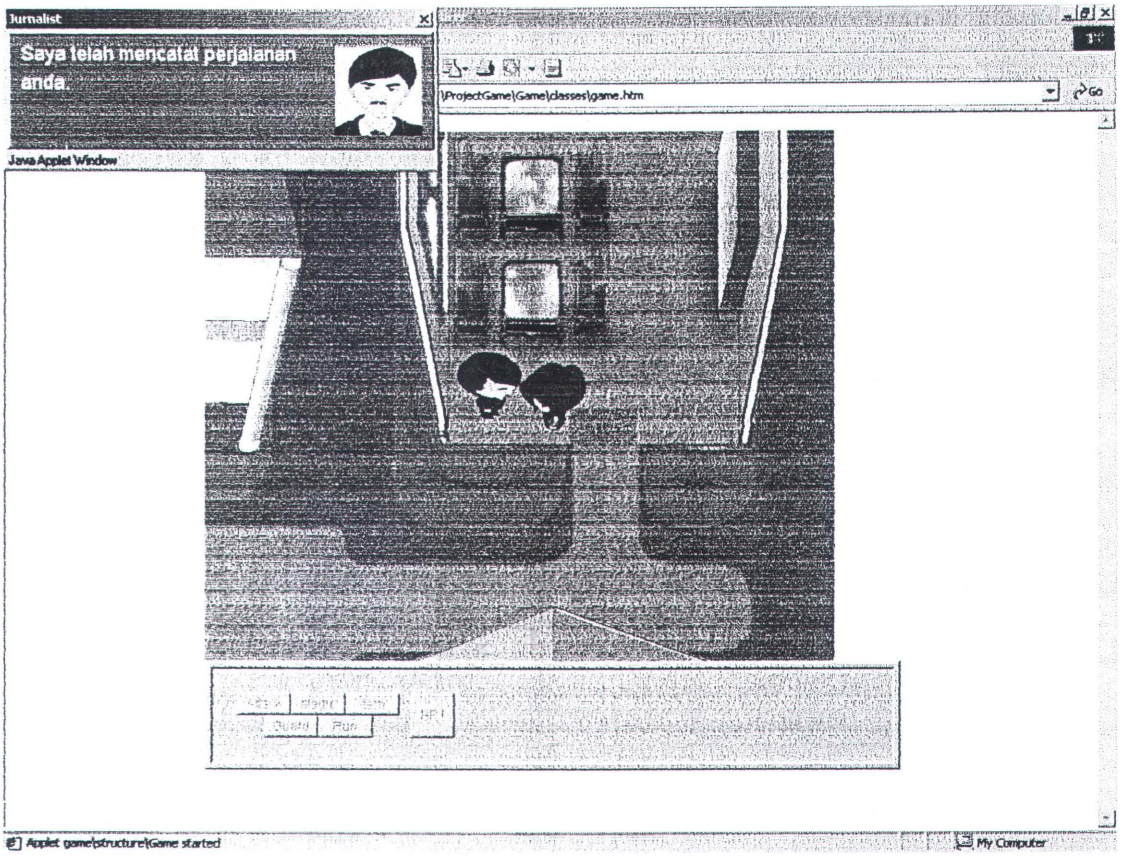


Gambar 5.21 Tampilan Arena Pertempuran

Kita bisa menentukan berbagai aksi hero melalui tombol-tombol yang ada pada panel tempur seperti terlihat di atas.

5.3.7 Penyimpanan dan Loading

Kita melakukan penyimpanan status game pada saat di dalam *town* atau *dungeon*. Bila pada *town*, kita melakukannya dengan berbicara pada people yang berprofesi sebagai pencatat perjalanan. Berikut ini adalah tampilan dialog ketika kita menyimpan status game di *town*.



Gambar 5.22 Tampilan Dialog Save Pada *Town*

Bila kita berada di *dungeon*, kita melakukannya pada kotak harta karun yang berfungsi sebagai prasasti pencatat perjalanan. Berikut ini adalah tampilan dialog ketika kita menyimpan status game di *dungeon*.

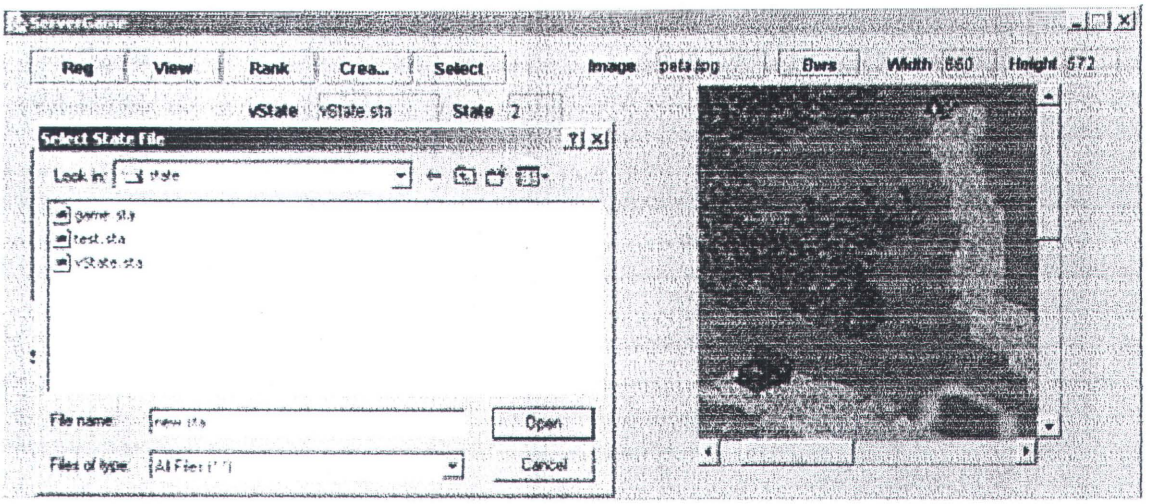
Ketika kita menyimpan status game, server game akan menciptakan file penyimpanan *.sav dengan nama sesuai dengan login pemain yang diletakkan pada folder save.

Kita melakukan load permainan dengan menekan tombol "Continue" pada tampilan judul game. Setelah kita menekannya, tampilan akan langsung dibawa ke pada saat terakhir kita menyimpan status game kita.

Ketika kita me-load status game, server game akan memberikan respon tampilan sebagai berikut.

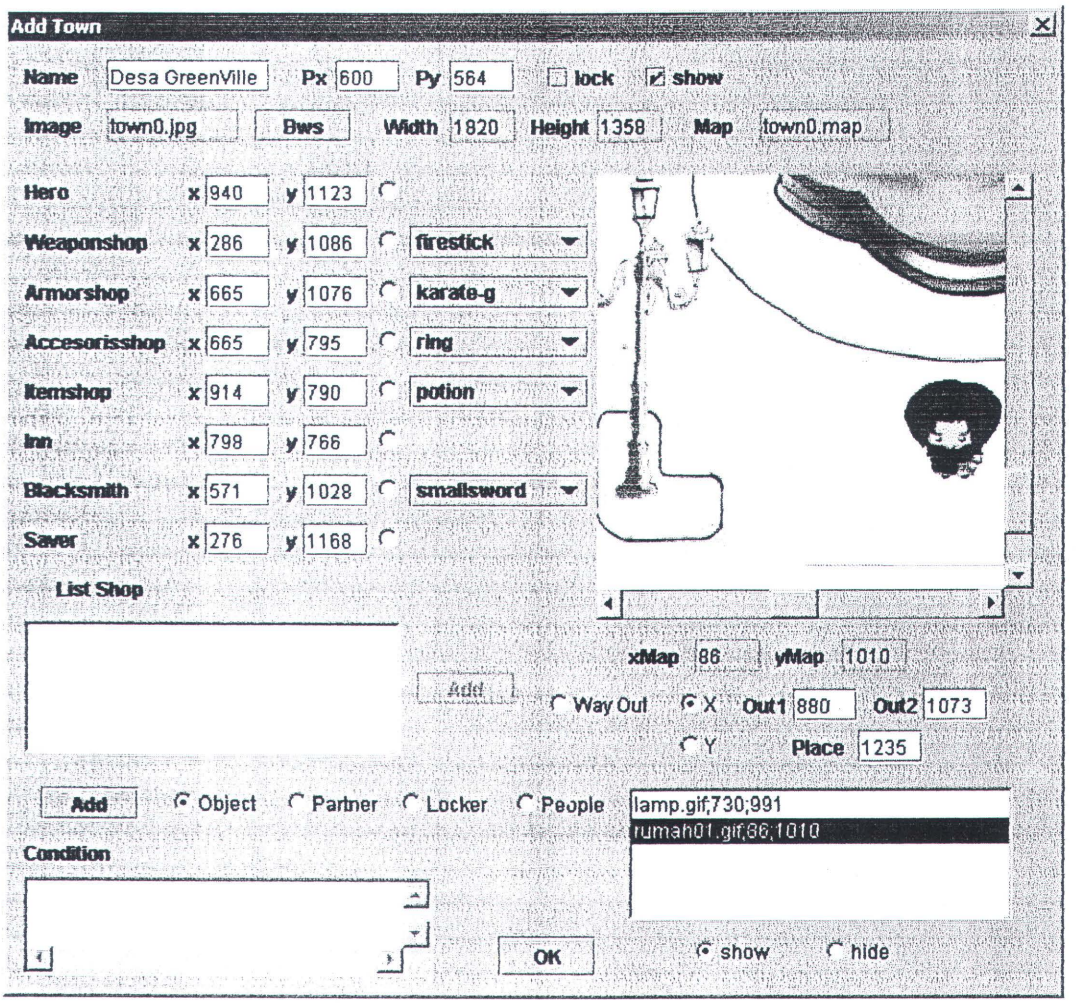
5.3.8 Menciptakan Rancangan Lingkungan Permainan Baru

Kini kita akan mencoba untuk menciptakan rancangan lingkungan permainan baru melalui fasilitas yang ada pada server game. Mula-mula memilih nama file yakni new.sta yang akan digunakan seperti terlihat pada tampilan berikut ini.



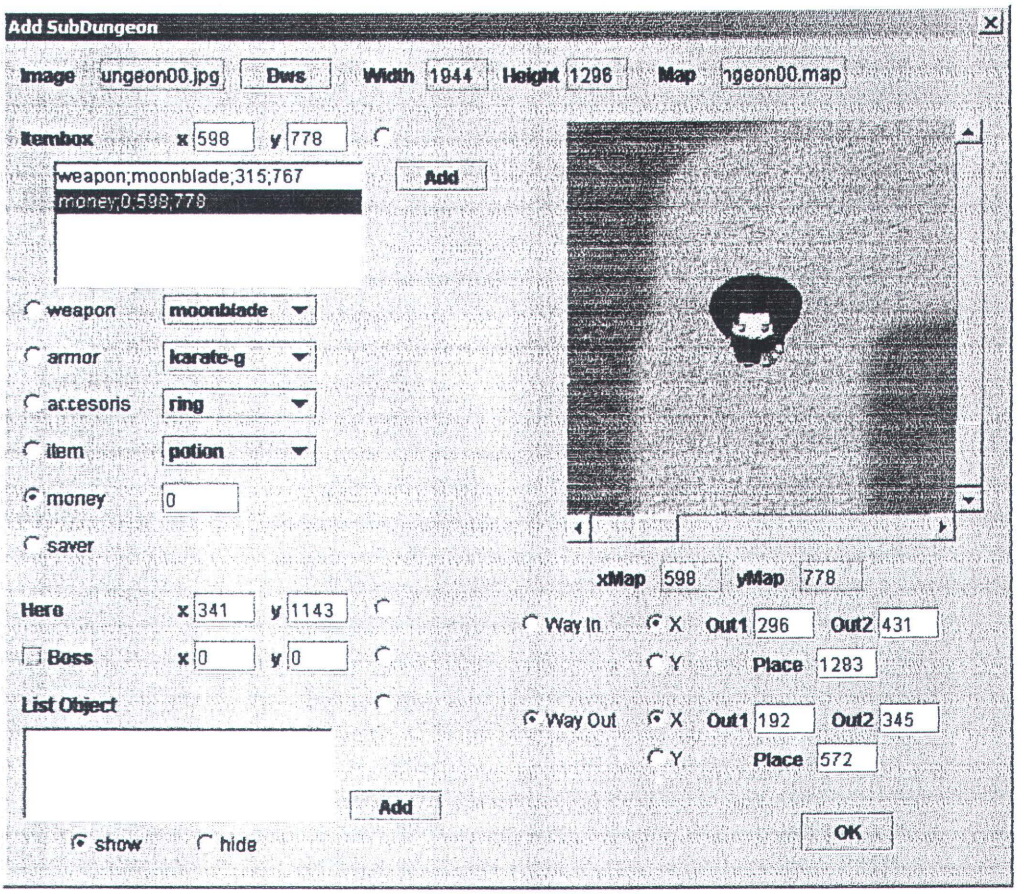
Gambar 5.23 Tampilan Menciptakan File Rancangan

Seperti terlihat pada sisi bagian kanan server game kita juga telah me-load image peta dunia yang akan digunakan dalam game kita nanti. Setelah menentukan koordinat state yang akan kita rancang pada peta dunia dengan mengkliknya, mula-mula kita akan mencoba untuk membuat sebuah rancangan *town*. Setelah menekan tombol "Add" pada server game, berikut ini adalah tampilan form rancangan *town* beserta nilai-nilai yang kita isikan.

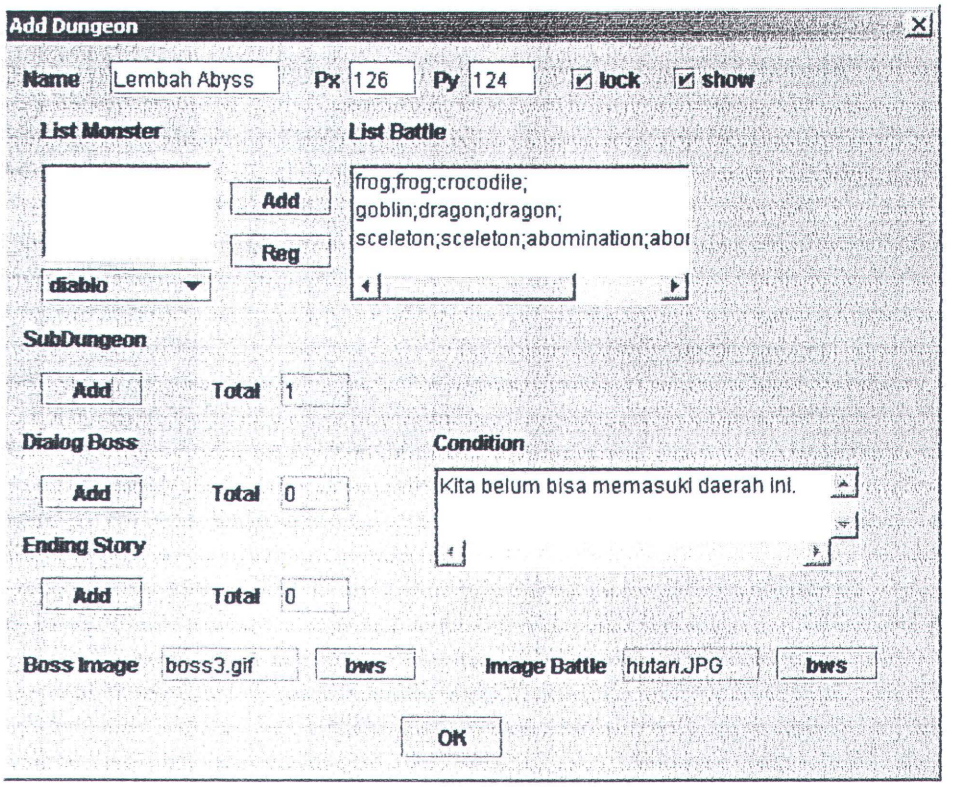


Gambar 5.24 Tampilan Form Rancangan *Town*

Kemudian kita akan mencoba untuk membuat sebuah rancangan *dungeon*. Berikut ini adalah tampilan form rancangan *dungeon* beserta nilai-nilai, sub*Dungeon*, serta monster-monster yang kita isikan.



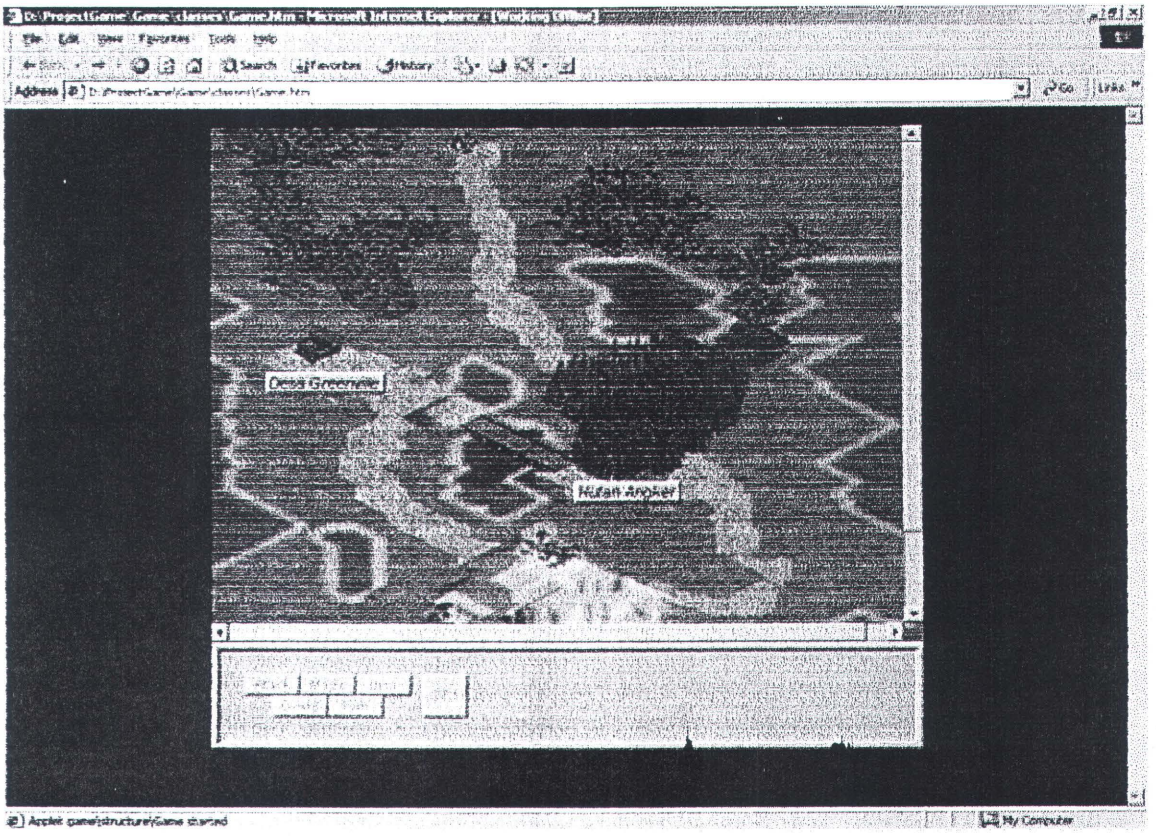
Gambar 5.25 Tampilan Form Rancangan Sub*Dungeon*



Gambar 5.26 Tampilan Form Rancangan *Dungeon*

Total state yang telah kita buat barusan adalah dua dan semua rancangan di atas telah tersimpan dalam file new.sta.

Setelah semua di atas, kita akan mencoba untuk me-load rancangan yang telah kita buat pada game. Dengan cara menjalankan game yang sama seperti yang telah dijelaskan di atas, berikut ini adalah tampilan peta dunia, isi *town*, isi *dungeon*, serta monster-monster yang dihadapi dalam pertempuran berdasarkan rancangan yang tadi kita buat.



Gambar 5.27 Tampilan Hasil Peta Dunia

5.4 Hasil Uji Coba

Dari uji coba terhadap aplikasi Role Playing Game yang telah dibuat, didapat hasil bahwa aplikasi dapat menjalankan seluruh fungsi yang ada padanya dengan baik.

Dengan hasil uji coba seperti pada pelaksanaan skenario di atas, aplikasi Role Playing Game ini telah memenuhi kebutuhan sebagai game berbasis web sesuai dengan tujuan tugas akhir ini. Mungkin kekurangannya adalah masih belum multi player.

BAB VI

PENUTUP

BAB VI

PENUTUP

Pada bab ini akan dijelaskan kesimpulan yang bisa diambil dari Tugas Akhir ini. Di samping itu pada bagian ini juga disertai kemungkinan pengembangan yang bisa dilakukan terhadap perangkat lunak yang telah dibuat.

5.1 Kesimpulan

Dari hasil seluruh perancangan, implementasi, serta uji coba game ini diambil kesimpulan sebagai berikut :

1. Pendekatan pemrograman berorientasi obyek dalam mengimplementasikan aplikasi game ini dilakukan dengan memilah-milah obyek-obyek yang ada dalam permainan seperti senjata, baju, aksesoris, sihir, monster, dan sebagainya sehingga pemrograman dan pengembangan proses dalam game menjadi lebih mudah dan *extensible*.
2. Dunia permainan pada server game diimplementasikan dalam bentuk file obyek yang terserialisasi. Struktur dalam obyek tersebut terdiri dari obyek-obyek yang berisi informasi setiap state yang ada pada dunia permainan seperti town dan dungeon.
3. Penyimpanan status permainan bisa dilakukan pada server dalam bentuk file obyek yang terserialisasi. Status permainan tersebut antara lain meliputi informasi seluruh hero dalam tim, informasi seluruh perlengkapan hero baik itu senjata, baju, aksesoris serta item, informasi seluruh state, dan informasi uang serta login pemain pada saat menyimpan.

4. Komunikasi data antara server dengan klien game dilakukan dengan obyek yang terserialisasi melalui *socket TCP*.
5. Sistem keamanan aplikasi dirancang dengan meletakkan obyek dunia permainan pada server dan hanya bisa diperoleh setelah melalui proses login.

5.2 Kemungkinan Pengembangan

Dalam pembuatan Tugas Akhir ini, terdapat beberapa kemungkinan pengembangan aplikasi yang bisa dilakukan, yaitu:

- Pengembangan aplikasi ini menjadi game berjenis *Massive MultiPlayer Online Role Playing Game*, yakni suatu genre game berjenis RPG yang bisa dimainkan oleh banyak pemain yang bisa saling berinteraksi dalam waktu nyata.
- Aplikasi game dibangun dengan Macromedia Flash. Macromedia Flash telah dikenal sebagai aplikasi yang handal dalam melakukan animasi pada web.
- Format data untuk pertukaran informasi *client-server* menggunakan format XML. XML telah menjadi standar pertukaran data dalam dunia web sekarang, sehingga Java pun menyediakan *library* untuk menanganinya. Aplikasi dalam Tugas Ahir ini masih memakai format data *string* biasa dengan *delimiter* yang telah ditentukan serta obyek Java yang telah diserialisasi. Dengan digunakannya XML dan *library* dari Java, jika terjadi perubahan struktur data, pengembangan aplikasi lebih mudah dilakukan.

DAFTAR PUSTAKA

DAFTAR PUSTAKA

1) Anuff, Ed; Java Source Book, Penuntun Pemrograman Java; Andi Yogyakarta; 1997
2) Mahmoud, Qusay H; Distributed Programming with Java; Manning Pubications Co.; 2000.
3) Sun Microsystem, Inc.; Java™ 2 SDK, Standard Edition Documentation; http://java.sun.com/j2se/1.4/docs/; Copyright © 2002 Sun Microsystem, Inc.
4) Sun Microsystem, Inc.; The Java™ Tutorial, A Practical Guide for Programmers; http://java.sun.com/docs/books/tutorial/; Nopember, 1999.
5) Jackson, Jerry R. & McClellan, Alan L.; Java by Example; Prentice Hall PTR; 1996 Sun Microsystems, Inc.; ISBN : 979-533-386-0
6) DarkDread; RPGs in QBASIC Tutorial; April, 1997 http://www.geocities.com/SiliconValley/Pines/1732/tut.htm
7) Perry, Jim (Machaira); VB RPG Tutorial; http://rookscape.com/vbgaming/vbrpgtut/vbrpgtut.php
8) Quatrani, Terry; Visual Modeling With Rational Rose and UML; Copyright © 1998 by Addison Wesley Longman, Inc.

4.2 Pembuatan Aplikasi Server Game

Aplikasi server game selain berfungsi untuk menangani proses login, save, load, serta permintaan ranking, aplikasi ini juga digunakan oleh administrator untuk mendaftarkan pemain baru dan mengatur rancangan state game.

4.2.1 Proses Login

Proses login pada server game ditangani oleh thread Connect yang diciptakan ketika Server*Socket* menerima stream melalui port 9999. Berikut ini penggalan pernyataan untuk proses tersebut.

```
Socket soc=server.accept();
if(port==9999){
      Connect connect=new Connect(soc,serverGame);
}
```

Setelah thread Connect berjalan maka mula-mula diciptakan koneksi ke basis data GameData yang berisi daftar pemain. Kemudian melakukan eksekusi query *"SELECT count(*) AS found,new FROM GameData WHERE login='"+request[1]+"' AND password='"+request[2]+"' GROUP BY new".* Berikut ini adalah pernyataan untuk proses login pada thread Connect.

```
if(request[0].equalsIgnoreCase("login")){
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      con=DriverManager.getConnection("jdbc:odbc:GameData","","");
      Statement statement=con.createStatement();
      ResultSet   rs=statement.executeQuery("SELECT   count(*)   AS   found,new   FROM
GameData WHERE login='"+request[1]+"' AND password='"+request[2]+"' GROUP BY new");
      int found=0;
      boolean newPlay=true;
      while(rs.next()){
            found=rs.getInt(1);
            newPlay=rs.getBoolean(2);
      }
      if(found>0){
            if(newPlay){
                  output.write("found true");
                  output.newLine();
                  output.flush();
                  serverGame.listStatus.add(request[1]+" login as new player");
            }
            else{
                  output.write("found false");
                  output.newLine();
                  output.flush();
                  serverGame.listStatus.add(request[1]+" login as continue player");
            }
      }
      else{
```

```
            output.write("nofound");
            output.newLine();
            output.flush();
            serverGame.listStatus.add(request[1]+" haven't been registered");
      }
      statement.close();
      con.close();
}
```

Seperti terlihat di atas bila login serta password ditemukan dan sesuai pada basis data kemudian dicek statusnya apakah pemain tersebut adalah pemain baru atau pemain yang telah menyimpan data permainannya. Kemudian informasi ditemukannya login pemain tersebut dalam basis data serta statusnya dikirimkan kembali ke game. Bila login tidak ditemukan, server juga akan mengirimkan informasi tidak diketemukannya login pemain tersebut.

4.2.2 Proses Save dan Load

Proses save pada server game ditangani oleh thread ConSave yang diciptakan ketika Server*Socket* menerima stream melalui port 10000. Berikut ini penggalan pernyataan untuk proses tersebut.

```
Socket soc=server.accept();
if(port==10000){
      ConSave conSave=new Connect(soc,serverGame);
}
```

Kemudian setelah thread ConSave berjalan, dibacalah obyek SavePack kiriman dari game yang kemudian dituliskan menjadi sebuah file "*.sav" dengan nama sesuai dengan login pemainnya. Berikut ini adalah pernyataan yang terdapat pada thread ConSave.

```
public void run(){
      try{
            SavePack savePack=(SavePack)objIn.readObject();
            File file=new File("save/"+savePack.login+".sav");
            ObjectOutputStream          objOut=new          ObjectOutputStream(new
FileOutputStream(file));
            objOut.writeObject(savePack);
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            con=DriverManager.getConnection("jdbc:odbc:GameData","","");
            Statement statement=con.createStatement();
            statement.executeUpdate("UPDATE    GameData    SET    new=false    WHERE
login='"+savePack.login+"'");
            statement.close();
            con.close();
            objOut.flush();
```

```
            objOut.close();
            objIn.close();
            socket.close();
            serverGame.listStatus.add(savePack.login+" save game");
      }
      catch(Exception e){
            System.out.println(e.toString());
      }
}
```

Seperti terlihat di atas setelah obyek SavePack disimpan pada file, diciptakan koneksi ke basis data GameData. Dengan query *"UPDATE GameData SET new=false WHERE login='"+savePack.login+"'"* basis data GameData di-update dimana pemain yang telah menyimpan permainannya menjadi berstatus pemain tidak baru lagi.

4.2.3 Pendaftaran Pemain

Pendaftaran pemain baru dilakukan oleh administrator pada form Add Player. Dalam form tersebut administrator memasukkan informasi pemain seperti nama, login, serta password ke dalam basis data GameData.

Berikut ini penggalan pernyataan koneksi ke basis data GameData untuk pendaftaran pemain baru.

```
try{
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      Connection con=DriverManager.getConnection("jdbc:odbc:GameData","","");
      Statement statement=con.createStatement();
      statement.executeUpdate("INSERT                INTO                GameData
VALUES('"+login+"','"+name+"','"+password+"',true)");
      statement.close();
      con.close();
}
catch(Exception sqle){}
```

Seperti terlihat di atas bahwa untuk menambahkan pemain baru mula-mula diciptakan koneksi ke basis data GameData yang kemudian mengeksekusi query *"INSERT INTO GameData VALUES('"+login+"','"+name+"','"+password+"',true)"*. Login, nama, serta password pemain dimasukkan ke dalam basis data GameData. Pada akhir

pernyataan query diset pula bahwa pemain yang didaftarkan tersebut berstatus pemain baru.

4.2.4 Pengaturan Rancangan State

Untuk menciptakan skenario rancangan state baru, diciptakanlah obyek vector vState baru yang nantinya digunakan untuk menampung obyek SetState yang disimpan dalam file "*.sta" dengan nama yang ditentukan oleh administrator pada file dialog. Berikut ini adalah pernyataan ketika menciptakan file skenario rancangan state baru.

```
Vector vState=new Vector();
FileDialog fDialog=new FileDialog(this,"Create New");
fDialog.setDirectory("state/");
fDialog.setFile("*.sta");
fDialog.show();
if(fDialog.getFile()!=null&&fDialog.getFile().substring(fDialog.getFile().length()-
3).equalsIgnoreCase("sta")){
      tState.setText(Integer.toString(vState.size()));
      tVector.setText(fDialog.getFile());
      dirState=fDialog.getDirectory();
      try{
            File file=new File(dirState+tVector.getText());
            ObjectOutputStream           objOut=new           ObjectOutputStream(new
FileOutputStream(file));
            objOut.writeObject(vState);
            objOut.flush();
            objOut.close();
      }
      catch(IOException ioe){}
      btnAdd.setEnabled(true);
}
```

Administrator bisa memilih untuk menambah *town* atau *dungeon* pada skenario rancangan state tersebut dengan posisi state yang ditentukan oleh administrator pada image peta dunia.

Pada form untuk menambah *town*, hal-hal yang bisa ditambahkan oleh administrator ke dalam *town* antara lain :

1. Nama *town* serta statusnya apakah terkunci dan terlihat.
2. Image *town* beserta pemetaan areanya.
3. Posisi awal hero.