

**UNIVERSITAS INDONESIA**

**PENERAPAN *SOFTWARE PRODUCT LINE* DALAM PENGEMBANGAN *ROLE-PLAYING GAME***

**SKRIPSI**

**FARIZ ABIRAFDI BENARTO**

**1206278416**

**FAKULTAS ILMU KOMPUTER**

**PROGRAM STUDI ILMU KOMPUTER**

**DEPOK**

**JUNI 2016**



**UNIVERSITAS INDONESIA**

**PENERAPAN *SOFTWARE PRODUCT LINE* DALAM PENGEMBANGAN *ROLE-PLAYING GAME***

**SKRIPSI**

**Diajukan sebagai salah satu syarat untuk memperoleh gelar**

**Sarjana Ilmu Komputer**

**FARIZ ABIRAFDI BENARTO**

**1206278416**

**FAKULTAS ILMU KOMPUTER**

**PROGRAM STUDI ILMU KOMPUTER**

**DEPOK**

**JUNI 2016**

**HALAMAN PERSETUJUAN**

**Judul** : Penerapan *Software Product Line* dalam Pengembangan *Role-Playing*

*Game*

**Nama** : Fariz Abirafdi Benarto

**NPM** : 1206278416

Laporan Skripsi ini telah diperiksa dan disetujui.

20 Juni 2016

Dr. Ade Azurat

Pembimbing Skripsi

**HALAMAN PERNYATAAN ORISINALITAS**

**Skripsi ini adalah hasil karya saya sendiri,**

**dan semua sumber baik yang dikutip maupun dirujuk**

**telah saya nyatakan dengan benar.**

**Nama : Fariz Abirafdi Benarto**

**NPM : 1206278416**

**Tanda Tangan :**

**Tanggal : 20 Juni 2016**

**HALAMAN PENGESAHAN**

Skripsi ini diajukan oleh :

Nama : Fariz Abirafdi Benarto

NPM : 1206278416

Program Studi : Ilmu Komputer

Judul Skripsi : Penerapan *Software Product Line* dalam

Pengembangan *Role-Playing Game*

Telah berhasil dipertahankan di hadapan Dewan Penguji dan diterima sebagai bagian persyaratan yang diperlukan untuk memperoleh gelar Sarjana Komputer pada Program Studi Ilmu Komputer, Fakultas Ilmu Komputer, Universitas Indonesia

DEWAN PENGUJI

Pembimbing : ................... ( .....tanda tangan......)

Penguji : ................... ( .....tanda tangan......)

Penguji : ................... ( .....tanda tangan......)

Ditetapkan di : Depok

Tanggal : ...........

KATA PENGANTAR

Puji syukur penulis panjatkan kehadirat Allah SWT, Tuhan Yang Maha Esa, karena berkat Rahmat dan Karunia-Nya penulis dapat menyelesaikan skripsi yang berjudul “Penerapan *Software Product Line* dalam Pengembangan *Role-Playing Game*”. Skripsi ini ditulis dalam rangka memenuhi salah satu syarat bagi penulis untuk dapat menyelesaikan pendidikan dan memperoleh gelar Sarjana Ilmu Komputer dari Jurusan Ilmu Komputer Universitas Indonesia.

Penulis menyadari bahwa skripsi ini tidak akan selesai tanpa bantuan dan dukungan dari berbagai pihak. Oleh karena itu, penulis ingin mengucapkan terima kasih kepada:

1. Bapak Dr. Ade Azurat selaku dosen pembimbing yang telah memberikan banyak sekali arahan, panduan, kritik, serta saran bagi penulis di tengah kesibukannya dan padatnya jadwal agar skripsi ini dapat diselesaikan.
2. Kedua orang tua, Mami dan Ayah, serta adik penulis, Farrel, atas bantuannya dalam berbagai bentuk mulai dari dukungan moral hingga material, serta untuk doa dan dukungan yang senantiasa diberikan kepada penulis.
3. Akhmad Agosto Prasetyo, Ilham Kusuma, Rahmat Hidayat, dan Rifqi Adiwidjaja selaku sahabat-sahabat penulis yang telah menemani penulis selama masa perkuliahan di Fakultas Ilmu Komputer Universitas Indonesia serta segala bentuk bantuan dan dukungan yang telah diberikan kepada penulis.
4. Muhammad Fadhlika selaku sahabat penulis sejak Sekolah Menengah Pertama yang selalu menjadi teman yang baik untuk bertukar pikiran dan pandangan selama 10 tahun terakhir.
5. Givana Sandita yang senantiasa memberikan dukungan kepada penulis, membantu penulis belajar untuk berbagai mata kuliah, menemani penulis mengerjakan tugas, serta senantiasa menemani penulis dalam penulisan skripsi ini.
6. Teman-teman sekelompok PPL C08 (Novie Kamalia, Ahmad Ghofari) termasuk Ilham Kusuma dan Givana Sandita yang telah menjadi lebih dari sekedar teman kelompok mata kuliah dan telah memberikan banyak bantuan dan dukungan kepada penulis sampai sekarang.
7. Pinta Kumbarani Nasution, Fandika Okdiba, dan Bagus Bagaskara selaku saudara-saudara sepupu penulis yang telah memberikan dukungan serta masukan kepada penulis selama penulisan skripsi ini.
8. ASTRO 2012 yang telah menjadi bagian dari hidup penulis selama menjalani masa perkuliahan.
9. Teman-teman yang pernah menjadi penghuni kos Graha Bouraq (Mas Pur, Mikael, Mas Pudang, Ardi, Japri) serta Tante dan Om Anshor selaku penjaga kos Graha Bouraq yang telah menemani penulis selama masa kos penulis.
10. Seluruh pihak yang tidak tersebutkan namanya, yang telah memberikan bantuan dan dukungan kepada penulis baik secara langsung maupun tidak.

Akhir kata, penulis berharap Allah SWT berkenan membalas segala kebaikan semua pihak yang telah membantu. Penulis menyadari bahwa masih terdapat banyak sekali kekurangan dalam penulisan skripsi ini. Penulis berharap semoga skripsi ini dapat membuka pintu bagi penelitian-penelitian lain yang serupa untuk dilakukan oleh mahasiswa-mahasiswa lain, serta menjadi acuan yang baik bagi mereka yang meneruskan penelitian ini.

Depok, 20 Juni 2016

Fariz Abirafdi Benarto

HALAMAN PERNYATAAN PERSETUJUAN PUBLIKASI

TUGAS AKHIR UNTUK KEPENTINGAN AKADEMIS

Sebagai sivitas akademik Universitas Indonesia, saya yang bertanda tangan di bawah ini:

Nama : Fariz Abirafdi Benarto

NPM : 1206278416

Program Studi : Ilmu Komputer

Fakultas : Ilmu Komputer

Jenis Karya : Skripsi

demi pengembangan ilmu pengetahuan, menyetujui untuk memberikan kepada Universitas Indonesia **Hak Bebas Royalti Noneksklusif (*Non-exclusive Royalty-Free Right*)** atas karya ilmiah saya yang berjudul:

Penerapan *Software Product Line* dalam Pengembangan *Role-playing Game*

beserta perangkat yang ada (jika diperlukan). Dengan Hak Bebas Royalti Non-eksklusif ini Universitas Indonesia berhak menyimpan, mengalihmedia/formatkan, mengelola dalam bentuk pangkalan data (*database*), merawat, dan memublikasikan tugas akhir saya selama tetap mencantumkan nama saya sebagai penulis/pencipta dan sebagai pemilik Hak Cipta.

Demikian pernyataan ini saya buat dengan sebenarnya.

Dibuat di : Depok

Pada tanggal : 20 Juni 2016

Yang menyatakan

(.......................................................)

**ABSTRAK**

Nama : Fariz Abirafdi Benarto

Program Studi : Ilmu Komputer

Judul : Penerapan *Software Product Line* dalam Pengembangan *Role-playing Game*

Skripsi ini membahas penerapan *Software Product Line* (SPL) dalam pengembangan *game* bergenre RPG (*role-playing game)*. *Game* yang digunakan sebagai obyek studi kasus dalam skripsi ini adalah Dungeon Crawl Stone Soup. Penelitian dilakukan dengan membuat dua buah versi *game* yang memiliki perbedaan konfigurasi dalam aspek atau fitur tertentu, namun tetap mempertahankan fitur-fitur inti sebagai kesamaan. Penelitian ini bertujuan untuk menilai dampak dari penerapan SPL dalam pengembangan *game*, berdasarkan penelitian-penelitian sebelumnya yang digunakan sebagai acuan dalam skripsi ini.

Kata kunci:

*Software product line*, *game development, role-playing game, software engineering*

**ABSTRACT**

Name : Fariz Abirafdi Benarto

Study Program: Computer Science

Title : Software Product Line Implementation in Role-playing Game Development

This research discusses the implementation of Software Product Line (SPL) in Role-playing Game development. The game used as case study object is Dungeon Crawl Stone Soup. The Research is conducted by creating two versions of the game which have differences in certain aspects and features, while maintaining core features as a commonality. The goal of this research is to assess the effects of SPL implementation in game development, based on previous works used as references in this research.

Keywords:

*Software product line*, *game development, role-playing game, software engineering*

**DAFTAR ISI**

HALAMAN JUDUL

HALAMAN PERSETUJUAN

HALAMAN PERNYATAAN ORISINALITAS

HALAMAN PENGESAHAN

KATA PENGANTAR

HALAMAN PERSETUJUAN PUBLIKASI ILMIAH

ABSTRAK

ABSTRACT

DAFTAR ISI

DAFTAR GAMBAR

**1. PENDAHULUAN**

1.1 Latar Belakang

1.2 Rumusan Masalah

1.3 Tujuan Penelitian

1.4 Manfaat Penelitian

1.5 Ruang Lingkup Penelitian

1.6 Metodologi Penelitian

1.7 Sistematika Penulisan

**2. TINJAUAN LITERATUR**

2.1 *Game Domain Analysis*

2.1.1 Rangkuman *Paper*

2.1.2 Penggunaan *Paper*

2.2 Eksperimen dengan *Code*

2.2.1 Rangkuman *Paper*

2.2.2 Penggunaan *Paper*

**3. STUDI KASUS**

3.1 Rangkuman studi kasus

3.1.1 *Game*

3.1.1.1 Ringkasan Umum

3.1.1.2 Genre and Mekanika *Gameplay*

3.1.1.3 Grafis

3.1.1.4 *Interface*

3.1.1.5 *Controls*

3.1.1.6 *Audio*

3.1.1.7 Platform

3.1.1.8 *Availability*

3.1.1.9 Lisensi

3.1.2 *Source Code*

3.1.2.1 Ringkasan Umum

3.1.2.2 *Repository*

3.1.2.3 Bahasa

3.1.2.4 *Packages*

3.1.2.5 *Project Hierarchy*

3.1.2.6 Instalasi dan *Building*

3.1.2.7 *Dependencies*

3.2 *Game Domain Analysis*

3.2.1 Membayangkan dan Menganalisis *Game Domain*

3.2.2 Mengidentifikasi *Emotion-based Requirements*

3.2.3 *Nonemotional Requirements*

3.2.4 Mengambil Contoh dari Domain

3.2.5 Menentukan dan Mengkaji Ulang Fitur *Game Domain*

3.2.6 Membahas Fitur Game yang Terkunci

3.2.7 Mengidentifikasi Beberapa Subdomain

3.2.8 Mengantisipasi Fitur yang Akan Datang

**4. EKSPERIMEN DAN ANALISIS**

4.1 Strategi untuk *Refactoring*

4.2 *Commonalities* dan *Variabilities*

4.2.1 *Commonalities*

4.2.2 *Variabilities*

4.3 Perbedaan dalam *Code*

4.3.1 *Refactoring* Umum

4.3.2 *Refactoring* Berdasarkan *Commonality* dan *Variability*

4.4 Analisis

**5. PENUTUP**

5.1 Kesimpulan

5.2 Saran

**DAFTAR REFERENSI**

**LAMPIRAN *SOURCE CODE***

**DAFTAR GAMBAR**

Gambar 2.1 Diagram ketersediaan *game engine* dan pendekatan *domain-specific*

Gambar 3.1 Diagram fitur RPG secara umum

Gambar 3.2 Diagram fitur Dungeon Crawl Stone Soup

Gambar 3.3 Far Cry (2004) menggunakan grafis 3D

Gambar 3.4 Minecraft (2010) menggunakan grafis resolusi rendah

Gambar 3.5 *Nonemotional Requirements*

Gambar 4.1 Bentuk *code* sistem skor sebelum pengubahan

Gambar 4.2 Bentuk *code* sistem skor setelah pengubahan

**BAB 1**

**PENDAHULUAN**

Bab ini menjelaskan latar belakang, rumusan masalah, tujuan penelitian, manfaat penelitian, ruang lingkup penelitian, dan sistematika penulisan.

**1.1 Latar Belakang**

Industri *game* video telah berkembang sejak tahun 1970-an. Pada masa itu, biaya pengembangan dan kapasitas komputer masih rendah, sehingga *game* dapat dibuat oleh seorang *programmer*.

Memasuki era modern, industri *game* telah berkembang pesat, dari produksi berbiaya rendah dengan ukuran tim pengembang yang kecil, hingga produksi berbiaya jutaan *dollar* dengan tim yang beranggotakan ratusan orang. Proyek *game* berukuran besar melibatkan banyak orang dengan berbagai peran dan spesifikasi pekerjaan, termasuk para *programmer*. Sebuah *game* AAA, atau *game* yang diproduksi dengan biaya dan promosi yang sangat tinggi, biasanya melibatkan banyak sekali *programmer*.

Beban kerja pengembang *game* sering kali ditentukan oleh genre dari *game* yang sedang dibuat. Salah satu *genre* yang paling berat untuk dikembangkan adalah genre *role-playing game*, melihat karakteristik dari genre tersebut yang biasanya memiliki banyak sekali aspek dan fitur yang berbeda.

Banyaknya *code* yang dikerjakan dan *programmer* yang berkontribusi sering kali menyebabkan *project hierarchy* tidak beraturan. Berbagai masalah sering dihadapi ketika berursan dengan *source code* yang besar, contohnya adalah ketika membuat sebuah perubahan pada *code* yang sudah ada. Mengubah *source code* yang hanya terdiri dari ratusan baris dan beberapa *file* mungkin bukan perkara sulit, tetapi ketika *code* yang dihadapi berjumlah puluhan hingga ratusan *file*, dengan tiap *file* memiliki hingga ribuan baris *code*, tentu akan menjadi sebuah pekerjaan yang rumit dan melelahkan. Banyak risiko yang dapat terjadi ketika mengubah *code*, salah satunya adalah kemungkinan bahwa terdapat *method* atau *class* yang gagal berfungsi karena perubahan yang dilakukan. Masalah akan menjadi semakin sulit untuk diatasi ketika *method* atau *class* tersebut berada dalam *file* yang berbeda dengan *file* yang diubah.

Dari kesulitan-kesulitan tersebut, dibutuhkan sebuah sistem yang mampu mengurangi beban kerja *programmer* dengan membuat struktur *code* yang lebih terorganisasi, sehingga perubahan pada *code* dapat dilakukan dengan mudah dan lancar. *Software product line*, atau SPL, muncul sebagai sebuah paradigma *development* yang didesain untuk mengoptimalkan waktu pemasaran, biaya, produktivitas, kualitas, dan aspek *development* lainnya. Penerapan SPL dalam pembuatan *game* dapat membantu memudahkan pengerjaan dengan menekankan struktur *code* yang baik, sehingga dapat mengurangi waktu produksi dan biaya yang disebabkan oleh proses pengubahan *code*.

**1.2 Rumusan Masalah**

Terdapat dua rumusan masalah yang dapat dijawab oleh penelitian ini, yaitu:

1. Bagaimana SPL dapat diterapkan dalam pengembangan *game* ?
2. Apa manfaat dari penerapan SPL dalam pengembangan *game* ?

**1.3 Tujuan Penelitian**

Berdasarkan pertanyaan-pertanyaan rumusan masalah, terdapat beberapa tujuan yang diharapkan dapat diraih oleh penelitian ini, yaitu:

1. Mengetahui metode yang dapat dilakukan untuk menerapkan SPL dalam pengembangan *game*.
2. Menilai dampak dari penerapan SPL pada proses pengembangan *game*.

**1.4 Manfaat Penelitian**

Penelitian ini diharapkan dapat menjadi acuan bagi penelitian-penelitian yang akan datang berikutnya, serta membantu menerapkan sebuah standar dalam proses pengembangan *game*.

**1.5 Ruang Lingkup Penelitian**

Penelitian ini bertujuan untuk mendalami penerapan SPL dalam pengembangan *game* secara umum, namun dalam pembahasan dan eksperimen penelitian ini berfokus pada *game* dengan genre RPG. Eksperimen penerapan SPL dalam studi kasus secara umum meliputi beberapa fitur yang akan dijelaskan lebih lanjut dalam bab Eksperimen dan Analisis.

**1.6 Metodologi Penelitian**

Terdapat beberapa tahapan penelitian yang dilakukan dalam skripsi ini, yaitu:

1. Tinjauan Literatur

Dalam tahapan ini, penelitian-penelitian terkait dengan penelitian yang dilakukan dalam skripsi ini dipelajari dan dianalisis untuk dijadikan sebagai acuan dalam eksperimen.

1. Analisis Studi Kasus

Untuk mendalami lebih lanjut mengenai *game* studi kasus, dilakukan analisis terhadap berbagai aspek dari *game* studi kasus.

1. Eksperimen dengan studi kasus

Dalam tahapan ini, dilakukan eksperimen pada *game* studi kasus berdasarkan informasi yang didapat dari tinjauan literatur dan analisis studi kasus.

1. Analisis hasil eksperimen

Dari eksperimen yang dilakukan, hasil yang didapat dicatat dan dianalisis mengenai kesesuaiannya dengan rumusan masalah, tinjauan literatur, serta analisis studi kasus.

1. Menarik kesimpulan

Dalam tahapan terakhir, yang dilakukan adalah menilai hasil dari eksperimen dan membuat kesimpulan sesuai dengan rumusan masalah.

**1.7 Outline**

Skripsi ini disusun menjadi lima bab dengan sistematika penulisan sebagai berikut:

* Bab 1 Pendahuluan

Bab ini menjelaskan latar belakang, rumusan masalah, tujuan penelitian, manfaat penelitian, ruang lingkup penelitian, metodologi penelitian dan sistematika penulisan.

* Bab 2 Tinjauan Literatur

Bab ini berisi tinjauan literatur yang digunakan sebagai acuan dari penelitian.

* Bab 3 Studi Kasus

Bab ini mendeskripsikan *game* yang digunakan sebagai studi kasus untuk di-*refactor* dalam penelitian.

* Bab 4 Eksperimen dan Analisis

Bab ini menjelaskan rincian eksperimen yang dilakukan, beserta penjelasan dasar-dasar dari langkah yang dilakukan.

* Bab 5 Kesimpulan

Bab ini berisi kesimpulan yang dapat ditarik dari hasil eksperimen dan analisis dalam penelitian, serta saran untuk penelitian selanjutnya.

**BAB 2**

**TINJAUAN LITERATUR**

Bab ini berisi tinjauan literatur yang digunakan sebagai acuan dari penelitian. Terdapat dua buah *paper* yang akan dijadikan acuan utama dalam penelitian skripsi ini.

**2.1 *Game Domain Analysis***

Proses analisis domain *game* untuk penelitian ini dilakukan mengacu pada *paper* yang ditulis oleh Andre W.B. Furtado, Andre L.M. Santos, Geber L. Ramalho, dan Eduardo Santana de Almeida yang berjudul Improving Digital Game Development with Software Product Lines.

**2.1.1 Rangkuman *Paper***

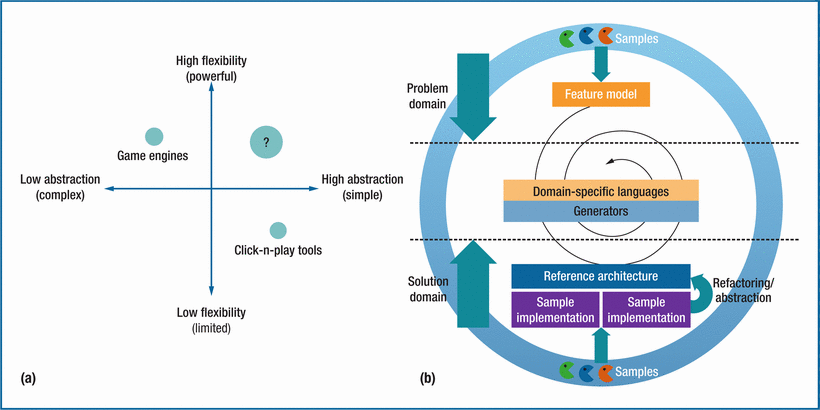
Dalam papernya, Furtado dkk mengatakan bahwa *software reuse*, termasuk *software product lines* (SPL), membuat rekayasa perangkat lunak lebih efektif dan produktif. Strategi dan paradigma yang dibawa oleh SPL membuat *developer* perangkat lunak lebih menganalisis dan mengimplementasi sistem secara kolektif daripada secara terpisah, serta lebih mengautomasi *software life cycle*.

Meskipun domain-domain dalam bidang lain telah sukses menerapkan konsep SPL, penerapannya dalam pengembangan *game* masih terhadang beberapa kendala. Kendala-kendala tersebut di antaranya adalah sulitnya proses rekayasa dalam pengembangan *game*, kesulitan integrasi komponen dan mengelola kompleksitas arsitektural, serta perbedaan proses antara pengembangan *game* dengan pengembangan perangkat lunak secara umum.

Untuk dapat menerapkan SPL dengan baik, Furtado dkk menjelaskan sebuah pendekatan berbasis SPL untuk menganalisis domain *game* dan mengimplementasi *domain assets* inti. Pendekatan ini bertujuan untuk membantu *developer* membuat DSL (*domain specific languages*) dan *generators*, aspek kunci SPL yang masih kurang didalami dalam konteks *game development*.

*Paper* ini membahas mengenai konsep automasi pengembangan *game*, di mana penulis paper juga membahas mengenai ketersediaan sumber daya untuk mengembangkan *game*. *Game engines* merupakan *tools* yang sangat berguna, namun rendahnya abstraksi dari *game engine* membuat *tool* tersebut cukup rumit untuk digunakan. Di sisi lain, terdapat banyak *tools* yang lebih sederhana dan mudah digunakan seperti *tools* yang memiliki sistem “*click-n-play*”, di mana pengguna *tools* hanya perlu men-*click* tombol-tombol yang disediakan, tanpa harus melakukan *scripting*. *Tools* seperti ini memiliki tingkat fleksibilitas yang rendah, karena aktivitas yang dapat dilakukan pengguna pun terbatas, sehingga tidak banyak yang dapat dilakukan pula oleh *tools* tersebut.

Kurangnya ketersediaan *tools* yang sederhana namun memiliki banyak kegunaan menghambat realisasi konsep automasi pengembangan *game*.



Gambar 2.1 Diagram ketersediaan *game engine* dan pendekatan *domain-specific*

Proses-proses SPL sering kali menargetkan pengembangan perangkat lunak secara umum, tetapi proses-proses tersebut sebenarnya dapat menjadi lebih efektif ketika dibatasi oleh domain makro. Furtado dkk membuat sebuah pendekatan *domain-specific* untuk pengembangan *game* yang menggunakan *game domain analysis* untuk membuat *core assets* seperti DSL dan *reference architectures* untuk SPL dari sebuah *game*.

Dalam *paper* ini, dilakukan sebuah eksperimen terhadap *game* bergenre *arcade* menggunakan sebuah *tool* SPL bernama ArcadEx. ArcadEx digunakan sebagai *tool* untuk membantu penerapan pendekatan *domain-specific language* pada pengembangan *game* bergenre *arcade* dua dimensi.

**2.1.2 Penggunaan *Paper***

*Paper* ini akan digunakan sebagai acuan dalam menganalisis *game* yang akan diuji. Hasil analisis studi kasus yang mengacu pada pendekatan *domain-specific* dari *paper* ini akan digunakan untuk menentukan *commonality* dan *variability* dari variasi yang akan dibuat.

**2.2 Eksperimen dengan *Code***

Eksperimen yang dilakukan dalam penelitian ini mengacu pada langkah-langkah dan panduan *refactoring* yang dikemukakan oleh Weishan Zhang dan Stan Jarzabek dalam *paper* yang berjudul Reuse without Compromising Performance: Industrial Experience from RPG Software Product Line for Mobile Devices.

**2.2.1 Rangkuman *Paper***

Paper ini melaporkan sebuah proyek industri perangkat *mobile* di mana pendekatan *product line* tidak hanya mendapatkan manfaat produktivitas *development* dan *maintenance*, tetapi juga *performance* perangkat lunak yang lebih baik. Penulis paper menerapkan pendekatan *extractive*, yaitu pendekatan yang me-*reuse* satu atau lebih produk perangkat lunak sebagai dasar dari *product line*. Fokus utama dari penelitian paper ini adalah penerapan SPL dalam pengembangan RPG untuk perangkat *mobile*, di mana penulis paper merumuskan RPG *product line architecture* (RPG-PLA). Untuk membantu penerapan RPG-PLA, digunakan sebuah *tool* untuk *build* dan mengelola RPG-PLA, yaitu XVCL.

Eksperimen yang dilakukan dalam *paper* ini adalah membandingkan dua buah tim pengembang *game* dengan kemampuan sebanding menggunakan metode yang berbeda. Tim pertama mengembangkan *game* menggunakan *tools* XVCL dan menerapkan SPL, sedangkan tim lainnya tidak menerapkan SPL. Dalam *paper* ini, dipaparkan bahwa tim yang menerapkan SPL dapat menyelesaikan proyek lebih cepat dibanding tim lain.

**2.2.2 Penggunaan *Paper***

*Paper* ini akan digunakan sebagai acuan dalam hal-hal yang lebih bersifat teknis, seperti pembuatan *feature model* dari *game* studi kasus dan langkah-langkah *refactoring*.

**BAB 3**

**STUDI KASUS**

Bab ini mendeskripsikan *game* yang digunakan sebagai studi kasus untuk di-*refactor* dalam penelitian.

**3.1 Rangkuman studi kasus**

Dalam subbab ini akan dijelaskan mengenai detail dari *game* studi kasus yang akan digunakan dalam eksperimen. Penjelasan akan dilakukan untuk sisi permainan dan struktur *project*.

**3.1.1 *Game***

Subbab ini menjelaskan *game* studi kasus dilihat sebagai sebuah *game* atau aplikasi.

**3.1.1.1 Ringkasan Umum**

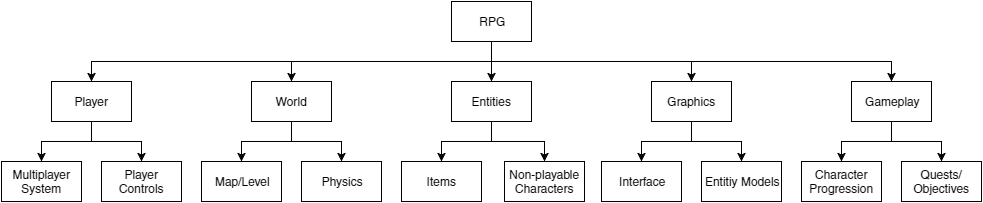
*Game* yang digunakan sebagai obyek studi kasus adalah Dungeon Crawl Stone Soup. Dungeon Crawl Stone Soup merupakan sebuah *game* yang pertama kali dikembangkan pada tahun 2006 sebagai upaya untuk menghidupkan kembali proyek *Linley’s Dungeon Crawl* yang dikembangkan oleh Linley Henzell pada tahun 1997.

Dungeon Crawl Stone Soup dapat dimenangkan dengan *win condition* tertentu, yaitu mendapatkan 3 dari 15 buah benda bernama *runes of zot*, mengambil sebuah benda bernama *orb of zot*, dan keluar dari *dungeon* dengan selamat.

**3.1.1.2 Genre dan Mekanika *Gameplay***

Pada umumnya, sebuah *game* dapat dikategorikan ke dalam sebuah genre. Genre dari sebuah *game* adalah jenis, tipe, atau kelompok dari *game* tersebut berdasarkan beberapa hal seperti cerita, mekanisme *gameplay*, model grafis, jumlah pemain, dan lain-lain. Dungeon Crawl Stone Soup dapat dikategorikan dalam genre RPG, atau *role-playing game*. RPG sendiri dapat didefinisikan sebagai sebuah *game* di mana pemain mengimitasi atau memerankan sebuah tokoh atau peran fiktif dalam sebuah cerita atau dunia, biasanya berlatar fantasi. Aslinya, RPG tidak terikat dalam *game* video. RPG berawal dari permainan papan (*board game*) atau permainan berbasis teks (*text-based*). Dalam studi kasus ini, RPG yang dimaksud adalah RPG dalam arti *game* video, di mana elemen-elemen *role-playing* yang terdapat dalam *game* tersebut berupa latar dan sistem yang terkomputerisasi.

*Game* yang termasuk dalam kategori genre RPG biasanya memiliki beberapa ciri khusus seperti pemain memainkan sebuah karakter, terdapat dunia—dinamis atau statis—yang dapat dijelajahi pemain, dan memiliki *character progression*. Dalam Dungeon Crawl Stone Soup, pemain memainkan sebuah tokoh yang memiliki peran tertentu, di mana tokoh tersebut dapat menjelajahi sebuah dunia yang dinamis. Setiap aksi signifikan yang dilakukan oleh pemain menghasilkan elemen-elemen tertentu yang secara langsung mengubah karakteristik tokoh yang dimainkan. Hal tersebut merupakan sebuah sistem *character progression*.

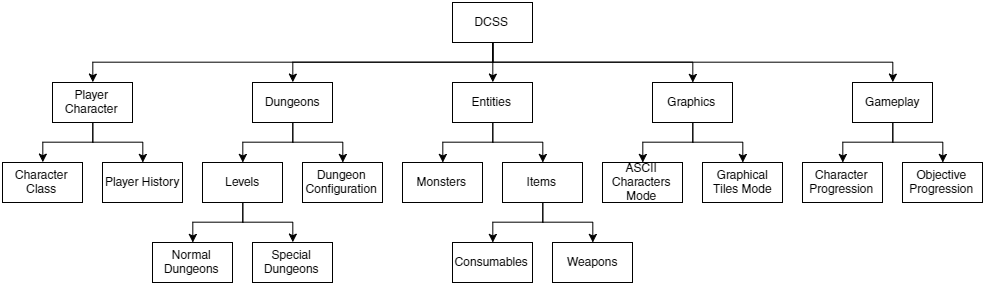


Gambar 3.1 Diagram fitur RPG secara umum

Dalam genre RPG sendiri terdapat beberapa sub-genre yang mengkategorikan sistem permainan dan elemen *role-playing* lebih lanjut. Pembagian sub-genre untuk RPG pun terbagi menjadi beberapa dasar pengelompokan, seperti berdasarkan unsur atau elemen *game* yang paling menonjol, model skenario permainan, jumlah pemain, dan lain-lain. Dungeon Crawl Stone Soup dapat dikategorikan ke dalam sub-genre *roguelike*.

*Roguelike* merupakan sub-genre dari RPG yang mengandung unsur *dungeon crawler* dan biasanya terdiri dari sejumlah *level* yang dihasilkan secara prosedural. Beberapa karakteristik *roguelike* adalah *turn-based gameplay*, grafis *tile-based*, dan *permanent death* dari karakter yang dimainkan. Salah satu ciri khas *roguelike* adalah grafis yang ditampilkan dengan karakter ASCII dan hanya dapat dimainkan menggunakan *keyboard*, walaupun beberapa *game* bersub-genre *roguelike* modern telah menyediakan fitur *graphical tile* yang menggunakan gambar untuk merepresentasikan elemen-elemen dalam *game*. *Game* dengan sub-genre *roguelike* biasanya lebih mengedepankan kompleksitas mekanik *gameplay* dibandingkan dengan *interface* yang ramah dengan pemain. *Turn-based gameplay* pada *roguelike* biasanya merupakan sistem *real-time* yang dimodifikasi sehingga waktu dalam dunia *game* tersebut hanya akan berjalan jika pemain melakukan suatu aktivitas. Dalam Dungeon Crawl Stone Soup, setiap aktifitas dilakukan dengan satu ketikan tombol di *keyboard* atau *mouse*. Lama waktu yang berjalan dalam satu ketikan tersebut bergantung pada lama waktu aktivitas yang dilakukan. Selain *turn-based gameplay*, Dungeon Crawl Stone Soup juga memiliki sistem *permanent death*, di mana karakter yang terbunuh dalam *game* akan hilang selamanya, tanpa bisa di-*load* kembali seperti dalam *game* bergenre lain. Walaupun karakter yang telah terbunuh tidak dapat di-*load*, pemain dapat melakukan *save* dan *load* untuk karakter yang masih hidup, untuk menyimpan dan melanjutkan *progress* dalam *game*.

Unsur *dungeon crawl* yang terdapat dalam *roguelike* merupakan sebuah sistem atau skenario di mana tujuan atau aktivitas utama pemain adalah menjelajahi sebuah atau sejumlah *dungeon*. Dalam *dungeon* yang dijelajahi, pemain dapat bertarung dengan karakter-karakter musuh dan mengumpulkan benda-benda yang memiliki signifikansi dalam sistem *character progression*. *Game* dengan model skenario *dungeon crawl* biasanya memiliki sistem *loot*, di mana pemain dapat mengumpulkan benda atau barang-barang yang tersedia di dalam dunia *game* tersebut dengan berbagai cara, mulai dari mengambil barang yang tersedia di tempat-tempat tertentu seperti misalnya emas di dalam peti harta, atau benda yang didapat dari karakter musuh yang telah dikalahkan.



Gambar 3.2 Diagram fitur Dungeon Crawl Stone Soup

Seperti kebanyakan *game* dengan genre *roguelike*, Dungeon Crawl Stone Soup memiliki latar fantasi, di mana dunia yang dijelajahi merupakan sebuah dunia fiktif dengan karakter-karakter yang berasal dari berbagai mitologi. Dalam Dungeon Crawl Stone Soup, unsur cerita tidak terlalu ditonjolkan, hanya disebutkan mengenai tujuan permainan yang merupakan *win condition* dari *game* ini. Tingkah laku pemain tidak memiliki dampak terhadap cerita latar dan sebaliknya, cerita latar tidak memiliki dampak apapun terhadap permainan, kecuali sebagai penjelas *win condition*. Pemain tidak dapat mengubah cerita dan *win condition* dengan cara apapun.

Setiap sesi permainan Dungeon Crawl Stone Soup memiliki konfigurasi *dungeon* yang berbeda. Fitur ini dibuat agar *game* memiliki *replay value* yang tinggi, sehingga pemain tidak merasa bosan. Fitur *random/procedurally-generated levels* di *game-game* *roguelike* juga dimaksudkan agar pemain tidak terdorong untuk menghafal pola *game* dan lebih menitikberatkan permainan pada strategi dan mempelajari kesalahan dari sesi-sesi sebelumnya.

Selain *dungeon level* yang *procedurally-generated*, fitur lain yang menaikkan *replay value* dan mengurangi unsur penghafalan pola adalah *scroll* dan *potion* yang memiliki efek yang di-*random* setiap sesinya. *Scroll* dan *potion* adalah salah satu elemen penting dalam *game* bertema fantasi dengan sistem *dungeon crawl*. *Scroll* merupakan gulungan perkamen yang memiliki efek sihir tertentu, sedangkan *potion* merupakan minuman obat atau pada kasus tertentu merupakan minuman racun. Dengan sistem seperti ini, pemain akan menjadi lebih berhati-hati ketika menggunakan *scroll* atau *potion*, karena harus memperhitungkan risiko yang terjadi jika benda yang digunakan memiliki efek samping yang buruk.

**3.1.1.3 Grafis**

Dungeon Crawl Stone Soup memiliki dua mode grafis, yaitu *console* dan *graphical tile*. Dalam mode *console*, *game* dijalankan di sebuah *console* dansemua elemen *game* direpresentasikan dalam karakter ASCII. Dalam mode *graphical tile*, *game* dijalankan dalam tampilan *user interface* interaktif, di mana semua elemen *game* memiliki representasi grafis berupa gambar 2D yang menyerupai deskripsi obyek yang direpresentasikan. Karena sistem yang digunakan adalah *tile-based*, maka elemen-elemen atau obyek yang berukuran lebih dari satu *tile* digambarkan melalui gabungan dari beberapa *tile*, bukan merupakan sebuah gambar tunggal.

**3.1.1.4 *Interface***

Mode *graphical tile* menyediakan tampilan *interface* untuk *inventory* pemain yang dapat diakses dan dilihat dengan mudah, langsung saat bermain. *Interface* mode *graphical tiles* secara umum memiliki lebih banyak tampilan, di mana tampilan yang terdapat di layar *game* tidak hanya karakter pemain dan dunia di sekitarnya, tetapi juga peta dari *level* dungeon yang sedang dijelajahi, visualisasi daftar *inventory* atau barang-barang yang dimiliki oleh pemain, daftar *skill* atau jurus yang dapat dilakukan oleh pemain, dan daftar *command* untuk membuat karakter pemain melakukan kegiatan tertentu secara otomatis.

**3.1.1.5 *Controls***

Masing-masing mode grafis yang disediakan Dungeon Crawl Stone Soup memiliki mekanisme *controls* yang berbeda. Mode *console* hanya dapat dikendalikan menggunakan *keyboard*, sedangkan mode *graphical tile* dapat dikendalikan dengan menggunakan *keyboard* dan *mouse*.

**3.1.1.6 Audio**

Dungeon Crawl Stone Soup menyediakan fitur suara yang dapat diaktifkan dengan mengubah *file* konfigurasi secara manual. Secara *default*, fitur suara tidak diaktifkan.

**3.1.1.7 Platform**

Untuk saat ini, Dungeon Crawl Stone Soup tersedia untuk *operating system* Windows, Mac OS X, dan Linux. Pengembangan *game* untuk platform Android saat ini masih berlangsung, namun *unstable build* terbaru dapat diunduh dari situs resmi Dungeon Crawl Stone Soup. Dungeon Crawl Stone Soup dapat dimainkan secara *local*, melalui *executable file* di platform Windows dan melalui *terminal* di platform Linux. Selain *local*, tersedia juga versi *web browser* yang menyediakan Dungeon Crawl Stone Soup. Versi yang disediakan di *web browser* adalah *graphical tile version*.

Untuk eksperimen, pengerjaan dilakukan pada platform Linux karena kemudahan dalam proses *installation* dan *compile*.

**3.1.1.8 *Availability***

Dungeon Crawl Stone Soup dapat diunduh secara gratis melalui situs resmi. Situs Dungeon Crawl Stone Soup juga menyediakan tautan untuk mengakses dan mengunduh repositori *source-code* yang tersedia secara ­*open-source*. Dungeon Crawl Stone Soup dapat diunduh melalui tautan <https://crawl.develz.org/download.htm>.

**3.1.1.9 Lisensi**

Dungeon Crawl Stone Soup dirilis di bawah lisensi GNU General Public License v2.0 atau setelahnya, yang berarti semua orang berhak mengakses, mengunduh, menyebarkan, dan mengubah *game* ini dengan bebas.

**3.1.2 *Source Code***

Subbab ini menjelaskan *source-code* dari *game* studi kasus.

**3.1.2.1 *Overview***

*Source-code* Dungeon Crawl Stone Soup di-*host* di situs repositori GitHub, di mana *code* dapat diakses dan diubah dengan bebas oleh kontributor-kontributor yang telah terdaftar.

**3.1.2.2 *Repository***

Repositori Dungeon Crawl Stone Soup dapat diakses melalui tautan <https://github.com/crawl/crawl>.

**3.1.2.3 Bahasa**

Bahasa pemrograman yang digunakan dalam pengembangan Dungeon Crawl Stone Soup adalah C/C++. Meskipun format *code* yang digunakan adalah C++, tetapi paradigma *code* yang digunakan belum menerapkan konsep *object-oriented*.

**3.1.2.4 *Packages***

Dungeon Crawl Stone Soup memanfaatkan beberapa *package* yang tersedia secara *open-source*, yaitu:

* Lua scripting language, yang digunakan untuk fungsionalitas *in-game* dan *user macros*.
* PCRE library, yang digunakan untuk *regular expressions*.
* SQLite library, yang digunakan sebagai *engine* dari *database*.
* SDL and SDL\_image libraries, yang digunakan untuk tampilan *tile*.
* libpng library, yang digunakan untuk memuat gambar *tile*.

**3.1.2.5 *Project Hierarchy***

Seluruh *source-code* utama yang membentuk *game* Dungeon Crawl Stone Soup terdapat dalam satu buah folder dengan direktori crawl/crawl-ref/source/. Setiap *file* yang mengandung *source-code* ditempatkan dalam satu *folder* tersebut, tanpa dipisahkan berdasarkan pengelompokan tertentu.

**3.1.2.6 Instalasi dan *Building***

Karena bahasa yang digunakan dalam *source code* adalah C/C++, maka untuk kemudahan eksperimen, *game* akan di-*install* pada platform Linux. Untuk platform Linux, instalasi dilakukan dengan memasukkan beberapa *command* di Terminal sebagai berikut:

# Install the source repository

echo 'deb https://crawl.develz.org/debian crawl 0.18' | sudo tee -a /etc/apt/sources.list

# Install the DCSS signing key

wget https://crawl.develz.org/debian/pubkey -O - | sudo apt-key add -

# update your package list

sudo apt-get update

# install console version

sudo apt-get install crawl

*Command* di atas digunakan untuk melakukan instalasi Dungeon Crawl Stone Soup versi *console*. Instalasi untuk versi *graphical tile* dapat menggunakan *command* berikut:

# install tiles version

sudo apt-get install crawl-tiles

Setelah instalasi selesai dilakukan, Dungeon Crawl Stone Soup dapat di-*compile* dengan *command* berikut:

# go to file directory

cd crawl/crawl-ref/source/

# compile console version

make

*Command* di atas digunakan untuk *compile* *game* versi *console*. Untuk *compile* versi *graphical tile* dapat menggunakan *command* berikut:

# compile tiles version

make TILES=y

Setelah selesai *compile*, Dungeon Crawl Stone Soup dapat dijalankan dengan *command*:

# launch the game

./crawl

**3.1.2.7 *Dependencies***

Untuk sistem Debian, *dependencies* dapat diunduh dengan *command* berikut:

apt-get install build-essential libncursesw5-dev bison flex liblua5.1-0-dev libsqlite3-dev libz-dev pkg-config libsdl2-image-dev libsdl2-mixer-dev libsdl2-dev libfreetype6-dev libpng-dev ttf-dejavu-core

**3.2 Game Domain Analysis**

Sebelum eksperimen langsung dengan *code* dimulai, dilakukan analisis terhadap *game domain* dari domain dan *game* studi kasus yang akan dikerjakan. Langkah-langkah analisis dilakukan mengikuti pendekatan *domain-specific* yang dijelaskan dalam *paper* Improving Digital Game Development with Software Product Lines.

**3.2.1 Membayangkan dan Menganalisis *Game Domain***

*Game domain* yang dibahas adalah domain dari Dungeon Crawl Stone Soup, yaitu *Role-playing game*. *Game domain* dianalisis dengan menjabarkan dan menjelaskan *game dimensions* inti yang terdapat dalam *game*. *Game dimensions* inti dari domain RPG dapat didefinisikan sebagai berikut:

1. Pemain
   1. Jumlah pemain

Sebagian besar *game* dalam domain RPG didesain untuk dimainkan oleh satu pemain, tetapi beberapa *game* memungkinkan lebih dari satu pemain bermain secara bersamaan dalam satu sesi permainan.

* 1. Mode *multiplayer*

*Game* yang menyediakan fitur *multiplayer* biasanya menyediakan mode-mode permainan tertentu untuk dimainkan oleh lebih dari satu orang.

1. Grafis
   1. Dimensi

Representasi visual dari *game* dapat berupa gambar (atau karakter ASCII untuk beberapa *game*) dua dimensi (2D) atau tiga dimensi (3D).



Gambar 3.3 Far Cry (2004) menggunakan grafis 3D

* 1. Resolusi

Untuk *game* yang menggunakan gambar dan model 3D, biasanya menggunakan gaya resolusi rendah (*pixelated*) atau resolusi tinggi (*high definition*).



Gambar 3.4 Minecraft (2010) menggunakan grafis resolusi rendah

1. Flow
   1. Level/*map*

Biasanya, permainan dalam *game* dipartisi menjadi beberapa level atau tingkatan. Sebuah level juga sering disebut sebagai *map* atau peta. Pada umumnya, setiap level atau *map* tidak saling berhubungan secara langsung dengan level dan *map* lainnya. Apa yang terjadi dalam suatu level tidak akan memengaruhi level lain. Sistem partisi dengan menggunakan level mengarahkan pemain langsung ke tujuan akhir permainan, berbeda dengan sistem *open-world* yang membiarkan pemain menjelajahi tempat manapun yang diinginkan. Hal ini membuat *game* yang menggunakan sistem level non-dinamis disebut bersifat *linear*.

* 1. World

Berbeda dengan sistem partisi *game* dengan level dan *map*, beberapa *game* justru memberikan sebuah dunia tanpa partisi yang dapat dijelajahi oleh pemain. Meskipun kadang dunia tersebut terkadang dipartisi untuk menghemat waktu dan memori saat komputer memuat aset-aset *game*, tetapi biasanya dunia-dunia yang dipartisi tersebut saling berhubungan secara dinamis.

1. Entities

*Entities* mencakup seluruh elemen benda yang terdapat dalam dunia *game* seperti senjata, *items*, musuh, serta *non-playable character*.

1. Events
   1. Player-controlled events

Terdapat *event-event* yang dipengaruhi oleh pemain. *Event-event* tersebut tidak memiliki ketentuan khusus yang ditetapkan oleh *developer*. Pemain yang menentukan terjadi atau tidaknya serta kapan *event* tersebut terjadi.

* 1. Scripted events

*Event* yang sudah dibuat oleh *developer* agar terjadi pada waktu dan batasan yang ditentukan.

1. Controller Input

*Controller input* mengatur input-input yang diterima oleh *game* melalui *keyboard*, *mouse*, ataupun *controller* lain seperti *joystick* dan *gamepad*.

1. Audio

*Audio* merupakan suara yang dihasilkan oleh *game*. Suara dapat berupa percakapan, musik, atau bunyi efek.

1. Physics

*Physics* merupakan elemen-elemen fisika yang berlaku di dalam dunia *game*, seperti gravitasi, gaya gesek, dan lain lain.

1. AI

*Artificial Intelligence* (AI) merupakan kecerdasan buatan yang menjalankan semua *entity* selain karakter pemain dalam *game*.

1. Networking
   1. Local

Beberapa *game* menyediakan fitur *multiplayer* dengan dua atau lebih *game instance* yang berjalan bersamaan dalam waktu yang sama. Biasanya, terdapat dua jenis jaringan untuk menyambungkan *game instances* tersebut. Jaringan *local* merupakan koneksi yang menghubungkan beberapa sistem dalam satu jaringan yang sama.

* 1. Online

*Online network* merupakan sistem koneksi yang menghubungkan dua atau lebih *game instance* yang tidak terhubung dalam sebuah jaringan *local*.

1. Character Progression

*Character progression* merupakan sistem yang mengatur bagaimana karakter pemain dapat berkembang. *Character progression* merupakan salah satu aspek penting yang mendefinisikan domain RPG.

1. Game Progression

*Game progression* merupakan sistem yang mengatur bagaimana *game* berjalan dan berkembang. Aspek ini berkaitan dengan aspek *flow* yang mengatur apakah sebuah *game* bersifat *linear* atau *open*. Sistem *game* *progression* secara umum menentukan bagaimana pemain dapat melaju ke tahap selanjutnya dalam permainan. Beberapa *game* seperti Assassin’s Creed memiliki *game progression* yang ditentukan oleh *character progression*, di mana cerita berjalan seiring dengan perkembangan karakter pemain.

1. World Mechanics

Aspek ini menjalankan banyak hal dalam dunia *game*, mulai dari kehidupan *non-playable character*, sistem ekonomi yang berlaku dalam dunia *game*, hingga sistem *quest* atau tugas bagi pemain.

**3.2.2 Mengidentifikasi *Emotion-based* *Requirements***

Berdasarkan *paper* Eight Ways Videogames Generate Emotion oleh Jonathan Frome, terdapat beberapa emosi yang banyak terdapat dalam sebuah *game*. Contoh emosi yang terdapat dalam *game* adalah:

1. Immersion

Daya tarik *game* terhadap pemain. Pemain merasa terbawa dengan alur cerita atau masuk dalam suasana *game*. *Immersion* yang tinggi akan membuat pemain merasa betah untuk terus bermain.

1. Surprise

Unsur *surprise* membuat pemain merasa terkejut oleh elemen-elemen dalam *game*. Kejutan-kejutan yang diberikan dalam *game* dapat membuat pemain semakin tertarik dan menghindari kebosanan.

1. Satisfaction

Kepuasan yang dirasakan oleh pemain ketika memperoleh sebuah keberhasilan dalam *game*. Unsur ini penting untuk memberi motivasi kepada pemain. Pemain akan merasa senang jika usaha yang dilakukannya dalam membuahkan sesuatu, sehingga elemen-elemen dalam *game* perlu dibuat agar memberikan suatu kepuasan kepada pemain yang telah memperoleh suatu pencapaian.

1. Intrigue

Pemain merasa tertarik atau penasaran dengan jalan cerita *game* atau *character progression*. Dengan adanya unsur intrik, maka *game* akan menarik lebih banyak pemain, serta menjaga pemain yang telah mulai bermain dari kebosanan.

1. Intensity

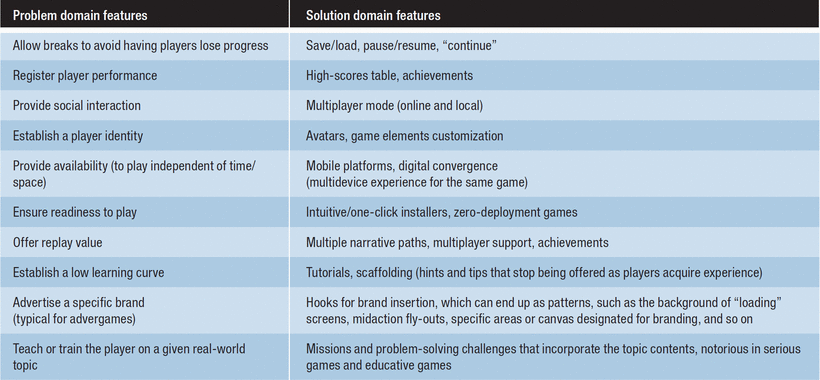
Tekanan emosi yang dirasakan pemain ketika bermain, baik emosi yang baik ataupun buruk. Intensitas tinggi akan membuat pemain merasa lebih bersemangat main karena pemain merasa lebih terlibat dalam sebuah *game*.

1. Frustration

Untuk memberikan kesan dan efek kepuasan, perlu diberikan kesan frustrasi dan kekesalan ketika pemain mengalami kegagalan. Dengan demikian, pemain akan lebih bersemangat bermain untuk menghindari kekalahan. Keberhasilan juga akan lebih memberikan kepuasan dengan adanya unsur frustrasi untuk kegagalan.

**3.2.3 *Nonemotional Requirements***

Selain *emotion-based requirements*, terdapat juga *non-emotional requirements* yang perlu dipertimbangkan dalam pengembangan *game*. *Nonemotional requirements* tidak hanya mengandung fitur-fitur dalam *game*, tetapi juga seluruh hal yang berhubungan dengan *game* tersebut sebagai sebuah aplikasi dilihat dari berbagai aspek. Dari *paper* yang diacu, terdapat beberapa contoh *nonemotional game features* yang termasuk dalam *non-emotional requirements*, seperti dalam gambar tabel berikut.

****

Gambar 3.5 *Nonemotional Requirements*

Beberapa fitur yang dapat diambil sebagai *requirement* penting, serta beberapa *requirement* lain yang perlu dipertimbangkan dalam pengembangan *game* adalah:

1. Breaks to avoid players losing progress

*Game* dalam domain RPG biasanya membutuhkan waktu yang lama untuk ditamatkan. Pemain butuh waktu untuk istirahat dan melakukan aktifitas lainnya setelah menghabiskan beberapa waktu dengan *game*. Agar *progress* pemain tidak hilang setelah pemain berhenti bermain, perlu dibuat sebuah sistem yang menyimpan *progress* pemain, atau sistem yang dapat membuat pemain meninggalkan permainan untuk sementara dengan kondisi *game* terhenti (*paused*).

1. Register player performance

Salah satu bentuk kepuasan pemain ketika bermain *game* adalah ketika berhasil mendapatkan nilai atau skor tinggi. Pemain tentu ingin agar skor yang telah diperolehnya disimpan agar dapat dilihat kembali oleh dirinya atau pemain lain.

1. Social interaction

Memasuki era media sosial *online*, banyak orang yang berkomunikasi dan berinteraksi secara *online* melalui aplikasi yang terhubung dengan Internet. Melalui media sosial, orang-orang berbagi informasi emngenai banyak hal. Salah satu fenomena sosial dalam dunia *game* adalah banyaknya *game*, terutama untuk perangkat *mobile*, yang terintegrasi dengan aplikasi media sosial. Integrasi ini memudahkan pemain untuk berbagi informasi mengenai *game* yang sedang dimainkannya atau prestasi yang diraihnya dalam *game* tersebut kepada orang lain.

1. Player identity

Salah satu tujuan utama dan motivasi orang memainkan *game* bergenre RPG adalah untuk merasakan atau mencoba menghidupi sebuah peran, baik fiktif ataupun berdasarkan kenyataan. Ketika memainkan sebuah karakter dalam *game*, sering kali orang ingin agar karakter tersebut merepresentasikan dirinya. Untuk itu, banyak *game* yang menyediakan fitur untuk membuat sebuah *avatar* pemain atau kustomisasi karakter pemain.

1. Platform availability

Pengembang *game* perlu *m*enentukan platform yang tepat untuk *game*-nya agar pemain dapat bermain semudah dan senyaman mungkin. Contohnya untuk *game* ringan dengan *paid features* yang hanya dimainkan sesaat setiap beberapa waktu akan lebih cocok untuk platform *mobile*, sedangkan *game* kompetitif dengan grafis realistis dan *controls* yang rumit lebih cocok untuk platform PC atau *console*.

1. Replay value

Agar pemain dapat terus memainkan *game* tanpa merasa bosan, perlu dibuat agar pemain mendapatkan pengalaman yang berbeda setiap kali bermain, juga agar *game* masih memiliki daya tarik setelah cerita utama ditamatkan. Contoh fitur yang menaikkan *replay value* sebuah *game* adalah fitur *multiplayer*, dunia yang bersifat *open* dengan banyak tempat untuk dijelajahi, serta level yang berbeda untuk setiap sesi permainan.

1. Low learning curve

Beberapa *game* memiliki tingkat kesulitan dan kerumitan yang tinggi, namun akan lebih baik bagi pemain jika *game progression* dibuat lambat untuk membiarkan pemain belajar dan membiasakan diri dengan mekanisme dan dunia *game*. Jika pemain dihadapkan dengan rintangan yang sulit sejak awal permainan sehingga menghambat *progress* pemain, pemain mungkin merasa tidak tertarik lagi dengan *game* tersebut. Meski demikian, hal ini tidak berlaku bagi *game*-*game* yang memang dipromosikan sebagai *game* yang sulit, atau tantangan dan kesulitan merupakan daya tarik utama *game* tersebut. Contoh *game* yang daya tarik utamanya merupakan tantangan dan kesulitan adalah Dark Souls dan FTL.

1. Accessibility

Kemudahan akses dan instalasi juga perlu dipertimbangkan oleh pengembang *game*. Seiring berkembangnya industri *game*, pengembang *game* semakin memperluas lingkup target pemain. *Game* yang banyak berkembang sekarang secara umum ditargetkan pada pemain yang sekedar *familiar* dengan teknologi. Hal ini berbeda dengan era 1990-an ketika pemain *game* merupakan orang-orang yang mahir mengoperasikan perangkat teknologi. Di masa sekarang, *game* yang sulit untuk diakses atau diinstal cenderung akan dihindari oleh pemain pada umumnya.

1. Security

Seperti halnya aplikasi dan program lain pada umumnya, pengembang *game* juga perlu memperhatikan aspek keamanan dari *game* yang dikembangkan. Banyaknya *game* di masa sekarang yang melibatkan *microtransaction* atau transaksi dalam *game* yang melibatkan uang dunia nyata, serta terhubungnya *game* dengan Internet membuat *game* rentan terhadap serangan peretas atau pemain yang mencoba mengeksploitasi *bug* untuk keuntungannya sendiri. Pengembang *game* perlu memastikan bahwa *game* yang dikembangkan aman dari segala bentuk serangan dan eksploitasi agar perusahaan pengembang dan pemain *game* terhindar dari kerugian.

1. Legalities

*Game* merupakan media *digital* yang terlibat dalam hukum, setidaknya untuk beberapa negara, terutama negara maju. Pengembang perlu memastikan bahwa *game* yang dibuatnya tidak menyalahi peraturan dan hukum di tempat *game* tersebut dibuat dan dipasarkan, baik untuk hukum yang mengatur konten dari *game* tersebut, atau hukum yang mengatur hak cipta serta lisensi dari aset-aset yang digunakan dalam *game*.

**3.2.4 Mengambil Contoh dari Domain**

Contoh *game* RPG yang memiliki *sequel* atau berseri adalah Assassin’s Creed. Seri Assassin’s Creed memiliki lebih dari 10 *game* yang memiliki banyak kesamaan *gameplay*, tetapi juga memiliki beberapa fitur yang dikembangkan atau diubah. Berikut adalah aspek-aspek yang memiliki kesamaan di setiap *game* dalam seri:

1. Story theme

Dalam seri ini, tema cerita tidak berubah. Dalam setiap *game*, pemain memerankan seorang *assassin*, atau pembunuh yang memerangi sebuah organisasi bernama *templars*.

1. Core mechanics

Cara bermain, kondisi kemenangan, serta batasan-batasan permainan secara umum tetap sama.

1. Character progression

Sistem *character progression* secara umum tetap sama. Terdapat beberapa perbedaan seperti cara mendapatkan poin untuk memperkuat karakter, tetapi struktur perkembangan karakternya tetap sama.

1. Level/map design

Meskipun detail model dan gambar dari level dan dunia masing-masing *game* berbeda, tetapi desain umumnya tetap sama. Terdapat beberapa perubahan atau penambahan elemen dunia seperti penambahan laut untuk dijelajahi mulai dari Assassin’s Creed III, tetapi desain dunia *game* secara keseluruhan tetap sama.

Sedangkan aspek-aspek yang berbeda adalah:

1. Story settings

Setiap *game* dalam seri ini memiliki latar cerita yang berbeda dengan tempat, waktu, dan karakter yang berbeda-beda.

1. Artwork

Hampir semua elemen dalam masing-masing *game* menggunakan aset model dan gambar yang berbeda dengan *game* lainnya.

1. Controls

Konfigurasi pengendalian karakter mengalami perubahan di beberapa *game*, dengan tujuan untuk mempermudah dan menyederhanakan *controls*.

1. Platform availability

Beberapa *game* di-*port*, atau diadaptasi dari sebuah platform ke beberapa platform yang berbeda.

**3.2.5 Menentukan dan Mengkaji Ulang Fitur *Game Domain***

*Game* bergenre RPG memiliki banyak fitur dan aspek. Dalam subbab ini, akan dibahas fitur-fitur apa saja dari domain RPG yang biasanya menjadi target penerapan SPL, baik fitur-fitur yang biasanya menjadi *commonality* atau memiliki *variability*. Berikut adalah fitur-fitur yang biasanya menjadi *commonality*:

1. Gameplay

*Game-game* RPG memiliki banyak perbedaan. *Game* yang berbeda dalam satu subdomainpun dapat memiliki perbedaan yang signifikan. Akan tetapi, karena secara umum subdomain-subdomain dari RPG dikelompokkan berdasarkan perbedaan *gameplay*, maka *gameplay* dari *game-game* yang terdapat dalam suatu subdomain memiliki banyak kesamaan, sehingga aspek *gameplay* cukup jarang menjadi perbedaan dalam pembuatan variasi. Sangat jarang terdapat sebuah *game* RPG yang memiliki beberapa tipe *gameplay* dari subdomain yang berbeda. Beberapa elemen dalam *gameplay* mungkin berbeda, namun inti *gameplay* seperti cara bermain, tujuan bermain, dan kondisi kemenangan biasanya tetap sama.

1. Story Theme/Emotion

Suatu *game* RPG biasanya memiliki tema atau latar cerita utama yang menjadi inti dari permainan. Meskipun dunia dan cerita dalam *game* RPG cenderung luas dan panjang, tetapi biasanya tetap dalam tema cerita dan emosi utama yang sama.

1. Character Progression

Sama seperti *gameplay*, sistem *character progression* merupakan sesuatu yang jarang diubah atau dijadikan perbedaan/variasi dalam sebuah *game*. *Character progression* sering menjadi identitas utama dari sebuah *game*, sehingga sistem tersebut dibiarkan sebagai sebuah *commonality* dalam *game*.

Sedangkan fitur-fitur yang biasanya memiliki *variablity* adalah:

1. Story Settings

Meskipun emosi dan tema utama cerita biasanya tidak menjadi perbedaan, latar cerita merupakan suatu aspek yang umum untuk dijadikan perbedaan. Perbedaan latar dapat berupa latar tempat atau waktu, tetapi tema cerita biasanya tetap sama.

1. Level/Map Design

Untuk menghindari kebosanan pemain, desain level dan dunia *game* biasanya dibuat berbeda-beda untuk setiap tahapan *progress* pemain. Level dan *map* yang berbeda-beda tersebut biasanya tetap memiliki *gameplay* dan cara bermain yang sama.

1. Character identitiy

*Game* RPG pada umumnya memiliki banyak karakter yang berbeda untuk peran yang berbeda, baik karakter yang dapat dimainkan oleh pemain, maupun karakter khusus AI yang tidak dapat dimainkan pemain atau *non-playable character* (NPC). Karena biasanya karakter-karakter tersebut memiliki tujuan dan sifat yang berbeda, maka identitas masing-masing karakterpun dibedakan. Perbedaan biasanya dilakukan berdasarkan sifat, seperti karakter baik atau karakter jahat. Beberapa karakter khusus yang memiliki tingkat kepentingan yang tinggi untuk alur cerita *game* biasanya juga dibedakan menjadi karakter-karakter yang unik.

1. Additional Features

Banyak *game* yang memiliki fitur-fitur tambahan seperti yang akan dibahas pada subbab berikutnya. Fitur-fitur tersebut biasanya merupakan *spin-off* dari fitur atau *gameplay* utama.

**3.2.6 Membahas Fitur *Game* yang Terkunci**

Definisi fitur terkunci dalam konteks ini adalah fitur yang baru dapat diakses oleh pemain ketika pemain tersebut telah memperoleh atau mencapai suatu batasan *progress* yang ditentukan. Beberapa fitur yang biasanya dibuat sebagai fitur terkunci dalam sebuah *game* adalah:

1. Cheat Codes

*Cheat codes* merupakan kode rahasia untuk membuka fitur-fitur atau benda *in-game* tertentu, atau mempercepat *character progression*. Fitur *cheat* dapat ditawarkan kepada pemain secara eksplisit, contohnya seperti beberapa *game* yang menampilkan menu untuk memasukkan kode *cheat*. Selain eksplisit, *cheat* dapat juga tersedia secara implisit, di mana pemain tidak diberitahu atau tidak ada tanda-tanda mengenai ketersediaan fitur *cheat* dalam *game* tersebut. *Cheat* yang tersembunyi biasanya berupa kombinasi tombol tertentu yang di-input oleh pemain.

1. Enhanced New Game

Dalam beberapa *game*, fitur ini disebut “New Game +”. Mode ini biasanya terbuka setelah pemain berhasil menamatkan permainan. Dalam mode ini pemain dapat bermain dari awal permainan, tetapi dengan *character progression* dan *unlockables/collectibles* yang telah didapat dari sesi *game* yang telah tamat.

1. Bonus Levels/Stages

Pemain dapat membuka level-level bonus yang tidak terdapat dalam cerita utama *game*. Level bonus dapat berupa rahasia atau eksplisit. Biasanya level-level bonus merupakan *spin-off* dari level atau cerita utama *game*.

1. Other Special Modes

Selain mode utama, terdapat mode-mode lain berupa tantangan atau mode santai, contohnya untuk *game* dengan genre *shooter*, terdapat mode tanpa senjata api, atau untuk game petualangan, baik yang bersifat *open* atau linear, terdapat mode arena/*colosseum* untuk mengadu pemain dengan karakter-karakter AI dalam game, atau melawan pemain lain.

1. Paid Features

Beberapa game memiliki fitur-fitur tertentu yang hanya dapat diakses jika pemain membayar atau membeli fitur tambahan tersebut kepada pembuat *game*.

**3.2.7 Mengidentifikasi Beberapa Subdomain**

Domain atau genre RPG memiliki banyak jenis dan variasi. Jenis-jenis dan variasi tersebut dapat didefinisikan sebagai subgenre atau subdomain dari RPG. Beberapa subdomain dari RPG dapat didefinisikan sebagi berikut:

1. Narrative

Subgenre *narrative* merupakan jenis RPG yang titik berat permainan dan *character progression*-nya terletak pada alur cerita. Contoh *game* yang termasuk dalam subgenre *narrative* adalah Deus Ex dan Mass Effect.

1. Sandbox

Subgenre ini menyediakan sebuah dunia yang bebas untuk dijelajahi. Pemain diberi sedikit atau tidak ada sama sekali batasan dalam level atau *character progression*. Ciri khas subgenre ini adalah bersifat *open world*, di mana dunia dalam *game* bebas untuk dijelajahi, minim pembatasan, dan biasanya bersifat dinamis. Beberapa *game sandbox* memiliki alur cerita, namun sebagian beberapa tidak. Contoh *game* yang termasuk dalam subgenre *sandbox* adalah Minecraft dan Terraria.

1. Dungeon Crawler

*Dungeon crawler* merupakan subgenre yang berfokus pada *character progression*. Biasanya *game* dengan subgenre ini dipartisi menjadi beberapa level, terkadang jumlah levelnya tidak terbatas. Salah satu ciri utama dari *dungeon crawler* adalah terdapat banyak *loot* atau benda-benda yang dapat diambil oleh pemain. Contoh *game* yang termasuk dalam subgenre *dungeon crawler* adalah Diablo dan Dungeon Crawl Stone Soup.

1. Mixed

Subgenre *mixed* merupakan pengelompokan untuk *game-game* RPG yang memiliki unsur-unsur dari dua atau lebih subgenre lain. Seiring bertambahnya jumlah *game* dan berkembangnya inovasi serta kreativitas pengembang *game*, subgenre ini menjadi subgenre yang paling umum dalam domain RPG. Kombinasi subgenre yang paling umum adalah *narrative* dan *sandbox*. Contoh *game* yang termasuk dalam subgenre *mixed* adalah The Elder Scrolls dan Fallout.

1. Strategy

Sering disebut sebagai *tactical* RPG. Subgenre *strategy* dalam konteks ini berbeda dengan genre *strategy*. Jenis *strategy* yang dibahas dalam subgenre ini adalah *game-game* RPG yang mencampurkan unsur strategi dalam permainannya. Dalam genre *strategy* sendiri terdapat dua jenis pengelompokan utama, yaitu *real-time strategy* dan *turn-based* *strategy*. Jenis yang paling sering terdapat dalam *strategy* RPG adalah *turn-based*, di mana pemain bermain bergantian dengan AI atau pemain lain. Meskipun *game-game* tersebut memiliki unsur strategi, tetapi karakteristik yang paling kuat dalam *game-game* tersebut tetap unsur *role-playing*. Tipe ini banyak terdapat dalam RPG yang dikembangkan oleh *developer* Jepang, atau yang dikenal sebagai JRPG. Contoh *game* yang termasuk dalam subgenre *strategy* adalah Pokemon dan Final Fantasy. *Game* seperti Heroes of Might and Magic yang pada dasarnya merupakan *game* bertema strategi namun memasukkan unsur-unsur RPG tidak termasuk dalam genre dan subgenre *strategy* RPG.

1. Action

Subgenre ini berfokus pada aspek aksi dari *game*. *Game-game* dengan subgenre ini biasanya memiliki banyak QTE (*quick time events*), di mana pemain diminta untuk melakukan suatu aksi seperti menekan tombol tertentu pada *controller* untuk konteks tertentu seperti untuk melawan musuh. Subgenre lain sepeti *dungeon crawler* dan *roguelike* banyak termasuk dalam subgenre ini. Contoh *game* yang termasuk dalam subgenre *action* adalah Dark Souls dan Dynasty Warriors.

1. Roguelike

*Roguelike* memiliki definisi yang terkadang *interchangeable* dengan *dungeon crawler*. Perbedaan antara *roguelike* dengan *dungeon crawler* adalah biasanya *roguelike* lebih menitikberatkan pada *replay value*, dengan ciri khas lain yang dijelaskan sebelumnya di subbab 3.1.1.2, sedangkan *dungeon crawler* merupakan *game* RPG dengan karakteristik *game* apapun yang mengandung skenario *dungeon crawl*. Biasanya setiap sesi permainan akan berbeda dengan sesi sebelumnya. Contoh *game* yang termasuk dalam subgenre *roguelike* adalah FTL dan The Binding of Isaac. Dungeon Crawl Stone Soup juga termasuk dalam *roguelike*.

1. MMORPG

MMORPG (*Massively Multiplayer Online Role-playing Game*) merupakan subgenre RPG untuk *game-game* didesain untuk dimainkan secara *online* dan melibatkan banyak pemain. Karakteristik dari *game-game* MMORPG biasanya merupakan gabungan dari *sandbox*, *dungeon crawler*, dan *action*. Contoh *game* yang termasuk dalam subgenre MMORPG adalah World of Warcraft dan Ragnarok Online.

1. JRPG

JRPG (*Japanese Role-playing Game*) merupakan subgenre RPG untuk *game-game* RPG yang dibuat oleh *developer* Jepang. JRPG memiliki ciri khusus seperti grafis yang *pixelated* atau *artwork* yang menyerupai *style* gambar *anime* atau *manga*. Karakteristik permainan JRPG biasanya merupakan gabungan atau variasi dari *dungeon crawler* dan *strategy*. Contoh *game* yang termasuk dalam subgenre JRPG adalah Shin Megami Tensei dan Breath of Fire.

**3.2.8 Mengantisipasi Fitur yang Akan Datang**

Fitur baru yang diantisipasi adalah fitur untuk *game* Dungeon Crawl Stone Soup secara spesifik. Antisipasi fitur baru dapat dilakukan dengan beberapa teknik seperti *retrospection*, *trend analysis*, dan *the morphologic box*. Berikut adalah fitur-fitur yang sering dikembangkan pada tahap lebih lanjut dalam pengembangan *game*:

1. Multiplayer

Untuk saat ini, Dungeon Crawl Stone Soup belum menyediakan fitur *multiplayer* yang memungkinkan lebih dari satu pemain bermain dalam satu sesi yang sama. Melihat populernya fitur ini pada *game-game* sejenis, penambahan fitur *multiplayer* patut diantisipasi.

1. Controller Support

Banyak pemain *game* berbasis PC yang menggunakan *controller* eksternal selain *mouse* dan *keyboard* untuk bermain. Salah satu bentuk *controller* yang cukup banyak digunakan adalah *joystick*. Genre *game* yang cukup sering memanfaatkan *controller* sepert *joystick* adalah *sports*, *racing*, dan RPG.

1. VR Support

Perkembangan teknologi hiburan *digital* yang pesat telah melahirkan sebuah teknologi baru, yaitu *virtual reality*. Teknologi ini dapat memungkinkan penggunanya memainkan sebuah *game* atau program simulasi dengan lebih realistis, menggunakan sebuah layar yang dijadikan semacam kacamata, serta *controller* yang mengikuti gerakan pemain di dunia nyata. Banyaknya *game* yang dibuat khusus untuk teknologi ini, dan banyaknya *game* lain yang turut mendukung teknologi ini memungkinkan penggunaan perangkat *virtual reality* dalam *game* sebagai suatu hal yang sangat umum di masa yang akan datang.

1. HD Remaster

Dungeon Crawl Stone Soup pertama kali dibuat pada tahun 1997. Pada masa itu, grafis dalam *game* masih sangat *pixelated*, ditambah dengan populernya grafis *pixelated* atau bahkan grafis ASCII dalam subgenre *roguelike*. Dalam beberapa tahun terakhir, banyak bermunculan *game* dengan tema *dungeon crawler* dan subgenre *roguelike* yang menawarkan grafis yang lebih modern, bahkan sebanding dengan *game-game* lain di domain yang berbeda. Grafis dan *interface* yang lebih baik akan menarik lebih banyak pemain dan menambah popularitas, sehingga bukan tidak mungkin Dungeon Crawl Stone Soup akan di-*remaster* menjadi sebuah *game* yang mengusung grafis *high definition* di masa depan.

1. Development for other platforms

Pengembangan untuk platform lain selain dari platform utama *game* yang dibuat merupakan suatu hal yang umum dilakukan oleh pengembang-pengembang *game*. Dungeon Crawl Stone Soup sendiri telah tersedia untuk berbagai platform seperti Windows, Linux, Mac OS X, dan Android, akan tetapi tidak menutup kemungkinan akan dibuat versi untuk platform lain seperti iOS atau perangkat *console* seperti PlayStation dan Xbox.

1. Cross-platform multiplayer support

Ketika fitur *multiplayer* telah diimplementasikan, ada kemungkinan bahwa akan dibuat sistem yang memungkinkan permainan *multiplayer* antar-platform. Fitur ini memungkinkan pemain berbasis Windows bermain dengan pemain lain yang berbasis Linux atau Mac OS X, atau bahkan Android, begitu pula sebaliknya.

**BAB 4**

**EKSPERIMEN DAN ANALISIS**

Bab ini menjelaskan rincian eksperimen yang dilakukan, beserta penjelasan dasar-dasar dari langkah yang dilakukan.

Eksperimen yang akan dilakukan dalam penelitian ini adalah melakukan perubahan pada *source code* dari *game* studi kasus. Perubahan yang akan dilakukan adalah membuat dua buah versi dari *game*, di mana masing-masing versi memiliki beberapa aspek atau fitur yang sama, tetapi untuk beberapa aspek atau fitur lain memiliki perbedaan konfigurasi.

**4.1 Strategi untuk *refactoring***

*SPL* mengedepankan kemudahan navigasi *source code* sebagai salah satu usaha untuk memudahkan *programmer* bekerja. Jika dilihat, *source code* studi kasus walaupun menggunakan bahasa C++ masih tidak memanfaatkan paradigma *object-oriented* dengan baik. Untuk memudahkan navigasi, dilakukan *refactoring* sehingga *source code* lebih memanfaatkan paradigma *object-oriented* dan menerapkan *SPL* dengan baik. Dari *paper* yang dibahas di bab tinjauan literatur, dapat dilakukan beberapa langkah untuk membantu *refactoring*:

1. Menghilangkan *constant interface*

*Interface* yang digunakan untuk menyimpan konstanta dinilai sebagai sebuah *anti-pattern* yang tidak cocok dengan paradigma *object-oriented*. Karena penggunaannya memberikan beberapa dampak negatif, maka penggunaan *interface* untuk mendefinisikan konstanta dapat dihapus. Konstanta dapat didefiniskan secara langsung di dalam *class*.

1. Mengubah atribut *class* menjadi variabel lokal

Atribut-atribut yang didefinisikan di dalam *class* namun hanya digunakan oleh sebuah *method* dapat dipindahkan dan didefinisikan di dalam *method* tersebut.

1. Menghilangkan *inheritance relationship* yang redundan

Beberapa *class* yang terhubung dengan *inheritance* namun memiliki sangat sedikit kesamaan, dapat dihapus hubungan *inheritance*-nya.

1. Mengubah iterasi *loop* menjadi perbandingan dengan nol

Dalam sebuah iterasi, membandingkan suatu bilangan dengan nol lebih cepat dibandingkan dengan bilangan lain, sehingga *loop* yang membandingkan dengan sebuah bilangan bukan nol dapat diganti menjadi membandingkan dengan nol. Contoh:

for(int i=0; i<6; i++)

*Loop* 1. Membandingkan dengan bilangan bukan nol

for(int i=5; i>=0; i--)

*Loop* 2. Membandingkan dengan bilangan nol

Bentuk *loop* 1 dapat diganti menjadi *loop* 2 untuk mempercepat perbandingan.

Strategi ini tidak terlalu relevan dengan konsep optimisasi SPL, tetapi merupakan salah satu cara yang dapat diterapkan untuk meningkatkan performa *code*.

1. Menghilangkan *class* yang tidak perlu

Beberapa *class* yang memiliki sangat sedikit *method* dapat dihapus dengan cara memindahkan *functionality* yang terdapat dalam *class* tersebut ke *class* lain.

1. Menghilangkan *class* dan *method* yang sudah *obsolete*

*Method*-*method* yang tidak pernah dipakai, seperti *method* dari implementasi lama yang sudah *obsolete* dan tidak digunakan dalam implementasi baru dapat dihapus.

*Class* dan *method* yang sudah tidak terpakai namun tidak dihapus berpotensi membingungkan *programmer* yang bekerja dengan *code* tersebut.

1. Menghindari perbandingan *string* yang lambat

Perbandingan *string* dapat memakan waktu, jadi perbandingan *string* yang tidak perlu atau panjang sebaiknya dihindari.

1. Mendeklarasi *method* dan variabel sebagai *final* untuk mempercepat akses

Pendeklarasian *final* dapat membantu *compiler* bekerja lebih cepat. Selain itu, penggunaan *final* juga dapat mencegah pengubahan referensi *method* atau variabel yang tidak diinginkan. Penggunaan *final* dalam *code* yang dikerjakan oleh banyak kontributor juga membantu kontributor lain untuk mengetahui bahwa *method* atau variabel tersebut tidak untuk diubah-ubah.

1. Mengganti *resizeable vector* dengan *array*

*Vector* yang dialokasikan ke *heap* akan memakan waktu yang lebih lama dalam mengerjakan suatu proses dibandingkan dengan *array* yang dialokasikan ke *stack*. Untuk itu, *vector* dapat diganti dengan *array*.

1. Tidak *throw* *exception* melainkan me-*return* sebuah *null object*

*Exception* sebaiknya hanya digunakan untuk menangani atau mendeteksi *error*. Ketika menangani *object* yang diperkirakan akan menghasilkan *null* atau tidak menghasilkan apapun, sebaiknya me-*return* sebuah *null object*.

**4.2 Commonalities dan Variabilities**

Dua buah versi yang akan dibuat pada *game* didasarkan pada tingkat kesulitan atau *difficulty* dari permainan. Versi *casual* merupakan versi yang menawarkan tingkat kesulitan rendah, sedangkan versi *hardcore* merupakan versi yang tingkat kesulitannya lebih sulit. Variasi ini diangkat sebagai topik eksperimen karena beberapa hal, yaitu:

1. Pembuatan variasi tingkat kesulitan merupakan hal yang sangat umum dalam pengembangan *game*, melihat banyaknya *game* dari berbagai domain termasuk domain RPG yang memiliki variasi beberapa tingkat kesulitan.
2. Pengubahan tingkat kesulitan tidak mengubah inti permainan, dengan demikian tidak mengubah banyak struktur *source code*
3. Meskipun pembuatan variasi tingkat kesulitan tidak menyentuh banyak *code*, tetapi lingkup dari eksperimen meliputi cukup banyak aspek untuk menonjolkan dampak dari pemanfaatan SPL.

Dengan pertimbangan tersebut, ditentukan aspek-aspek *game* yang akan dibiarkan menjadi *commonality* dan aspek-aspek yang akan diubah menjadi *variant* yang memiliki *variability*.

**4.2.1 Commonalities**

1. Core gameplay

Inti dari permainan akan tetap sama. Segala hal yang menyangkut cara bermain, tujuan permainan, kondisi kemenangan, dan sebagainya akan tetap sama di kedua versi.

1. Controls

Konfigurasi *controls* akan tetap sama. Tidak ada perubahan dalam bagaimana cara pemain mengendalikan karakter.

1. Enemy types

Tipe-tipe musuh yang ada di dalam *dungeon* akan tetap sama. Meskipun konfigurasi kekuatan dan sifat musuh mungkin berbeda, tetapi bentuk, proporsi kekuatan, dan sifat secara umumnya akan tetap sama.

1. Weapon/skill types

Senjata dan *skill* yang tersedia untuk pemain akan tetap sama.

1. Dungeon design

Meskipun secara umum bentuk dan konfigurasi *dungeon* dibuat secara *random*, tetapi batasan-batasan dan kriteria pembuatan *dungeon* akan tetap sama.

**4.2.2 Variabilities**

1. Player freedom

Kebebasan pemain untuk versi *hardcore* akan dibatasi. Pemain tidak dapat melakukan beberapa aktifitas sebebas versi *casual*.

1. Enemy behavior

Untuk versi *hardcore*, musuh akan bergerak lebih cepat dan menyerang lebih sering.

1. Player capability

Kemampuan karakter pemain akan dikurangi untuk versi *hardcore*

1. Enemy capability

Kemampuan musuh akan ditambah untuk versi *hardcore*

1. Scoring system

Sistem penambahan skor akan berbeda untuk tiap versi. Versi *hardcore* akan memiliki variabel penambah skor yang lebih tinggi dari versi *casual*.

**4.3 Perbedaan dalam *code***

Perubahan dilakukan dengan mengganti konstanta dan variabel yang memengaruhi aspek-aspek yang dijelaskan dalan subbab *variabilities*.

**4.3.1 *Refactoring* Umum**

*Refactoring* umum dilakukan berdasarkan strategi yang telah dijelaskan dalam subbab sebelumnya. Dari strategi-strategi yang yang diacu dari *paper* acuan tersebut, terdapat signifikansi dan relevansi yang berbeda untuk tiap strategi. Signifikansi dan relevansi strategi-strategi tersebut akan dibahas bersama dengan pertimbangan terhadap kondisi *code* studi kasus.

1. Menghilangkan *constant interface*

Strategi ini tidak terpakai dalam eksperimen, karena *code* studi kasus memang tidak mengandung *constant interface*.

1. Mengubah atribut *class* menjadi variabel lokal

Pemindahan sebuah atribut yang hanya dipakai oleh sebuah *method* dari dalam *class* ke dalam *method* memiliki signifikansi dalam memudahkan navigasi. Selain itu, menempatkan variabel spesifik dalam *method-method* tertentu yang menggunakannya membantu *programmer* melakukan kesalahan ketika membuat variasi. Penerapan SPL dalam membuat variasi menekankan bahwa masing-masing variasi memiliki sifat *fine-grained*, atau memiliki detail yang berbeda dengan variasi lainnya.

1. Menghilangkan *inheritance relationship* yang redundan

Strategi ini tidak terpakai dalam eksperimen, karena *code* studi kasus memang tidak mengandung *constant interface*.

1. Mengubah iterasi *loop* menjadi perbandingan dengan nol

Strategi ini memiliki signifikansi yang rendah terhadap performa dan efisiensi *code*, serta kurang relevan dengan konsep yang ditekankan SPL.

1. Menghilangkan *class* yang tidak perlu

Memindahkan *method-method* dari *class* yang hanya memilikis edikit method agar kelas tersebut dapat dihapus merupakan strategi yang relevan dengan penerapan SPL. Strategi ini membantu agar pengerjaan variasi tidak memiliki efek samping yang tidak diinginkan pada *class-class* di luar *class* yang sedang diubah.

1. Menghilangkan *class* dan *method* yang sudah *obsolete*

Menghilangkan *class* tertentu yang sudah tidak digunakan merupakan sebuah *good programming practice* yang tidak terlalu relevan dengan penerapan SPL, namun tetap memiliki dampak yang baik seperti untuk kemudahan navigasi.

1. Menghindari perbandingan *string* yang lambat

Strategi ini memiliki signifikansi yang rendah terhadap performa dan efisiensi *code*, serta kurang relevan dengan konsep yang ditekankan SPL.

1. Mendeklarasi *method* dan variabel sebagai *final* untuk mempercepat akses

Strategi ini memiliki signifikansi yang rendah terhadap performa dan efisiensi *code*, serta kurang relevan dengan konsep yang ditekankan SPL.

1. Mengganti *resizeable vector* dengan *array*

Strategi ini memiliki signifikansi yang rendah terhadap performa dan efisiensi *code*, serta kurang relevan dengan konsep yang ditekankan SPL.

1. Tidak *throw* *exception* melainkan me-*return* sebuah *null object*

Strategi ini memiliki signifikansi yang rendah terhadap performa dan efisiensi *code*, serta kurang relevan dengan konsep yang ditekankan SPL.

**4.3.2 *Refactoring* Berdasarkan *Commonality* dan *Variability***

Untuk membuat variasi sistem skor dengan menerapkan konsep SPL, masing-masing variasi dari skor harus *fine grained*. Hal tersebut dicapai dengan cara memisahkan method untuk menentukan skor akhir untuk masing-masing versi.

Idealnya, untuk menerapkan SPL pada pembuatan variasi, masing-masing versi dibuat sebuah *class* yang mencakup masing-masing aspek yang berbeda antar versi.

Bentuk struktur *code* awal tidak menerapkan pendekatan SPL:

*Class*

*- Method A*

*- Function*

*- Method B*

*- Function*

*- Method C*

*- Function*

Bentuk ideal pendekatan SPL-nya adalah:

*Class 1 Class 2*

*- Method A1 - Method A2*

*- Function 1 - Function 2*

*- Method B1 - Method B2*

*- Function 1 - Function 2*

*- Method C1 - Method C2*

*- Function 1 - Function 2*

Struktur 1

Karena banyaknya *class* dalam *source code* yang berkaitan satu sama lain, pemisahan method menjadi beberapa *class* berdasarkan variasi menjadi suatu hal yang sangat sulit. Untuk itu, dalam studi kasus ini, penerapan SPL dilakukan dengan mengubah struktur menjadi:

*Class*

*- Method A1*

*- Function 1*

*- Method A2*

*- Function 2*

*- Method B1*

*- Function 1*

*- Method B2*

*- Function 2*

*- Method C1*

*- Function 1*

*- Method C2*

*- Function 1*

Struktur 2

Beberapa fungsionalitas juga memiliki keterkaitan dengan fungsi dan *method* lain, sehingga penerapan SPL dapat dilakukan dengan mengubah struktur menjadi:

*Class*

*- Method A*

*- Function 1*

*- Function 2*

*- Method B*

*- Function 1*

*- Function 2*

*- Method C*

*- Function 1*

*- Function 2*

Struktur 3

Karena banyaknya fungsionalitas dalam *game*, maka eksperimen hanya dilakukan pada bagian-bagian *code* yang paling representatif dengan aspek yang akan divariasikan dalam eksperimen ini.

1. Player freedom

Kebebasan pemain diubah dengan membatasi aksi-aksi yang dapat dilakukan oleh pemain. Karena saling terkaitnya satu fungsi dengan fungsi lain dalam *class* yang diubah, maka pengubahan dilakukan dengan menerapkan struktur 3. *Code* yang diubah adalah player.cc di mana bentuk *code* sebelum pengubahan adalah sebagai berikut:

*method*

*function*

*if*

*function*

*function*

Perubahan dilakukan dengan cara membuat sebuah *method* baru dengan bentuk yang sama, tetapi dengan konfigurasi *function* yang berbeda, sehingga masing-masing variasi akan memiliki *method* sendiri dengan konfigurasi yang berbeda. Bentuk *code* setelah dilakukan perubahan akan menjadi:

*method casual*

*function casual*

*if*

*function casual*

*function casual*

*method hardcore*

*function hardcore*

*if*

*function*  *hardcore*

*function hardcore*

1. Enemy behavior

Kekuatan pemain diubah dengan mengubah konfigurasi tingkah laku musuh. Karena saling terkaitnya satu fungsi dengan fungsi lain dalam *class* yang diubah, maka pengubahan dilakukan dengan menerapkan struktur 3. *Code* yang diubah adalah mon-behv.cc, di mana struktur *code* secara umum berbentuk seperti berikut:

*method*

*function*

*if*

*function*

*function*

Perubahan dilakukan dengan cara membuat sebuah *method* baru dengan bentuk yang sama, tetapi dengan konfigurasi *function* yang berbeda, sehingga masing-masing variasi akan memiliki *method* sendiri dengan konfigurasi yang berbeda. Bentuk *code* setelah dilakukan perubahan akan menjadi:

*method casual*

*function casual*

*if*

*function casual*

*function casual*

*method hardcore*

*function hardcore*

*if*

*function*  *hardcore*

*function hardcore*

1. Player capability

Kekuatan pemain diubah dengan mengubah kemampuan pemain untuk menyerang musuh. Karena saling terkaitnya satu fungsi dengan fungsi lain dalam *class* yang diubah, maka pengubahan dilakukan dengan menerapkan struktur 3.

Berkas yang diubah adalah attack.cc, di mana *code* awal memiliki struktur sebagai berikut:

*method*

*if* (pemain)

*function* pemain

*if* (musuh)

*function* musuh

Perubahan dilakukan dengan cara membuat sebuah variabel *flag* yang menandakan versi, kemudian membuat variasi pada masing-masing *function* yang diakses ketika variabel *flag* sesuai. Penerapan ini juga berlaku untuk aspek *enemy strength*.

Bentuk *code* setelah dilakukan perubahan akan menjadi:

*method*

*if* (pemain)

*function* pemain *casual*

*function* pemain *hardcore*

*if* (musuh)

*function* musuh *casual*

*function* musuh *hardcore*

Pembuatan variasi hingga *fine-grained* cukup sulit dilakukan karena banyaknya elemen-elemen dalam *method* dan *function* yang terhubung dengan *method* atau bahkan *class* lain, sehingga membutuhkan waktu yang sangat lama untuk *refactoring*.

1. Enemy capability

Kekuatan musuh diubah dengan mengubah kemampuan musuh untuk menyerang pemain. Penjelasan eksperimen mengacu pada penjelasan aspek sebelumnya.

1. Scoring system

Dalam aspek ini, unsur-unsur yang divariasikan adalah banyaknya skor akhir yang didapat oleh pemain untuk *experience* yang didapatkan. Untuk menghindari ketidakcocokan dengan *class* lain, maka pengubahan dilakukan dengan menerapkan struktur 2. *Code* yang diubah adalah hiscores.cc.

Bentuk *code* secara umum sebelum pengubahan adalah:

*method*

*function*

*function*

Perubahan dilakukan dengan cara membuat sebuah *method* baru dengan bentuk yang sama, tetapi dengan konfigurasi *function* yang berbeda, sehingga masing-masing variasi akan memiliki *method* sendiri dengan konfigurasi yang berbeda. Bentuk *code* setelah dilakukan perubahan akan menjadi:

*Method casual*

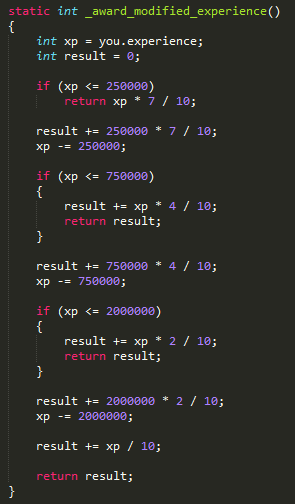
*Function casual*

*Function casual*

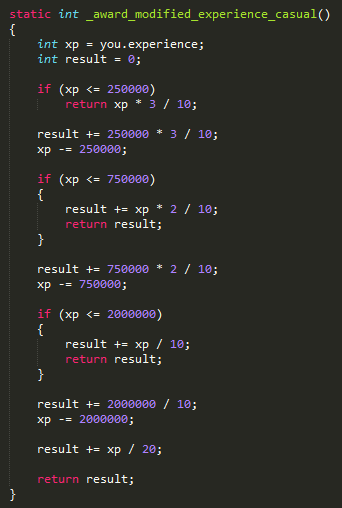
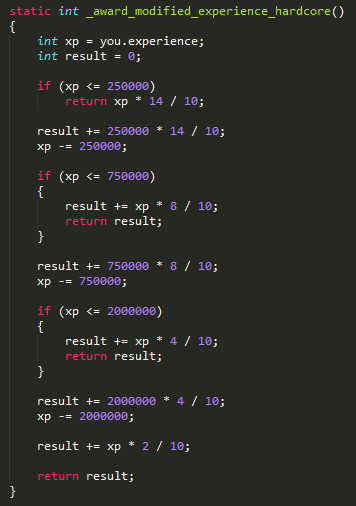
*Method hardcore*

*Function hardcore*

*Function hardcore*



Gambar 4.1 Bentuk *code* sistem skor sebelum pengubahan

Gambar 4.2 Bentuk *code* sistem skor setelah pengubahan

**4.4 Analisis**

Dari hasil eksperimen yang telah dilakukan, terdapat beberapa hal yang perlu diperhatikan:

1. Penerapan SPL pada *code* atau proyek yang sudah berjalan atau sudah jadi akan sangat berat untuk dilakukan
2. Pengerjaan tidak dapat dilakukan secara optimal karena banyaknya *class* dan *method* yang saling berhubungan
3. Sulit membuat perubahan pada *code* yang kurang menerapkan paradigma *object-oriented*

**BAB 5**

**PENUTUP**

Bab ini berisi kesimpulan dari hasil ekperimen dan saran untuk penelitian selanjutmya.

**5.1 Kesimpulan**

Berdasarkan analisis dari hasil eksperimen, dapat ditarik beberapa kesimpulan, yaitu:

1. Penerapan SPL sebaiknya dilakukan pada *code* yang menerapkan paradigma *object-oriented*
2. Sebelum membuat perubahan pada *code* yang kurang menerapkan paradigma *object-oriented*, sebaiknya dilakukan refactoring untuk mengubah *code* menjadi *object-oriented*.
3. Penerapan SPL akan menjadi kurang efisien jika diterapkan pada proyek yang sudah berjalan dengan ukuran *source code* yang besar, karena beban *refactoring* yang harus ditanggung sangat tinggi, terutama untuk kasus di mana paradigma *source code* tidak cocok dengan SPL.
4. SPL akan sangat membantu pembuatan variasi pada *game*, dengan kondisi *source code* dari *game* tersebut cocok untuk penerapan SPL.
5. Pembuatan variasi *game* tanpa penerapan SPL akan menyulitkan pengembang *game*, terutama untuk proyek-proyek dengan ukuran besar dengan keterkaitan suatu *class* dengan *class* lain, karena perubahan yang dilakukan berpotensi untuk menimbulkan *bug* yang tidak terduga.

Dari penelitian yang telah dilakukan, dapat disimpulkan bahwa SPL dapat diterapkan dengan langkah-langkah berikut:

1. Melakukan analisis pada domain *game*
2. Menganalisis fitur yang akan divariasikan
3. Melakukan *refactor* terhadap *code* untuk memudahkan penerapan SPL
4. Memisahkan seluruh fungsionalitas dari suatu versi dengan versi lainnya
   1. Membuat variasi dari *method-method* dan *function-function* yang berkaitan dengan aspek *game* yang ingin divariasikan.
   2. Membuat *class* yang berbeda untuk masing-masing variasi.
   3. Memindahkan *method-method* dan *function-function* tersebut ke dalam *class-class* yang berbeda sesuai dengan variasinya.

**5.2 Saran**

Untuk penelitian berikutnya, disarankan untuk memerhatikan beberapa hal berikut:

1. Untuk menunjang penelitian ini, disarankan melakukan penelitian dengan studi kasus yang berbeda. Terdapat banyak sekali jenis dan model proyek *game* yang berbeda dalam industri *game*, sehingga satu jenis contoh studi kasus saja tidak akan cukup untuk mencakup seluruh topik mengenai penerapan SPL dalam pengembangan *game*.
2. Jika penelitian yang dilakukan memiliki lingkup yang serupa, disarankan untuk memilih studi kasus dengan karakteristik yang berbeda dengan penelitian ini. Domain RPG memiliki banyak jenis *game* dengan struktur proyek yang berbeda-beda, sehingga akan lebih baik jika melakukan penelitian dengan studi kasus yang memiliki perbedaan karakteristik.
3. Selain jenis dan karakteristik studi kasus yang berbeda, disarankan pula untuk mengambil contoh variasi yang berbeda. Variasi yang berbeda terkadang membutuhkan langkah-langkah yang berbeda pula.
4. Untuk memudahkan penelitian, gunakan studi kasus yang menerapkan paradigma *object-oriented* dalam *source code*-nya, sehingga beban *refactoring* tidak terlalu berat.
5. Untuk penelitian skala kecil, gunakan studi kasus dengan ukuran secukupnya. Proyek yang memiliki ukuran sangat besar dan memiliki banyak sekali keterkaitan antar *function*-nya akan menyulitkan pembuatan variasi dalam penelitian.

**DAFTAR REFERENSI**

1. Frome, J. (2007). Eight Ways Videogames Generate Emotion. *DiGRA ’07 – Proceedings of the 2007 DiGRA International Conference: Situated Play*, 4:831–835.
2. Furtado, A. W. B., et al. (2011). Improving Digital Game Development with Software Product Lines. *IEEE Software*,28(5):30–37.
3. Nascimento, L. M., de Almeida, E. S., Meira, S. R. d. L. (2008). A Case Study in Software Product Lines – The Case of the Mobile Game Domain. *2008 34th Euromicro Conference Software Engineering and Advanced Applications*, 43–50.
4. Thiboust, J. (2013). Focusing Creativity: RPG Genres. <http://www.gamasutra.com/view/feature/185353/focusing_creativity_rpg_genres.php> Diakses pada tanggal 26 April 2016
5. Zhang, W., & Jarzabek, S. (2005). Reuse without Compromising Performance: Industrial Experience from RPG Software Product Line for Mobile Devices. *9th International Conference, SPLC 2005*, 57–69.

Referensi Gambar

Gambar 2.1 Furtado, A. W. B., et al. (2011). Improving Digital Game

Development with Software Product Lines.

Gambar 3.3 <http://ixbtlabs.com/articles2/digest3d/0407/pics-farcry.html>

Gambar 3.4 <http://www.neoseeker.com/minecraft/screenshots/>

Gambar 3.5 Furtado, A. W. B., et al. (2011). Improving Digital Game

Development with Software Product Lines.

**LAMPIRAN *SOURCE CODE***

1. Method yang diubah dari player.cc
2. int player\_res\_fire**(**bool calc\_unid**,** bool temp**,** bool items**)**
3. **{**
4. #if TAG\_MAJOR\_VERSION == 34
5. **if** **(**you**.**species **==** SP\_DJINNI**)**
6. **return** 4**;** // full immunity
7. #endif
8. int rf **=** 0**;**
9. **if** **(**items**)**
10. **{**
11. // rings of fire resistance/fire
12. rf **+=** you**.**wearing**(**EQ\_RINGS**,** RING\_PROTECTION\_FROM\_FIRE**,** calc\_unid**);**
13. rf **+=** you**.**wearing**(**EQ\_RINGS**,** RING\_FIRE**,** calc\_unid**);**
14. // rings of ice
15. rf **-=** you**.**wearing**(**EQ\_RINGS**,** RING\_ICE**,** calc\_unid**);**
16. // Staves
17. rf **+=** you**.**wearing**(**EQ\_STAFF**,** STAFF\_FIRE**,** calc\_unid**);**
18. // body armour:
19. const item\_def **\***body\_armour **=** you**.**slot\_item**(**EQ\_BODY\_ARMOUR**);**
20. **if** **(**body\_armour**)**
21. rf **+=** armour\_type\_prop**(**body\_armour**->**sub\_type**,** ARMF\_RES\_FIRE**);**
22. // ego armours
23. rf **+=** you**.**wearing\_ego**(**EQ\_ALL\_ARMOUR**,** SPARM\_FIRE\_RESISTANCE**);**
24. rf **+=** you**.**wearing\_ego**(**EQ\_ALL\_ARMOUR**,** SPARM\_RESISTANCE**);**
25. // randart weapons:
26. rf **+=** you**.**scan\_artefacts**(**ARTP\_FIRE**,** calc\_unid**);**
27. // dragonskin cloak: 0.5 to draconic resistances
28. **if** **(**calc\_unid **&&** player\_equip\_unrand**(**UNRAND\_DRAGONSKIN**)**
29. **&&** coinflip**())**
30. **{**
31. rf**++;**
32. **}**
33. **}**
34. // species:
35. **if** **(**you**.**species **==** SP\_MUMMY**)**
36. rf**--;**
37. #if TAG\_MAJOR\_VERSION == 34
38. **if** **(**you**.**species **==** SP\_LAVA\_ORC**)**
39. **{**
40. **if** **(**temperature\_effect**(**LORC\_FIRE\_RES\_I**))**
41. rf**++;**
42. **if** **(**temperature\_effect**(**LORC\_FIRE\_RES\_II**))**
43. rf**++;**
44. **if** **(**temperature\_effect**(**LORC\_FIRE\_RES\_III**))**
45. rf**++;**
46. **}**
47. #endif
48. // mutations:
49. rf **+=** player\_mutation\_level**(**MUT\_HEAT\_RESISTANCE**,** temp**);**
50. rf **-=** player\_mutation\_level**(**MUT\_HEAT\_VULNERABILITY**,** temp**);**
51. rf **-=** player\_mutation\_level**(**MUT\_TEMPERATURE\_SENSITIVITY**,** temp**);**
52. rf **+=** player\_mutation\_level**(**MUT\_MOLTEN\_SCALES**,** temp**)** **==** 3 **?** 1 **:** 0**;**
53. // spells:
54. **if** **(**temp**)**
55. **{**
56. **if** **(**you**.**duration**[**DUR\_RESISTANCE**])**
57. rf**++;**
58. **if** **(**you**.**duration**[**DUR\_FIRE\_SHIELD**])**
59. rf **+=** 2**;**
60. **if** **(**you**.**duration**[**DUR\_QAZLAL\_FIRE\_RES**])**
61. rf**++;**
62. rf **+=** get\_form**()->**res\_fire**();**
63. **}**
64. **if** **(**rf **>** 3**)**
65. rf **=** 3**;**
66. **if** **(**temp **&&** you**.**duration**[**DUR\_FIRE\_VULN**])**
67. rf**--;**
68. **if** **(**rf **<** **-**3**)**
69. rf **=** **-**3**;**
70. **return** rf**;**
71. **}**
72. Method yang diubah dari mon-behv.cc
73. void handle\_behaviour**(**monster**\*** mon**)**
74. **{**
75. // Test spawners should always be BEH\_SEEK against a foe, since
76. // their only purpose is to spew out monsters for testing
77. // purposes.
78. **if** **(**mon**->**type **==** MONS\_TEST\_SPAWNER**)**
79. **{**
80. **for** **(**monster\_iterator mi**;** mi**;** **++**mi**)**
81. **{**
82. **if** **(**mon**->**attitude **!=** mi**->**attitude**)**
83. **{**
84. mon**->**foe **=** mi**->**mindex**();**
85. mon**->**target **=** mi**->**pos**();**
86. mon**->**behaviour **=** BEH\_SEEK**;**
87. **return;**
88. **}**
89. **}**
90. **}**
91. bool changed **=** **true;**
92. bool isFriendly **=** mon**->**friendly**();**
93. bool isNeutral **=** mon**->**neutral**();**
94. bool wontAttack **=** mon**->**wont\_attack**()** **&&** **!**mon**->**has\_ench**(**ENCH\_INSANE**);**
95. // Whether the player position is in LOS of the monster.
96. bool proxPlayer **=** **!**crawl\_state**.**game\_is\_arena**()** **&&** mon**->**see\_cell**(**you**.**pos**());**
97. // If set, pretend the player isn't there, but only for hostile monsters.
98. **if** **(**proxPlayer **&&** crawl\_state**.**disables**[**DIS\_MON\_SIGHT**]** **&&** **!**mon**->**wont\_attack**())**
99. proxPlayer **=** **false;**
100. bool proxFoe**;**
101. bool isHealthy **=** **(**mon**->**hit\_points **>** mon**->**max\_hit\_points **/** 2**);**
102. bool isSmart **=** **(**mons\_intel**(**mon**)** **>=** I\_HUMAN**);**
103. bool isScared **=** mon**->**has\_ench**(**ENCH\_FEAR**);**
104. bool isPacified **=** mon**->**pacified**();**
105. bool patrolling **=** mon**->**is\_patrolling**();**
106. static vector**<**level\_exit**>** e**;**
107. static int e\_index **=** **-**1**;**
108. //mprf("AI debug: mon %d behv=%d foe=%d pos=%d %d target=%d %d",
109. // mon->mindex(), mon->behaviour, mon->foe, mon->pos().x,
110. // mon->pos().y, mon->target.x, mon->target.y);
111. // Check for confusion -- early out.
112. **if** **(**mon**->**has\_ench**(**ENCH\_CONFUSION**))**
113. **{**
114. set\_random\_target**(**mon**);**
115. **return;**
116. **}**
117. **if** **(**mons\_is\_fleeing\_sanctuary**(**mon**)**
118. **&&** mons\_is\_fleeing**(**mon**)**
119. **&&** is\_sanctuary**(**you**.**pos**()))**
120. **{**
121. **return;**
122. **}**
123. // Make sure monsters are not targeting the player in arena mode.
124. ASSERT**(!**crawl\_state**.**game\_is\_arena**()** **||** mon**->**foe **!=** MHITYOU**);**
125. // Validate current target exists.
126. \_mon\_check\_foe\_invalid**(**mon**);**
127. actor **\***owner **=** **(**mon**->**summoner **?** actor\_by\_mid**(**mon**->**summoner**)** **:** **nullptr);**
128. **if** **(**mon**->**type **==** MONS\_SPECTRAL\_WEAPON**)**
129. **{**
130. // Do nothing if we're still being placed
131. **if** **(!**mon**->**summoner**)**
132. **return;**
133. owner **=** actor\_by\_mid**(**mon**->**summoner**);**
134. **if** **(!**owner **||** **!**owner**->**alive**())**
135. **{**
136. end\_spectral\_weapon**(**mon**,** **false);**
137. **return;**
138. **}**
139. // Only go after the target if it's still near the owner, and
140. // so are we. The weapon is restricted to a leash range of 2,
141. // and things reachable within that leash range [qoala]
142. const int leash **=** 2**;**
143. **if** **(**grid\_distance**(**owner**->**pos**(),** mon**->**pos**())** **>** leash
144. **||** mon**->**foe **==** MHITNOT**)**
145. **{**
146. mon**->**foe **=** owner**->**mindex**();**
147. mon**->**target **=** owner**->**pos**();**
148. **}**
149. **else** **if** **(**mon**->**props**.**exists**(**SW\_TARGET\_MID**))**
150. **{**
151. actor **\***atarget **=** actor\_by\_mid**(**mon**->**props**[**SW\_TARGET\_MID**].**get\_int**());**
152. **if** **(**atarget **&&** atarget**->**alive**()**
153. **&&** **(**grid\_distance**(**owner**->**pos**(),** atarget**->**pos**())**
154. **<=** **((**mon**->**reach\_range**()** **==** REACH\_TWO**)** **?** leash **+** 2 **:** leash **+** 1**)))**
155. **{**
156. mon**->**target **=** atarget**->**pos**();**
157. mon**->**foe **=** atarget**->**mindex**();**
158. **}**
159. **else**
160. reset\_spectral\_weapon**(**mon**);**
161. **}**
162. **}**
163. // Change proxPlayer depending on invisibility and standing
164. // in shallow water.
165. **if** **(**proxPlayer **&&** **!**you**.**visible\_to**(**mon**))**
166. proxPlayer **=** \_monster\_guesses\_invis\_player**(\***mon**);**
167. // Set friendly target, if they don't already have one.
168. // Berserking allies ignore your commands!
169. **if** **(**isFriendly
170. **&&** **(**mon**->**foe **==** MHITNOT **||** mon**->**foe **==** MHITYOU**)**
171. **&&** **!**mon**->**berserk\_or\_insane**()**
172. **&&** mon**->**behaviour **!=** BEH\_WITHDRAW
173. **&&** mon**->**type **!=** MONS\_GIANT\_SPORE
174. **&&** mon**->**type **!=** MONS\_BALL\_LIGHTNING
175. **&&** **!**mons\_is\_avatar**(**mon**->**type**))**
176. **{**
177. **if** **(**you**.**pet\_target **!=** MHITNOT**)**
178. mon**->**foe **=** you**.**pet\_target**;**
179. **else** **if** **(**mons\_class\_is\_stationary**(**mon**->**type**))**
180. set\_nearest\_monster\_foe**(**mon**);**
181. **}**
182. // Instead, berserkers attack nearest monsters.
183. **if** **(**mon**->**behaviour **!=** BEH\_SLEEP
184. **&&** **(**mon**->**has\_ench**(**ENCH\_INSANE**)**
185. **||** **((**mon**->**berserk**()**
186. **||** mon**->**type **==** MONS\_GIANT\_SPORE
187. **||** mon**->**type **==** MONS\_BALL\_LIGHTNING**)**
188. **&&** **(**mon**->**foe **==** MHITNOT
189. **||** isFriendly **&&** mon**->**foe **==** MHITYOU**))))**
190. **{**
191. // Intelligent monsters prefer to attack the player,
192. // even when berserking.
193. **if** **(!**isFriendly
194. **&&** **!**mon**->**has\_ench**(**ENCH\_INSANE**)**
195. **&&** proxPlayer
196. **&&** mons\_intel**(**mon**)** **>=** I\_HUMAN**)**
197. **{**
198. mon**->**foe **=** MHITYOU**;**
199. **}**
200. **else**
201. set\_nearest\_monster\_foe**(**mon**);**
202. **}**
203. // Pacified monsters leaving the level prefer not to attack.
204. // Others choose the nearest foe.
205. // XXX: This is currently expensive, so we don't want to do it
206. // every turn for every monster.
207. **if** **(!**isPacified **&&** mon**->**foe **==** MHITNOT
208. **&&** mon**->**behaviour **!=** BEH\_SLEEP
209. **&&** **(**proxPlayer **||** one\_chance\_in**(**3**)))**
210. **{**
211. set\_nearest\_monster\_foe**(**mon**);**
212. **}**
213. // Friendly summons will come back to the player if they go out of sight.
214. **if** **(!**summon\_can\_attack**(**mon**))**
215. mon**->**target **=** you**.**pos**();**
216. // Monsters do not attack themselves. {dlb}
217. **if** **(**mon**->**foe **==** mon**->**mindex**())**
218. mon**->**foe **=** MHITNOT**;**
219. // Friendly and good neutral monsters do not attack other friendly
220. // and good neutral monsters.
221. **if** **(!**mons\_is\_avatar**(**mon**->**type**)** **&&** mon**->**foe **!=** MHITNOT **&&** mon**->**foe **!=** MHITYOU
222. **&&** wontAttack **&&** menv**[**mon**->**foe**].**wont\_attack**())**
223. **{**
224. mon**->**foe **=** MHITNOT**;**
225. **}**
226. // Neutral monsters prefer not to attack players, or other neutrals.
227. **if** **(**isNeutral
228. **&&** **!**mon**->**has\_ench**(**ENCH\_INSANE**)**
229. **&&** mon**->**foe **!=** MHITNOT
230. **&&** **(**mon**->**foe **==** MHITYOU **||** menv**[**mon**->**foe**].**neutral**()))**
231. **{**
232. mon**->**foe **=** MHITNOT**;**
233. **}**
234. // Unfriendly monsters fighting other monsters will usually
235. // target the player, if they're healthy.
236. **if** **(!**isFriendly **&&** **!**isNeutral
237. **&&** **!**mons\_is\_avatar**(**mon**->**type**)**
238. **&&** mon**->**foe **!=** MHITYOU **&&** mon**->**foe **!=** MHITNOT
239. **&&** proxPlayer **&&** **!**mon**->**berserk\_or\_insane**()**
240. **&&** isHealthy
241. **&&** **!**one\_chance\_in**(**3**))**
242. **{**
243. mon**->**foe **=** MHITYOU**;**
244. **}**
245. // Validate current target again.
246. \_mon\_check\_foe\_invalid**(**mon**);**
247. **if** **(**mon**->**has\_ench**(**ENCH\_HAUNTING**))**
248. **{**
249. actor**\*** targ **=** mon**->**get\_ench**(**ENCH\_HAUNTING**).**agent**();**
250. **if** **(**targ **&&** targ**->**alive**())**
251. **{**
252. mon**->**foe **=** targ**->**mindex**();**
253. mon**->**target **=** targ**->**pos**();**
254. **}**
255. **}**
256. **while** **(**changed**)**
257. **{**
258. const actor**\*** afoe **=** mon**->**get\_foe**();**
259. proxFoe **=** afoe **&&** mon**->**can\_see**(\***afoe**);**
260. **if** **(**mon**->**foe **==** MHITYOU**)**
261. **{**
262. // monster::get\_foe returns nullptr for friendly monsters with
263. // foe == MHITYOU, so make afoe point to the player here.
264. // -cao
265. afoe **=** **&**you**;**
266. proxFoe **=** proxPlayer**;** // Take invis into account.
267. **}**
268. coord\_def foepos **=** coord\_def**(**0**,**0**);**
269. **if** **(**afoe**)**
270. foepos **=** afoe**->**pos**();**
271. **if** **(**mon**->**pos**()** **==** mon**->**firing\_pos**)**
272. mon**->**firing\_pos**.**reset**();**
273. // Track changes to state; attitude never changes here.
274. beh\_type new\_beh **=** mon**->**behaviour**;**
275. unsigned short new\_foe **=** mon**->**foe**;**
276. // Take care of monster state changes.
277. **switch** **(**mon**->**behaviour**)**
278. **{**
279. **case** BEH\_SLEEP**:**
280. // default sleep state
281. mon**->**target **=** mon**->**pos**();**
282. new\_foe **=** MHITNOT**;**
283. **break;**
284. **case** BEH\_SEEK**:**
285. // No foe? Then wander or seek the player.
286. **if** **(**mon**->**foe **==** MHITNOT**)**
287. **{**
288. **if** **(**crawl\_state**.**game\_is\_arena**()**
289. **||** **!**proxPlayer **&&** **!**isFriendly
290. **||** isNeutral **&&** **!**mon**->**has\_ench**(**ENCH\_INSANE**)**
291. **||** patrolling
292. **||** mon**->**type **==** MONS\_GIANT\_SPORE
293. **||** mon**->**type **==** MONS\_BALL\_LIGHTNING**)**
294. **{**
295. new\_beh **=** BEH\_WANDER**;**
296. **}**
297. **else**
298. **{**
299. new\_foe **=** MHITYOU**;**
300. mon**->**target **=** you**.**pos**();**
301. **}**
302. **break;**
303. **}**
304. // just because a move takes us closer to the target doesn't mean
305. // that the move will stay in los of the target, and if it leaves
306. // los of the target, it's possible for just naively moving toward
307. // the target will not let us reach it (due to walls or whatever)
308. **if** **(!**mon**->**see\_cell**(**mon**->**target**))**
309. try\_pathfind**(**mon**);**
310. // Foe gone out of LOS?
311. **if** **(!**proxFoe
312. **&&** **!(**mon**->**friendly**()**
313. **&&** mon**->**foe **==** MHITYOU
314. **&&** mon**->**is\_travelling**()**
315. **&&** mon**->**travel\_target **==** MTRAV\_FOE**))**
316. **{**
317. // If their foe is marked, the monster always knows exactly
318. // where they are.
319. **if** **(**mons\_foe\_is\_marked**(**mon**)** **||** mon**->**has\_ench**(**ENCH\_HAUNTING**))**
320. **{**
321. mon**->**target **=** afoe**->**pos**();**
322. try\_pathfind**(**mon**);**
323. **break;**
324. **}**
325. // Maybe the foe is just invisible.
326. **if** **(**mon**->**target**.**origin**()** **&&** afoe **&&** mon**->**near\_foe**())**
327. **{**
328. \_guess\_invis\_foe\_pos**(**mon**);**
329. **if** **(**mon**->**target**.**origin**())**
330. **{**
331. // Having a seeking mon with a foe who's target is
332. // (0, 0) can lead to asserts, so lets try to
333. // avoid that.
334. set\_nearest\_monster\_foe**(**mon**);**
335. **if** **(**mon**->**foe **==** MHITNOT**)**
336. **{**
337. new\_beh **=** BEH\_WANDER**;**
338. **break;**
339. **}**
340. mon**->**target **=** mon**->**get\_foe**()->**pos**();**
341. **}**
342. **}**
343. **if** **(**mon**->**travel\_target **==** MTRAV\_MERFOLK\_AVATAR**)**
344. mon**->**travel\_target **=** MTRAV\_NONE**;**
345. // Spectral weapons simply seek back to their owner if
346. // they can't see their seek target.
347. **if** **(**mons\_is\_avatar**(**mon**->**type**))**
348. **{**
349. // XXX: should owner ever not be set here?
350. new\_foe **=** owner **?** owner**->**mindex**()** **:** MHITNOT**;**
351. mon**->**target **=** owner **?** owner**->**pos**()** **:** mon**->**pos**();**
352. **break;**
353. **}**
354. **else** **if** **(**isFriendly **&&** mon**->**foe **!=** MHITYOU**)**
355. **{**
356. **if** **(**patrolling **||** crawl\_state**.**game\_is\_arena**())**
357. **{**
358. new\_foe **=** MHITNOT**;**
359. new\_beh **=** BEH\_WANDER**;**
360. **}**
361. // If the player can see the target location, do not reset
362. // our target, even if this monster cannot (we'll assume
363. // the player passes along this information to allies)
364. // EXCEPTION: invisible enemies for allies without sinv
365. // (otherwise your allies get stuck doing nothing)
366. **else** **if** **(!**foepos**.**origin**()** **&&** you**.**see\_cell**(**foepos**)**
367. **&&** afoe**->**visible\_to**(**mon**))**
368. **{**
369. try\_pathfind**(**mon**);**
370. **}**
371. **else**
372. **{**
373. new\_foe **=** MHITYOU**;**
374. mon**->**target **=** foepos**;**
375. **}**
376. **break;**
377. **}**
378. ASSERT**(**mon**->**foe **!=** MHITNOT**);**
379. **if** **(**mon**->**foe\_memory **>** 0**)**
380. **{**
381. // If we've arrived at our target x,y
382. // do a stealth check. If the foe
383. // fails, monster will then start
384. // tracking foe's CURRENT position,
385. // but only for a few moves (smell and
386. // intuition only go so far).
387. **if** **(**mon**->**pos**()** **==** mon**->**target**)**
388. **{**
389. **if** **(**mon**->**foe **==** MHITYOU**)**
390. **{**
391. **if** **(**x\_chance\_in\_y**(**50**,** you**.**stealth**())**
392. **||** you**.**penance**[**GOD\_ASHENZARI**]** **&&** coinflip**())**
393. **{**
394. mon**->**target **=** you**.**pos**();**
395. **}**
396. **else**
397. mon**->**foe\_memory **=** 0**;**
398. **}**
399. **else**
400. **{**
401. **if** **(**coinflip**())** // XXX: cheesy!
402. mon**->**target **=** menv**[**mon**->**foe**].**pos**();**
403. **else**
404. mon**->**foe\_memory **=** 0**;**
405. **}**
406. **}**
407. **}**
408. **if** **(**mon**->**foe\_memory **<=** 0
409. **&&** **!(**mon**->**friendly**()** **&&** mon**->**foe **==** MHITYOU**))**
410. **{**
411. new\_beh **=** BEH\_WANDER**;**
412. **}**
413. // If the player walk out of the LOS of a monster with a ranged
414. // attack, we assume it sees in which direction the player went
415. // and it tries to find a line of fire instead of following the
416. // player.
417. **else** **if** **(**grid\_distance**(**mon**->**target**,** you**.**pos**())** **==** 1
418. **&&** \_mon\_tries\_regain\_los**(**mon**))**
419. **{**
420. \_set\_firing\_pos**(**mon**,** you**.**pos**());**
421. **}**
422. **else** //
423. mon**->**firing\_pos**.**reset**();**
424. **if** **(!**isFriendly**)**
425. **break;**
426. **else** **if** **(**mons\_is\_avatar**(**mon**->**type**)**
427. **&&** owner
428. **&&** **!**owner**->**is\_player**())**
429. **{**
430. mon**->**foe **=** owner**->**mindex**();**
431. **break;**
432. **}**
433. **}**
434. ASSERT**(**proxFoe **||** isFriendly**);**
435. ASSERT**(**mon**->**foe **!=** MHITNOT**);**
436. // Monster can see foe: set memory in case it loses sight.
437. // Hack: smarter monsters will tend to pursue the player longer.
438. **switch** **(**mons\_intel**(**mon**))**
439. **{**
440. **case** I\_HUMAN**:**
441. mon**->**foe\_memory **=** random\_range**(**450**,** 1000**);**
442. **break;**
443. **case** I\_ANIMAL**:**
444. mon**->**foe\_memory **=** random\_range**(**250**,** 550**);**
445. **break;**
446. **case** I\_BRAINLESS**:**
447. mon**->**foe\_memory **=** random\_range**(**100**,** 300**);**
448. **break;**
449. **}**
450. \_decide\_monster\_firing\_position**(**mon**,** owner**);**
451. **break;**
452. **case** BEH\_WANDER**:**
453. **if** **(**isPacified**)**
454. **{**
455. // If a pacified monster isn't travelling toward
456. // someplace from which it can leave the level, make it
457. // start doing so. If there's no such place, either
458. // search the level for such a place again, or travel
459. // randomly.
460. **if** **(**mon**->**travel\_target **!=** MTRAV\_PATROL**)**
461. **{**
462. new\_foe **=** MHITNOT**;**
463. mon**->**travel\_path**.**clear**();**
464. e\_index **=** mons\_find\_nearest\_level\_exit**(**mon**,** e**);**
465. **if** **(**e\_index **==** **-**1 **||** one\_chance\_in**(**20**))**
466. e\_index **=** mons\_find\_nearest\_level\_exit**(**mon**,** e**,** **true);**
467. **if** **(**e\_index **!=** **-**1**)**
468. **{**
469. mon**->**travel\_target **=** MTRAV\_PATROL**;**
470. patrolling **=** **true;**
471. mon**->**patrol\_point **=** e**[**e\_index**].**target**;**
472. mon**->**target **=** e**[**e\_index**].**target**;**
473. **}**
474. **else**
475. **{**
476. mon**->**travel\_target **=** MTRAV\_NONE**;**
477. patrolling **=** **false;**
478. mon**->**patrol\_point**.**reset**();**
479. set\_random\_target**(**mon**);**
480. **}**
481. **}**
482. **if** **(**pacified\_leave\_level**(**mon**,** e**,** e\_index**))**
483. **return;**
484. **}**
485. **if** **(**mon**->**strict\_neutral**()** **&&** mons\_is\_slime**(**mon**)**
486. **&&** you\_worship**(**GOD\_JIYVA**))**
487. **{**
488. set\_random\_slime\_target**(**mon**);**
489. **}**
490. // Is our foe in LOS?
491. // Batty monsters don't automatically reseek so that
492. // they'll flitter away, we'll reset them just before
493. // they get movement in handle\_monsters() instead. -- bwr
494. **if** **(**proxFoe **&&** **!**mons\_is\_batty**(**mon**)** **||** mons\_foe\_is\_marked**(**mon**))**
495. **{**
496. new\_beh **=** BEH\_SEEK**;**
497. **break;**
498. **}**
499. // Creatures not currently pursuing another foe are
500. // alerted by a sentinel's mark
501. **if** **(**mon**->**foe **==** MHITNOT **&&** you**.**duration**[**DUR\_SENTINEL\_MARK**]**
502. **&&** **(!**isFriendly **&&** **!**mons\_is\_avatar**(**mon**->**type**)** **&&** **!**isNeutral
503. **&&** **!**isPacified
504. **||** mon**->**has\_ench**(**ENCH\_INSANE**)))**
505. **{**
506. new\_foe **=** MHITYOU**;**
507. new\_beh **=** BEH\_SEEK**;**
508. **break;**
509. **}**
510. check\_wander\_target**(**mon**,** isPacified**);**
511. // During their wanderings, monsters will eventually relax
512. // their guard (stupid ones will do so faster, smart
513. // monsters have longer memories). Pacified monsters will
514. // also eventually switch the place from which they want to
515. // leave the level, in case their current choice is blocked.
516. **if** **(!**proxFoe **&&** **!**mons\_is\_avatar**(**mon**->**type**)** **&&** mon**->**foe **!=** MHITNOT
517. **&&** one\_chance\_in**(**isSmart **?** 60 **:** 20**)**
518. **&&** **!**mons\_foe\_is\_marked**(**mon**)**
519. **||** isPacified **&&** one\_chance\_in**(**isSmart **?** 40 **:** 120**))**
520. **{**
521. new\_foe **=** MHITNOT**;**
522. **if** **(**mon**->**is\_travelling**()** **&&** mon**->**travel\_target **!=** MTRAV\_PATROL
523. **||** isPacified**)**
524. **{**
525. #ifdef DEBUG\_PATHFIND
526. mpr**(**"It's been too long! Stop travelling."**);**
527. #endif
528. mon**->**travel\_path**.**clear**();**
529. mon**->**travel\_target **=** MTRAV\_NONE**;**
530. **if** **(**isPacified **&&** e\_index **!=** **-**1**)**
531. e**[**e\_index**].**unreachable **=** **true;**
532. **}**
533. **}**
534. **break;**
535. **case** BEH\_RETREAT**:**
536. // If the target can be reached, there is a chance the monster will
537. // try to attack. The chance is low to prevent the player from
538. // dancing in and out of the water.
539. try\_pathfind**(**mon**);**
540. **if** **(**one\_chance\_in**(**10**)** **&&** **!**target\_is\_unreachable**(**mon**)**
541. **||** mons\_can\_attack**(**mon**))**
542. **{**
543. new\_beh **=** BEH\_SEEK**;**
544. **}**
545. **else** **if** **(!**proxPlayer **&&** one\_chance\_in**(**5**))**
546. new\_beh **=** BEH\_WANDER**;**
547. **else** **if** **(**proxPlayer**)**
548. mon**->**target **=** foepos**;**
549. **break;**
550. **case** BEH\_FLEE**:**
551. // Check for healed.
552. **if** **(**isHealthy **&&** **!**isScared**)**
553. new\_beh **=** BEH\_SEEK**;**
554. // Smart monsters flee until they can flee no more...
555. // possible to get a 'CORNERED' event, at which point
556. // we can jump back to WANDER if the foe isn't present.
557. **if** **(**isFriendly**)**
558. **{**
559. // Special-cased below so that it will flee \*towards\* you.
560. **if** **(**mon**->**foe **==** MHITYOU**)**
561. mon**->**target **=** you**.**pos**();**
562. **}**
563. **else** **if** **(**proxFoe**)**
564. **{**
565. // Special-cased below so that it will flee \*from\* the
566. // correct position.
567. mon**->**target **=** foepos**;**
568. **}**
569. **break;**
570. **case** BEH\_CORNERED**:**
571. // If we were able to move since becoming cornered, resume fleeing
572. **if** **(**mon**->**pos**()** **!=** mon**->**props**[**"last\_pos"**].**get\_coord**())**
573. **{**
574. new\_beh **=** BEH\_FLEE**;**
575. mon**->**props**.**erase**(**"last\_pos"**);**
576. **}**
577. // Foe gone out of LOS?
578. **if** **(!**proxFoe**)**
579. **{**
580. **if** **((**isFriendly **||** proxPlayer**)**
581. **&&** **(!**isNeutral **||** mon**->**has\_ench**(**ENCH\_INSANE**))**
582. **&&** **!**patrolling
583. **&&** **!**crawl\_state**.**game\_is\_arena**())**
584. **{**
585. new\_foe **=** MHITYOU**;**
586. **}**
587. **else**
588. new\_beh **=** BEH\_WANDER**;**
589. **}**
590. **else**
591. mon**->**target **=** foepos**;**
592. **break;**
593. **case** BEH\_WITHDRAW**:**
594. **{**
595. **if** **(!**isFriendly**)**
596. **{**
597. new\_beh **=** BEH\_WANDER**;**
598. **break;**
599. **}**
600. bool stop\_retreat **=** **false;**
601. // We've approached our next destination, re-evaluate
602. **if** **(**grid\_distance**(**mon**->**target**,** mon**->**pos**())** **<=** 1**)**
603. **{**
604. // Continue on to the rally point
605. **if** **(**mon**->**target **!=** mon**->**patrol\_point**)**
606. mon**->**target **=** mon**->**patrol\_point**;**
607. // Reached rally point, stop withdrawing
608. **else**
609. stop\_retreat **=** **true;**
610. **}**
611. **else** **if** **(**grid\_distance**(**mon**->**pos**(),** you**.**pos**())** **>** LOS\_RADIUS **+** 2**)**
612. **{**
613. // We're too far from the player. Idle around and wait for
614. // them to catch up.
615. **if** **(!**mon**->**props**.**exists**(**"idle\_point"**))**
616. **{**
617. mon**->**props**[**"idle\_point"**]** **=** mon**->**pos**();**
618. mon**->**props**[**"idle\_deadline"**]** **=** you**.**elapsed\_time **+** 200**;**
619. **}**
620. mon**->**target **=** clamp\_in\_bounds**(**
621. mon**->**props**[**"idle\_point"**].**get\_coord**()**
622. **+** coord\_def**(**random\_range**(-**2**,** 2**),**
623. random\_range**(-**2**,** 2**)));**
624. **if** **(**you**.**elapsed\_time **>=** mon**->**props**[**"idle\_deadline"**].**get\_int**())**
625. stop\_retreat **=** **true;**
626. **}**
627. **else**
628. **{**
629. // Be more lenient about player distance if a monster is
630. // idling (to prevent it from repeatedly resetting idle
631. // time if its own wanderings bring it closer to the player)
632. **if** **(**mon**->**props**.**exists**(**"idle\_point"**)**
633. **&&** grid\_distance**(**mon**->**pos**(),** you**.**pos**())** **<** LOS\_RADIUS**)**
634. **{**
635. mon**->**props**.**erase**(**"idle\_point"**);**
636. mon**->**props**.**erase**(**"idle\_deadline"**);**
637. mon**->**target **=** mon**->**patrol\_point**;**
638. **}**
639. **if** **(**mon**->**pos**()** **==** mon**->**props**[**"last\_pos"**].**get\_coord**())**
640. **{**
641. **if** **(!**mon**->**props**.**exists**(**"blocked\_deadline"**))**
642. mon**->**props**[**"blocked\_deadline"**]** **=** you**.**elapsed\_time **+** 30**;**
643. **if** **(!**mon**->**props**.**exists**(**"idle\_deadline"**))**
644. mon**->**props**[**"idle\_deadline"**]** **=** you**.**elapsed\_time **+** 200**;**
645. **if** **(**you**.**elapsed\_time **>=** mon**->**props**[**"blocked\_deadline"**].**get\_int**()**
646. **||** you**.**elapsed\_time **>=** mon**->**props**[**"idle\_deadline"**].**get\_int**())**
647. **{**
648. stop\_retreat **=** **true;**
649. **}**
650. **}**
651. **else**
652. **{**
653. mon**->**props**.**erase**(**"blocked\_deadline"**);**
654. mon**->**props**.**erase**(**"idle\_deadline"**);**
655. **}**
656. **}**
657. **if** **(**stop\_retreat**)**
658. **{**
659. new\_beh **=** BEH\_SEEK**;**
660. new\_foe **=** MHITYOU**;**
661. mon**->**props**.**erase**(**"last\_pos"**);**
662. mon**->**props**.**erase**(**"idle\_point"**);**
663. mon**->**props**.**erase**(**"blocked\_deadline"**);**
664. mon**->**props**.**erase**(**"idle\_deadline"**);**
665. **}**
666. **else**
667. mon**->**props**[**"last\_pos"**]** **=** mon**->**pos**();**
668. **break;**
669. **}**
670. **default:**
671. **return;** // uh oh
672. **}**
673. changed **=** **(**new\_beh **!=** mon**->**behaviour **||** new\_foe **!=** mon**->**foe**);**
674. mon**->**behaviour **=** new\_beh**;**
675. **if** **(**mon**->**foe **!=** new\_foe**)**
676. mon**->**foe\_memory **=** 0**;**
677. mon**->**foe **=** new\_foe**;**
678. **}**
679. **}**
680. Method yang diubah dari attack.cc
681. int attack**::**calc\_to\_hit**(**bool random**)**
682. **{**
683. int mhit **=** attacker**->**is\_player**()** **?**
684. 15 **+** **(**you**.**dex**()** **/** 2**)**
685. **:** calc\_mon\_to\_hit\_base**();**
686. #ifdef DEBUG\_DIAGNOSTICS
687. const int base\_hit **=** mhit**;**
688. #endif
689. // This if statement is temporary, it should be removed when the
690. // implementation of a more universal (and elegant) to-hit calculation
691. // is designed. The actual code is copied from the old mons\_to\_hit and
692. // player\_to\_hit methods.
693. **if** **(**attacker**->**is\_player**())**
694. **{**
695. // fighting contribution
696. mhit **+=** maybe\_random\_div**(**you**.**skill**(**SK\_FIGHTING**,** 100**),** 100**,** random**);**
697. // weapon skill contribution
698. **if** **(**using\_weapon**())**
699. **{**
700. **if** **(**wpn\_skill **!=** SK\_FIGHTING**)**
701. **{**
702. **if** **(**you**.**skill**(**wpn\_skill**)** **<** 1 **&&** player\_in\_a\_dangerous\_place**())**
703. xom\_is\_stimulated**(**10**);** // Xom thinks that is mildly amusing.
704. mhit **+=** maybe\_random\_div**(**you**.**skill**(**wpn\_skill**,** 100**),** 100**,**
705. random**);**
706. **}**
707. **}**
708. **else** **if** **(**you**.**form\_uses\_xl**())**
709. mhit **+=** maybe\_random\_div**(**you**.**experience\_level **\*** 100**,** 100**,** random**);**
710. **else**
711. **{**
712. // Claws give a slight bonus to accuracy when active
713. mhit **+=** **(**player\_mutation\_level**(**MUT\_CLAWS**)** **>** 0
714. **&&** wpn\_skill **==** SK\_UNARMED\_COMBAT**)** **?** 4 **:** 2**;**
715. mhit **+=** maybe\_random\_div**(**you**.**skill**(**wpn\_skill**,** 100**),** 100**,**
716. random**);**
717. **}**
718. // weapon bonus contribution
719. **if** **(**using\_weapon**())**
720. **{**
721. **if** **(**weapon**->**base\_type **==** OBJ\_WEAPONS**)**
722. **{**
723. mhit **+=** weapon**->**plus**;**
724. mhit **+=** property**(\***weapon**,** PWPN\_HIT**);**
725. **}**
726. **else** **if** **(**weapon**->**base\_type **==** OBJ\_STAVES**)**
727. mhit **+=** property**(\***weapon**,** PWPN\_HIT**);**
728. **else** **if** **(**weapon**->**base\_type **==** OBJ\_RODS**)**
729. **{**
730. mhit **+=** property**(\***weapon**,** PWPN\_HIT**);**
731. mhit **+=** weapon**->**rod\_plus**;**
732. **}**
733. **}**
734. // slaying bonus
735. mhit **+=** slaying\_bonus**(**wpn\_skill **==** SK\_THROWING
736. **||** **(**weapon **&&** is\_range\_weapon**(\***weapon**)**
737. **&&** using\_weapon**()));**
738. // hunger penalty
739. **if** **(**you**.**hunger\_state **<=** HS\_STARVING**)**
740. mhit **-=** 5**;**
741. // armour penalty
742. mhit **-=** 2 **\*** **(**attacker\_armour\_tohit\_penalty **+** attacker\_shield\_tohit\_penalty**);**
743. // vertigo penalty
744. **if** **(**you**.**duration**[**DUR\_VERTIGO**])**
745. mhit **-=** 7**;**
746. // mutation
747. **if** **(**player\_mutation\_level**(**MUT\_EYEBALLS**))**
748. mhit **+=** player\_mutation\_level**(**MUT\_EYEBALLS**)** **+** 1**;**
749. // hit roll
750. mhit **=** maybe\_random2**(**mhit**,** random**);**
751. **}**
752. **else** // Monster to-hit.
753. **{**
754. **if** **(**using\_weapon**())**
755. mhit **+=** weapon**->**plus **+** property**(\***weapon**,** PWPN\_HIT**);**
756. const int jewellery **=** attacker**->**as\_monster**()->**inv**[**MSLOT\_JEWELLERY**];**
757. **if** **(**jewellery **!=** NON\_ITEM
758. **&&** mitm**[**jewellery**].**is\_type**(**OBJ\_JEWELLERY**,** RING\_SLAYING**))**
759. **{**
760. mhit **+=** mitm**[**jewellery**].**plus**;**
761. **}**
762. mhit **+=** attacker**->**scan\_artefacts**(**ARTP\_SLAYING**);**
763. **if** **(**using\_weapon**()** **&&** weapon**->**base\_type **==** OBJ\_RODS**)**
764. mhit **+=** weapon**->**rod\_plus**;**
765. **}**
766. // Penalties for both players and monsters:
767. mhit **-=** 5 **\*** attacker**->**inaccuracy**();**
768. **if** **(**attacker**->**confused**())**
769. mhit **-=** 5**;**
770. **if** **(**using\_weapon**()** **&&** is\_unrandom\_artefact**(\***weapon**,** UNRAND\_WOE**))**
771. **return** AUTOMATIC\_HIT**;**
772. // If no defender, we're calculating to-hit for debug-display
773. // purposes, so don't drop down to defender code below
774. **if** **(**defender **==** **nullptr)**
775. **return** mhit**;**
776. **if** **(!**defender**->**visible\_to**(**attacker**))**
777. **if** **(**attacker**->**is\_player**())**
778. mhit **-=** 6**;**
779. **else**
780. mhit **=** mhit **\*** 65 **/** 100**;**
781. **else**
782. **{**
783. // This can only help if you're visible!
784. const int how\_transparent **=** player\_mutation\_level**(**MUT\_TRANSLUCENT\_SKIN**);**
785. **if** **(**defender**->**is\_player**()** **&&** how\_transparent**)**
786. mhit **-=** 2 **\*** how\_transparent**;**
787. **if** **(**defender**->**backlit**(false))**
788. mhit **+=** 2 **+** random2**(**8**);**
789. **else** **if** **(!**attacker**->**nightvision**()**
790. **&&** defender**->**umbra**())**
791. mhit **-=** 2 **+** random2**(**4**);**
792. **}**
793. // Don't delay doing this roll until test\_hit().
794. **if** **(!**attacker**->**is\_player**())**
795. mhit **=** random2**(**mhit **+** 1**);**
796. dprf**(**DIAG\_COMBAT**,** "%s: Base to-hit: %d, Final to-hit: %d"**,**
797. attacker**->**name**(**DESC\_PLAIN**).**c\_str**(),**
798. base\_hit**,** mhit**);**
799. **return** mhit**;**
800. **}**
801. Method yang diubah dari hiscores.cc
802. static int \_award\_modified\_experience**()**
803. **{**
804. int xp **=** you**.**experience**;**
805. int result **=** 0**;**
806. **if** **(**xp **<=** 250000**)**
807. **return** xp **\*** 7 **/** 10**;**
808. result **+=** 250000 **\*** 7 **/** 10**;**
809. xp **-=** 250000**;**
810. **if** **(**xp **<=** 750000**)**
811. **{**
812. result **+=** xp **\*** 4 **/** 10**;**
813. **return** result**;**
814. **}**
815. result **+=** 750000 **\*** 4 **/** 10**;**
816. xp **-=** 750000**;**
817. **if** **(**xp **<=** 2000000**)**
818. **{**
819. result **+=** xp **\*** 2 **/** 10**;**
820. **return** result**;**
821. **}**
822. result **+=** 2000000 **\*** 2 **/** 10**;**
823. xp **-=** 2000000**;**
824. result **+=** xp **/** 10**;**
825. **return** result**;**
826. **}**