PENGEMBANGAN *GAME* EDUKASI KIMIA *ADVENTURE OF LITTLE ALCHEMIST*

TUGAS AKHIR
Diajukan untuk Memenuhi Salah Satu Syarat Kelulusan
Program Pendidikan Sarjana

Oleh :
Jimmy Sanjaya
2017130041

JURUSAN INFORMATIKA
SEKOLAH TINGGI MANAJEMEN INFORMATIKA & KOMPUTER - LIKMI
BANDUNG
2021

PENGEMBANGAN *GAME* EDUKASI KIMIA *ADVENTURE OF LITTLE ALCHEMIST*

Oleh:
Jimmy Sanjaya
2017130041

Bandung, 30 Juli 2021
Menyetujui,

Maria Christina, S.Kom., M.Kom.	Dhanny Setiawan, S.T., M.T.
Ko-Pembimbing	Pembimbing

Dhanny Setiawan, S.T., M.T.
Ketua Jurusan

JURUSAN INFORMATIKA
SEKOLAH TINGGI MANAJEMEN INFORMATIKA & KOMPUTER LIKMI
BANDUNG
2021

ABSTRAK

Penelitian ini berjudul "***PENGEMBANGAN GAME EDUKASI KIMIA ADVENTURE OF LITTLE ALCHEMIST***". Tujuan dilakukan penelitian ini adalah untuk mengembangkan *game* edukasi dengan genre role playing game, sebagai sarana bagi siswa memahami pelajaran mengenai zat aditif dan zat adiktif yang menceritakan tentang seorang anak bernama Vincent dengan tugas membantu penduduk desa dan mengalahkan naga-naga yang terdapat di setiap desa. Berdasarkan cerita dalam *game* akan dibentuk beberapa *main quest* yang harus pemain selesaikan. Seluruh *quest* yang terdapat dalam *game* akan dilengkapi dengan dialog yang berguna memberi petunjuk bagi pemain melanjutkan cerita. Karakter yang terdapat dalam game memiliki status yang terdiri dari *hit point*, *magic point*, *attack*, *defense*, *magic attack*, *magic defense*, *agility*, dan *luck* yang akan berkembang sesuai dengan kenaikan *level* karakter. *Game* yang dibuat diharapkan dapat menambah wawasan pemain dengan cara yang menyenangkan. *Game* dirancang untuk dapat berjalan pada perangkat komputer berbasis sistem operasi *windows*.

Penelitian ini menggunakan metode penelitian berorientasi objek dan model pengembangan yang digunakan adalah model *Prototyping*. Analisis dan perancangan pada penelitian ini akan terdiri dari beberapa diagram UML antara lain *use case* diagram, skenario diagram, *class* diagram, dan *activity* diagram. Metode pengujian yang dilakukan pada penelitian ini adalah *blackbox* dan pengujian dari pihak *expertise* berupa respon kuesioner. Pada pengujian akan ditampilkan antar muka yang sudah diimplementasi dan pengujian fungsi yang membandingkan antara hasil yang diharapkan dengan hasil dari pengujian. Perangkat lunak yang digunakan dalam pengembangan *game* adalah *RPG Maker MV*.

Hasil dari penelitian ini adalah game dapat dibuat menggunakan perangkat lunak *RPG Maker MV* dan semua fungsionalitas yang ada di dalamnya dapat berfungsi dengan baik. *Game* yang dibuat dengan tujuan membantu siswa belajar mengenai zat aditif dan zat adiktif, mudah dimengerti dan mudah digunakan oleh pemain berdasarkan hasil dari kuesioner.

Kata kunci : adiktif, aditif, edukasi, *game*, *prototyping*, RPG, UML

KATA PENGANTAR

Puji syukur penulis panjatkan kepada Tuhan Yang Maha Esa, karena berkat serta rahmat-Nya sehingga peneliti akhirnya dapat menempuh serta menyelesaikan Tugas Akhir ini dengan baik. Tugas Akhir ini berjudul "PENGEMBANGAN GAME EDUKASI KIMIA ADVENTURE OF LITTLE ALCHEMIST" disusun untuk memenuhi persyaratan akademik serta mendapatkan gelar Sarjana Strata 1 (S1), di Sekolah Tinggi Manajemen Informatika dan Komputer LIKMI, Bandung.

Peneliti mengakui bahwa penulisan penelitian ini jauh dari kata sempurna, masih banyak kekurangan dalam diri peneliti untuk menyusun penelitian ini oleh sebab itu, peneliti terbuka dan menerima setiap masukan maupun kritikan dari para pembaca.

Dalam penyusunan penelitian ini, peneliti menyadari bahwa tidak lepas dari bantuan dan dorongan baik secara langsung atau tidak langsung dari berbagai pihak sehingga peneliti dapat menyelesaikan Tugas Akhir ini. Untuk itu perkenankanlah peneliti menyampaikan rasa terima kasih yang sebesar-besarnya dan penghargaan yang tidak terhingga terutama yang terhormat :

1. Bapak Dhanny Setiawan, S.T., M.T. selaku dosen pembimbing dan ketua jurusan program studi S1 Informatika di STMIK LIKMI yang telah memberikan dukungan serta arahan kepada penulis.
2. Ibu Maria Christina, S.Kom., M.Kom. selaku dosen ko-pembimbing yang sudah banyak membantu penulis baik dalam meluangkan waktu, tenaga, dan pikiran untuk memberikan banyak saran dan arahan dalam membimbing peneliti mengerjakan Tugas Akhir ini.
3. Kedua orang tua serta keluarga besar yang telah mendukung dan memberi motivasi dan fasilitas selama penulis menyelesaikan tugas akhir ini.
4. STMIK LIKMI yang telah memberikan kesempatan bagi peneliti untuk menempuh pendidikan serta seluruh dosen di STMIK LIKMI atas ilmu dan kesabarannya dalam memberikan pengajaran yang terbaik.

5. Seluruh teman, sahabat bahkan kekasih peneliti yaitu Elvying Chen yang selalu mendukung, membantu dan memberikan motivasi dalam menyelesaikan tugas akhir ini.
6. Vinsensius Christian Ferry, Kanisius Adi Prasetyo, Bernardus Dwi Wahyu, Kristianto Wibawa sebagai teman seperjuangan dan bertukar pikiran penulis.
7. Semua pihak yang terlibat secara tidak langsung dalam penyusunan Tugas Akhir ini dimana peneliti tidak dapat disebutkan satu-persatu.

Akhir kata peneliti hanya dapat berterima kasih dan berdoa sekiranya Tuhan memberikan berkat karunia dan rahmat-Nya serta membalas segala kebaikan pada semua pihak yang telah banyak membantu peneliti. Peneliti berharap semoga Tugas Akhir ini dapat bermanfaat bagi para pembaca.

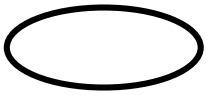
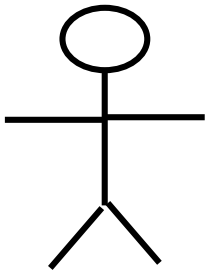
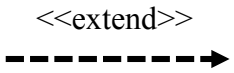
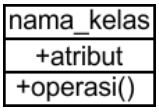
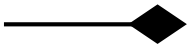
Bandung, 30 Juli 2021

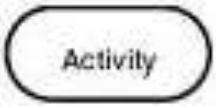
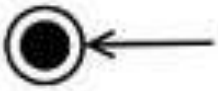
Penulis

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR SIMBOL

Simbol	Nama Simbol	Deskripsi Simbol
	*Use Case*	Simbol yang menggambarkan fungsionalitas yang diharapkan dari sistem yang dibangun.
	*Actor*	Simbol untuk entitas eksternal dengan nama dan deskripsi unik yang berinteraksi dengan sistem melalui komponen *use case*. *Actor* tidak hanya untuk menggambarkan elemen sebagai manusia, namun dapat juga berupa perangkat lunak, perangkat keras dan sistem
	Communication	Simbol yang menunjukkan bahwa *actor* berkomunikasi dengan sistem dan menggunakan fungsi tertentu yang digambarkan berupa garis tegas yang dihubungkan antara simbol *actor* dan *use case*.
	*Extend*	Simbol ini termasuk dalam simbol *relationship* dengan fungsi serupa seperti *include* yang memiliki perbedaan yaitu *base use case* dapat menggunakan perilaku yang terdapat pada *extending use case*.
	*Class*	Class adalah sebuah spesifikasi yang jika diinstansiasi akan menghasilkan sebuah objek dan merupakan inti dari pengembangan dan desain berorientasi objek.
	*Composition*	Simbol yang melambangkan hubungan dimana child class tidak bisa berdiri sendri tanpa parent class.

Simbol	Nama Simbol	Deskripsi Simbol
	*Activity*	Simbol yang melambangkan eksekusi suatu tindakan pada objek atau oleh objek. Pada dasarnya setiap tindakan atau peristiwa yang terjadi diwakili menggunakan simbol ini.
	*Start*	Simbol yang melambangkan titik kondisi awal sebelum aktivitas berjalan.
	*Stop*	Simbol yang melambangkan kondisi akhir dari suatu proses atau aktifitas.
	*Decision* & *Merge*	Simbol melambangkan keputusan dengan beberapa jalur alternatif. Ketika sebuah activity membutuhkan keputusan sebelum menuju activity berikutnya maka perlu menambahkan simbol ini diantara kedua activity tersebut. Alternatif yang dihasilkan harus diberi label dengan kondisi.
[condition] 	*Control Flow*	Simbol ini menggambarkan transisi dari satu *activity* ke yang lain dan biasanya digambar dengan garis panah. Kondisi dari transisi dapat ditambahkan pada bagian atas simbol.

DAFTAR LAMPIRAN

DAFTAR ISI

BAB I

PENDAHULUAN

1.1 Latar Belakang Masalah

Pada masa sekarang ini perkembangan *game* sangatlah pesat, karena didukung oleh kemajuan teknologi sehingga banyak orang-orang yang memiliki akses terhadap komputer, laptop atau perangkat *mobile* lainnya. *Game* sendiri sudah menjadi hal yang tidak asing di kehidupan, banyak orang yang mengisi waktu luangnya dengan bermain *game* sebagai sumber hiburan. Tidak sedikit juga yang bahkan menghabiskan uang yang mereka miliki untuk membeli barang-barang di dalam *game* demi mendapat keuntungan, kekuatan atau bahkan hanya untuk sekedar mengganti penampilan yang mereka miliki di dalam *game* tersebut. Tahun 2002 merupakan tahun rilis dari sebuah *game RPG* yang berjudul *Ragnarok Online*, sebuah *game* yang populer pada masanya dimana *game* ini menyediakan fitur-fitur seperti *class, change job,* dan *trading*.

Definisi *RPG* menurut W. Wibawanto adalah : "*RPG adalah permainan di mana pemain menjalankan peran karakter dalam sebuah cerita fiksi*." (Wibawanto, 2020:7) Di dalam *game RPG* ini setiap pemain harus memerankan karakter sesuai pilihannya masing-masing contohnya *Swordsman, Mage, Assassin*, dan sebagainya. Secara umum pemain akan dihadapkan pada sebuah permasalahan dan diberikan misi atau yang sering dikenal dengan istilah *quest* untuk menyelesaikan permasalahan tersebut. Sebuah *game RPG* biasanya memiliki sebuah cerita, elemen ini berperan penting dalam membawa pemain untuk mengembangkan karakter, menemukan masalah, dan menyelesaikan masalah. Kemudian ada pula *Non Player Character* atau disingkat dengan NPC, karakter ini selalu dibutuhkan sebagai penyusun cerita dimana NPC akan memberikan informasi, memberikan *quest*, menyediakan *item-item* atau hanya sebagai pemanis dalam *game*.

Ketika pemain bermain *game RPG*, seringkali mereka mendapat pengetahuan yang ada di dalam *game* tersebut seperti sains, bahasa, politik, sejarah, dan beberapa

bidang lain. Tentunya pengetahuan tentang kimia menjadi salah satu pengetahuan yang menjadi fokus utama dalam *game* yang akan dibuat. Berdasarkan situs (news.detik.com), di masa pandemi seperti ini Kementerian Pendidikan dan Kebudayaan atau disingkat menjadi Kemdikbud telah melakukan survei dan mendapatkan kesimpulan bahwa rata-rata siswa kurang memahami pelajaran yang diberikan gurunya saat kegiatan belajar jarak jauh, salah satu yang menjadi alasannya adalah siswa merasa kesulitan untuk berkonsentrasi saat belajar di rumah. *Game* yang menarik dapat membuat pemainnya menjadi fokus dan konsentrasi, hal inilah yang diharapkan mampu mengatasi masalah kurangnya konsentrasi pada siswa yang belajar di rumah.

Kimia menjadi salah satu mata pelajaran wajib yang terdapat dalam kurikulum sekolah, namun sebagian siswa masih menganggap sulit pelajaran kimia. Hal ini tentu berdampak terhadap motivasi belajar para siswa. Inovasi diperlukan untuk merubah pola pikir para siswa agar menyukai pelajaran kimia seperti menggunakan sarana teknologi yang sudah berkembang pesat menjadi sarana belajar siswa. Siswa SMP merupakan anak-anak remaja dimana mereka cenderung lebih suka bermain, mereka dapat menghabiskan berjam-jam untuk bermain *game* dalam satu hari. *Game* yang dikemas menjadi sarana pendidikan diharapkan mampu meningkatkan motivasi para siswa untuk mempelajari kimia secara menyenangkan, karena dalam *game* terdapat animasi, serta cerita yang membuat pemain tidak merasa bosan.

Zat aditif dan zat adiktif sangat mudah ditemukan dalam kehidupan sehari-hari, dimana kedua zat ini memiliki efek yang merugikan masyarakat dalam hal ini yaitu zat adiktif. Oleh karena itu, kedua zat ini diajarkan di SMP kelas 8 mata pelajaran IPA sehingga siswa diharapkan mampu untuk membedakan zat yang bermanfaat maupun yang merugikan. Dengan menggunakan *game* edukasi kimia zat aditif dan zat adiktif sebagai bahan pembelajaran, para siswa diharapkan dapat menjadi lebih tertarik mempelajari kimia dan dapat memahami materi pelajaran dengan lebih baik.

RPG Maker MV merupakan sebuah program untuk membuat sebuah *game RPG* 2 dimensi, menggunakan *Javascript* sebagai bahasa pemrogramannya. *Software* ini memiliki fitur-fitur seperti *database* yang dibedakan dalam bentuk *tab*, setiap *tab* memiliki kategori masing-masing seperti kategori untuk *monster*, karakter, *item*, dan kategori lainnya. Fitur selanjutnya adalah *Map Editor* dimana pengguna bisa membuat peta untuk dijelajah oleh karakter, selain itu terdapat juga fitur *Event* untuk memberikan sebuah interaksi antara karakter dengan NPC, atau untuk perpindahan *map* dan sebagainya.

Dari penjelasan latar belakang diatas, maka akan dibuat sebuah *game RPG* yang berjudul "***PENGEMBANGAN GAME EDUKASI ADVENTURE OF LITTLE ALCHEMIST***".

1.2 Rumusan Masalah

Berdasarkan latar belakang di atas, maka rumusan masalah yang dapat disimpulkan adalah bagaimana membuat aplikasi *game* edukasi kimia yang mudah dimengerti dan mudah digunakan menggunakan *RPG Maker MV* ?

1.3 Tujuan

Berdasarkan rumusan masalah diatas maka tujuan yang ingin dicapai dari penelitian ini adalah membuat aplikasi *game* edukasi kimia yang mudah dimengerti dan mudah digunakan menggunakan *RPG Maker MV*.

1.4 Batasan Masalah

Batasan-batasan masalah adalah sebagai berikut :

1. Aplikasi *game* ini dibuat menggunakan *RPG Maker MV* dan ditujukan untuk pengguna komputer yang mempunyai sistem operasi *Windows* versi 7/8/8.1/10.
2. *Game* ini hanya bisa dimainkan oleh satu pengguna.
3. Aplikasi *game* ini hanya bisa dimainkan secara *offline*.

4. Aplikasi *game* ditujukan sebagai bahan pembelajaran bagi murid Sekolah Menengah Pertama kelas 8.
5. Materi kimia yang akan dijelaskan meliputi Zat Aditif dan Zat Adiktif.
6. *Item* dalam *game* yang digunakan sebagai zat aditif adalah gula dan garam.
7. *Item* dalam *game* yang digunakan sebagai zat adiktif adalah kafein dan alkohol.

1.5 Kegunaan Hasil

Kegunaan aplikasi yang dibuat adalah :

1. Dapat menjadi sarana untuk siswa yang mengalami kesulitan belajar di kelas.
2. Dapat menambah wawasan siswa dengan cara yang menyenangkan.
3. Materi pelajaran dapat tersampaikan dengan baik kepada siswa

1.6 Metodologi Penelitian

1. Studi Pustaka

   Pengumpulan data-data dan informasi dari berbagai sumber seperti jurnal, buku, internet, mengenai *Game,* Edukasi*,* Kimia*,* dan *RPG Maker MV* .

2. Perancangan Sistem

   Menentukan fitur-fitur apa saja yang akan dibuat, alur cerita *game* menggunakan *RPG Maker MV* tersebut.

3. Implementasi Sistem

   Melakukan konstruksi *game* dengan mengimplementasikan rancangan sistem yang sudah dibuat sebelumnya.

4. Pengujian dan Perbaikan

   Melakukan pengujian kepada *game* yang sudah dibuat dengan cara mencobanya lalu kemudian memperbaiki kesalahan yang ada.

1.7 Sistematika Penulisan

Bab I Pendahuluan : Bab ini akan menjelaskan tentang latar belakang masalah, rumusan masalah, tujuan aplikasi dibuat, batasan masalah, kegunaan hasil, metodologi penelitian, dan sistematika penulisan.

Bab II Landasan Teori : Bab ini terdiri dari sumber-sumber literatur yang digunakan untuk menyusun landasan teori, akan dijelaskan mengenai teori-teori rekayasa perangkat lunak, metodologi penelitian, *Unified Modeling Language*, model pengembangan, *game*, edukasi, kimia, dan perangkat lunak yang menjadi alat bantu untuk membuat aplikasi yaitu *RPG Maker MV*.

Bab III Analisis dan Perancangan Perangkat Lunak : Bab ini menjelaskan secara garis besar hasil aplikasi program tugas akhir, spesifikasi kebutuhan perangkat lunak, diagram-diagram perancangan perangkat lunak, dan rancangan antar muka aplikasi program yang dibuat.

Bab IV Implementasi dan Pengujian Perangkat Lunak : Bab ini berisi rincian perangkat keras yang penulis pakai untuk menguji aplikasi, kemudian pengujian antar muka, lalu pengujian fungsi perangkat lunak.

Bab V Kesimpulan dan Saran : Bab ini menjelaskan hasil yang diperoleh dengan adanya perancangan perangkat lunak, dan usulan-usulan perancangan perangkat lunak.

BAB II

LANDASAN TEORI

2.1 Rekayasa Perangkat Lunak

2.1.1 Definisi Rekayasa Perangkat Lunak

Rekayasa Perangkat Lunak atau sering disebut sebagai *Software Engineering* merupakan istilah yang akan sering didengar jika ingin membangun sebuah *software*. Perkembangan perangkat lunak saat ini semakin menyesuaikan dengan kebutuhan manusia, salah satu contohnya adalah dalam mendukung perkembangan di bidang bisnis, perangkat lunak dituntut memenuhi kebutuhan-kebutuhan dan tantangan dari dunia bisnis yang semakin berkembang karena kemajuan teknologi. Berikut beberapa pengertian rekayasa perangkat lunak menurut beberapa ahli :

Roger S. Pressman dalam bukunya yang berjudul "*Software Engineering: A Practioner's Approach Eighth Edition*" mengatakan bahwa rekayasa perangkat lunak adalah : "*Software engineering encompasses a process, a collection of methods (practice) and an array of tools that allow professionals to build high-quality computer software.*" (Pressman and Maxim, 2015). Dari kutipan tersebut maka dapat disimpulkan bahwa rekayasa perangkat lunak meliputi sebuah proses, metode, dan berbagai macam alat dengan tujuan untuk membangun perangkat lunak berkualitas tinggi.

Rekayasa perangkat lunak menurut Lila Setiyani dalam buku "*REKAYASA PERANGKAT LUNAK*" yaitu : "*Suatu program yang dapat beroperasi untuk memanipulasi informasi, yang di dalamnya terdapat instruksi – instruksi berupa fitur atau fungsi tertentu sehingga informasi yang telah di peroleh dapat tercetak dalam bentuk maya*" (Setiyani, 2019:2)

Berdasarkan pengertian beberapa ahli diatas mengenai definisi perangkat lunak, maka dapat disimpulkan bahwa rekayasa perangkat lunak adalah suatu proses, metode

yang di dalamnya terdapat instruksi-instruksi berupa fungsi tertentu dengan tujuan untuk menghasilkan sebuah perangkat lunak yang berkualitas tinggi.

2.1.2 Metodologi Penelitian Berorientasi Objek

Pada masa awal perkembangan perangkat lunak, metode yang kebanyakan digunakan adalah terstruktur, dimana metode ini cocok digunakan untuk menyelesaikan masalah yang kecil karena tidak banyak dilakukan perubahan yang berarti, namun akan jadi sulit ketika masalah yang dihadapi rumit. Metode ini juga memiliki kekurangan karena cenderung tidak dapat menggunakan kembali kode yang sebelumnya sudah dibuat.

Metodologi berorientasi objek merupakan suatu cara membangun sistem perangkat lunak yang berorientasikan kepada objek, dan masing-masing objek tersebut memiliki atribut dan *method*. Atribut adalah data yang mewakili keadaan sebuah objek, sedangkan *method* adalah perilaku yang menunjukan kemampuan yang dimiliki oleh objek tersebut. Metode ini dapat mengatasi kelemahan dari metode terstruktur karena metode orientasi objek terdiri dari kelas-kelas yang memisahkan setiap kode program menjadi kelompok-kelompok kecil sesuai dengan fungsinya. Metode ini juga memiliki kelebihan karena kode program dapat digunakan kembali, objek dan kelas dapat digunakan berulang-ulang jika dibandingkan dengan metode terstruktur yang cenderung tidak *re-usable*.

Menurut buku yang ditulis oleh Bernd Bruegge dan Allen H. Dutoit yang berjudul "*Object-oriented software Engineering : Using UML patterns, and Java*" terdapat enam aktivitas pengembangan dalam pengembangan perangkat lunak berorientasi objek, yakni *requirements elicitation, analysis, system design, object design, implementation,* dan *testing* (Bruegge & Dutoit, 2014:6) berikut penjelasan aktivitasnya :

1. *Requirements Elicitation*

   Proses mendefinisikan kebutuhan sistem, hasilnya adalah spesifikasi sistem yang *client* butuhkan dan juga analisa model yang pengembang sanggup intepretasikan. Elisitasi kebutuhan berbicara tentang komunikasi antara pengembang, *client*, dan

pengguna untuk mendefinisikan sebuah sistem baru, jadi komunikasi adalah kunci agar sistem yang dihasilkan dapat mudah digunakan nantinya.

2. *Analysis*

   Pengembang merumuskan spesifikasi kebutuhan yang dihasilkan dari tahap sebelumnya dan memeriksa lebih detil jika terdapat kesalahan atau sesuatu yang ambigu. *Client* dan pengguna biasanya dilibatkan pada aktivitas ini jika ada spesifikasi kebutuhan yang perlu diganti atau jika terdapat informasi tambahan yang diperlukan. Bagian ini berfokus pada identifikasi objek-objek, perilaku objek, dan relasi antar objek. Tahap analisis berguna untuk memodelkan sebuah sistem yang tepat, lengkap, konsisten, dan dapat diverifikasi.

3. *System Design*

   System Design merupakan proses mentransformasikan model analisis menjadi model desain. Selama fase ini pihak pengembang mendefinisikan tujuan dari proyek serta mendekomposisikan sistem kedalam *subsystem* sehingga dapat direalisasikan oleh masing-masing tim, pengembang juga memilih strategi untuk perancangan sistem seperti perangkat keras atau perangkat lunak. *Subsystem* tahap awal biasanya belum memenuhi semua tujuan desain, maka dari itu pengembang terus memperbaiki *subsystem* yang ada sampai semua tujuan tercapai.

4. *Object Design*

   Jika kita ibaratkan sebuah *puzzle*, setelah beberapa kali tahap analisis dan desain sistem, pengembang biasanya disisakan hanya beberapa potongan *puzzle* saja yang hilang, dan tahap desain objek inilah yang akan menemukan bagian yang hilang itu. Pihak pengembang menentukan solusi untuk menjembatani tahap analisis dan tahap desain sistem dan desain objek menghasilkan model objek yang lebih terperinci dan deskripsi yang lebih tepat untuk setiap elemen.

5. *Implementation*

   Tahap implementasi merupakan aktivitas untuk mengintegrasi semua objek, sehingga dapat terciptanya sebuah sistem yang berfungsi dengan baik. Tahap integrasi merupakan implementasi dari setiap atribut dan *method* dari setiap objek serta mencakup *source code* dengan menggunakan konsep pemrograman orientasi objek.

6. *Testing*

   Pengujian merupakan sebuah proses untuk menemukan perbedaan antara *expected behavior* dari model sistem dan *observed behavior* dari sistem yang diimplementasikan. Ketika perbedaan d*item*ukan, pihak pengembang mengidentifikasi cacat dan memodifikasi sistem untuk memperbaikinya. Tujuan pengujian adalah untuk merusak sistem, biasanya yang melakukan pengujian adalah pihak pengembang dari luar yang tidak membangun sistem. Pengujian untuk setiap aktivitas meliputi inspeksi komponen, yaitu menemukan kesalahan pada komponen individu melalui pemeriksaan terhadap *source code*, kemudian *usability testing* yaitu untuk menemukan perbedaan antara apa yang sistem lakukan dan ekspektasi dari pengguna, ada juga *unit testing* sebuah tahap dimana komponen individu perangkat lunak diuji, berikutnya *integration testing* untuk menemukan kesalahan dengan cara menggabungkan beberapa komponen, yang terakhir adalah *system testing* yang berfokus pada sistem yang sudah selesai, kebutuhan fungsionalitas maupun non-fungsional, serta sasaran lingkungan dari sistem tersebut.

2.1.3 Unified Modeling Language

Unified Modeling Language merupakan alat bantu untuk menggambarkan produk pada tahap desain untuk pengembangan perangkat lunak yang berorientasi objek, dan berasal dari penyatuan beberapa notasi berorientasi objek lainnya sehingga menjadi notasi standar. Dalam buku "*Object-Oriented Software Engineering Using UML, Patterns, and Java, Third Edition*" tujuan dari penggunaan *UML* adalah sebagai : "*The goal of UML is to provide a standard notation that can be used by all object-oriented methods and to select and integrate the best elements of precursor notations*." (Bruegge & Dutoit, 2014:28). *Unified Modeling Language* menampilkan beberapa diagram, diantaranya :

1. *Use Case Diagram*

   Use Case Diagram digunakan pada tahap *requirements elicitation* dan juga analisis untuk merepresentasikan fungsionalitas dari sebuah sistem. *Use case* berfokus pada *behavior* dari sistem, sebuah *use case* mendeskripsikan sebuah fungsi yang disediakan sistem. Menurut buku "*UML@Classroom: An introduction to object-oriented modeling*" terdapat beberapa komponen di dalam sebuah use case, yaitu :

   a. *Use Case*

      Use case mendeskripsikan fungsionalitas dari suatu sistem, *use case* adalah sebuah aksi atas apa yang dikerjakan oleh sistem. Pada umumnya use case berbentuk elips dan diberi nama di dalamnya.

   b. *Actor*

      Actor merupakan simbol untuk entitas eksternal, *actor* selalu berinteraksi dengan sistem melalui komponen use case. *Actor* bisa berupa manusia seperti murid atau dosen, namun juga *non-human* seperti perangkat lunak, perangkat keras, atau sistem.(Seidl, 2015:25)

c. *Relationship*

Relationship merupakan simbol yang menunjukan hubungan antara simbol-simbol pada diagram *use case*. Simbol ini berupa garis yang menghubungkan antara *actor* dengan *use case* atau dapat berupa hubungan antara *use case* satu dengan *use case* lainnya. Terdapat empat buah relasi dalam diagram *use case*, yaitu : *communication*, *include*, *extend*, dan *inheritance*. (Bruegge & Dutoit, 2014:44)

d. Communication

Communication atau *Associations* merupakan penghubung antara *actor* dengan *use case*, yang mengekspresikan bahwa *actor* tersebut berkomunikasi dengan sistem dan menggunakan fungsionalitas tertentu. Setiap *actor* harus berkomunikasi dengan minimal satu buah *use case*.

e. *Include*

Ketika mendeskripsikan sebuah sistem yang cukup kompleks, model *use case* bisa menjadi sedikit kompleks dan bisa terdapat *redundancy*. Kita mengurangi kompleksitas dari model tersebut dengan cara mengidentifikasi hal-hal yang mirip di beberapa *use case*. Simbol *include* adalah sebuah gambar tanda panah dengan garis yang putus-putus dan bertuliskan *include*, simbol ini diarahkan dari use case satu ke *use case* lainnya. *Use case* pertama disebut dengan *base use case* dan *use case* lainnya disebut dengan *included use case*. *Base use case* selalu membutuhkan *behavior* dari *included use case* agar bisa berfungsi, dan di sisi lain *included use case* bisa dieksekusi oleh dirinya sendiri.

f. *Extend*

Relasi *extend* adalah sebuah alternatif yang bertujuan mengurangi kompleksitas dalam model *use case*. Simbol *extend* mirip dengan simbol *include*, yaitu sebuah tanda panah dengan garis putus-putus namun bertuliskan *extend*. Perbedaannya adalah masing-masing *use case* dapat berfungsi mandiri, namun base use case

dapat menggunakan *behavior* yang terdapat pada *extending use case*. Sebuah *use case* bisa *extend use case* lainnya dengan cara menambahkan *Event*s.

g. *Inheritance*

Relasi *inheritance* merupakan mekanisme ketiga untuk mengurangi kompleksitas dari sebuah model, relasi *extend* dan *inheritance* merupakan dua hal yang berbeda. Penggunaan relasi *inheritance* digunakan hanya untuk *actor*, hubungan ini menunjukan *behavior* yang mirip yang dimiliki antar *actor* dalam hal perilaku pada sistem, sehingga dapat mengurangi jumlah interaksi *actor* yang berbeda untuk *use case* yang sama.

2. *Class Diagram*

Class Diagram digunakan untuk mendeskripsikan struktur dari sebuah sistem. Kelas-kelas adalah sebuah abstraksi yang menspesifikan struktur yang umum dan juga *behavior* dari serangkaian objek-objek. Objek adalah instansi dari kelas-kelas yang diciptakan, diubah, dan dihancurkan pada saat sistem dieksekusi. Sebuah objek memiliki keadaan dimana di dalamnya terdapat nilai dari atribut-atribut dan hubungannya dengan objek-objek lain.

3. *Sequence Diagram*

Sequence Diagram digunakan untuk memformalisasikan *behavior* dinamis dari sebuah sistem dan juga memvisualisasikan komunikasi diantara objek-objek. Mereka berguna untuk mengidentifikasi objek tambahan yang berpartisipasi dalam *use case*. *Sequence diagram* menggambarkan interaksi antara objek untuk memenuhi suatu tugas, mengendalikan urutan kronologis pesan dan konsep modularisasi yang memungkinkan untuk memodelkan interaksi yang sudah kompleks.

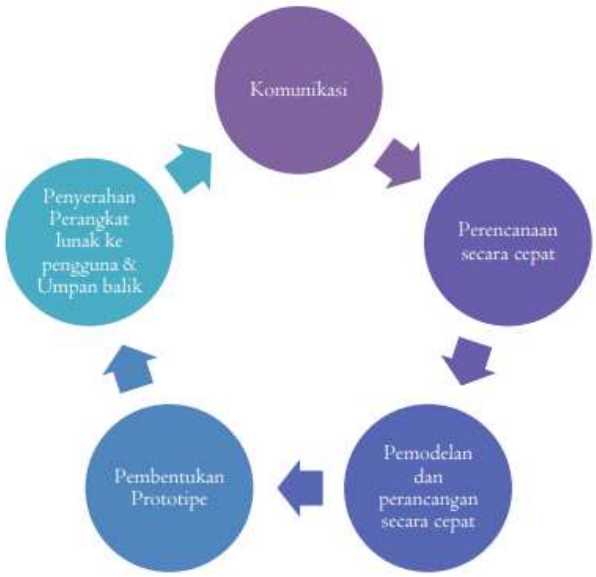
4. *Activity Diagram*

Sebuah *activity diagram* mendeskripsikan *behavior* sebuah sistem dalam hal aktivitas. Aktivitas adalah memodelkan elemen-elemen yang mereprentasikan eksekusi dari serangkaian operasi. Eksekusi dari sebuah aktivitas bisa dipicu dari aktivitas lain yang

sudah selesai, dari ketersediaan objek, atau dari kejadian dari luar. *Activity diagram* mirip dengan *diagram flowchart*, mereka bisa digunakan untuk merepresentasikan arus data.

2.1.4 Model Pengembangan Sistem Perangkat Lunak *Prototyping*

Model pengembangan *prototype* biasanya digunakan pada saat pengguna perangkat lunak mendefiniskan kebutuhannya secara umum dan tidak dapat mendefiniskan kebutuhan secara rinci misalnya, terkait fitur-fitur atau fungsi-fungsi yang ingin digunakan nantinya. Dengan kasus seperti itu, pendekatan *prototype* akan bermanfaat, paradigma ini juga seringkali membantu pihak pengembang perangkat lunak dan *client* untuk memahami kebutuhan perangkat lunak yang akan dikembangkan karena mereka dapat melihat dan melakukan pengerjaan dengan bagian dari sistem sejak awal proses pengembangan. (Setiyani, 2019:27).



Gambar 2.1
Paradigma pembuatan *prototype*

(Sumber : Setiyani, 2019:29)

Berdasarkan Gambar 2.1, pembuatan *prototype* dimulai dengan dilakukannya komunikasi antara pihak pengembang dengan *client*, kemudian menetapkan sasaran pengembangan secara keseluruhan dan mengidentifikasi spesifikasi kebutuhan. Lalu pembuatan *prototype* direncanakan dan didesain dengan cepat, kemudian rancangan tersebut dikonstruksi untuk membuat *prototype*. *Prototype* diserahkan kepada *client*, dan mereka melakukan evaluasi terhadap *prototype* tersebut, yang pada akhirnya memberikan umpan balik untuk perangkat lunak diperbaiki atau dikembangkan oleh pihak pengembang. Pengulangan yang terjadi pada saat *prototype* diperbaiki secara tidak langsung memenuhi kebutuhan *client*, dan pada saat yang sama memungkinkan pihak pengembang lebih memahami kebutuhan dari perangkat lunak yang sedang dikerjakan. Semakin banyak pengembangan maka perangkat lunak berevolusi untuk memenuhi kebutuhan pengguna.

2.1.5 Pengujian Perangkat Lunak

Pengujian perangkat lunak merupakan aktivitas yang bertujuan untuk mengevaluasi kemampuan dari sebuah program yang biasa dikenal dengan sebutan *Software Testing*. Tujuan dari pengujian adalah untuk menjamin kualitas perangkat lunak memiliki kualitas yang tinggi, dapat dilihat dari beberapa faktor seperti tingkat keamanan, *portability*, *reusability,* dan sebagainya.

Selain itu tujuan dari pengujian perangkat lunak juga untuk menemukan cacat yang biasa dilakukan oleh *programmer* saat mengembangkan perangkat lunak. Hal ini berpengaruh terhadap kepercayaan *client* sebagai pengguna jasa agar produk yang diberikan berkualitas. Metode pengujian perangkat lunak biasanya dibagi menjadi dua, yaitu *blackbox testing*, dan *whitebox testing.*

Blackbox testing merupakan cara pengujian yang hanya didasari oleh pengamatan terhadap data uji dan juga fungsionalitas dari perangkat lunak. Penguji diibaratkan hanya bisa melihat dari bagian luarnya saja, tanpa mengetahui isi dari *blackbox* tersebut. Pengujian *interface* dan fungsionalitas yang menjadi fokus dari *blackbox testing* ini.

Selain teknik pengujian *blackbox* dan *whitebox*, terdapat juga teknik pengujian bernama *rating scale*. Menurut *website* "*penelitianilmiah.com*", *rating scale* adalah pertanyaan survei tertutup yang digunakan untuk mewakili umpan balik responden, yang bertujuan untuk menunjukan tingkatan tertentu dalam instrumen penelitian yang dibuat (Rina Hayati, 2021).

2.2 *Game*

2.2.1 Definisi *Game*

Pada umumnya orang mengenal *game* sebagai permainan untuk menghibur atau untuk mengisi waktu luang agar tidak bosan, agar lebih menarik permainan tersebut diberi aturan-aturan untuk membatasi perilaku pemain dan menentukan permainan. Dalam *game* terdapat target-target yang ingin dicapai oleh pemainnya. Berikut adalah beberapa definisi tentang *game* menurut para ahli secara khusus :

Definisi *game* yang pertama menurut Elliot Avedon dan Brian Sutton-Smith yang disertakan dalam buku "*Educational Game Design Fundamentals*" adalah sebagai berikut : "*Games are an exercise of voluntary control systems, in which there is a contest between powers, confined by rules in order to produce a disequilibrial outcome*."(Kalmpourtzis, 2018:69).

Definisi *game* yang kedua menurut Greg Costikyan yang disertakan dalam buku "*Educational Game Design Fundamentals*" adalah sebagai berikut : "*A game is an interactive structure that requires players to struggle toward goals*."(Kalmpourtzis, 2018:70).

Definisi *game* yang ketiga menurut Jesse Schell yang disertakan dalam buku "*Educational Game Design Fundamentals*" adalah sebagai berikut : "*A game is a problem-solving activity, approached with a playful attitude*."(Kalmpourtzis, 2018:71).

Menurut beberapa pengertian tentang definisi *game* dari para ahli diatas, dapat disimpulkan bahwa *game* merupakan kegiatan untuk memecahkan masalah dengan cara

yang menyenangkan dan interaktif, yang di dalamnya terdapat aturan-aturan yang mengharuskan pemain berjuang untuk mencapai tujuannya.

2.2.2 Genre *Game*

Dalam video *game*, genre mengacu kepada jenis tantangan yang *game* tersebut tawarkan, terlepas dari konten yang di dalamnya. Contohnya *shooter game* tetaplah satu genre meskipun berlatar belakang di dunia fantasi, luar angkasa atau bergaya barat. Berikut ini adalah pengertian genre menurut buku "*Fundamentals of Game Design, Second Edition*" yaitu : " *A genre is a category of games characterized by a particular set of challenges, regardless of setting or game-world content*." (Adams, 2010:70)

Seiring dengan perkembangan *game*, saat ini sudah banyak genre *game* yang diciptakan dan dikembangkan. Di bawah ini merupakan beberapa jenis genre *game* klasik :

1. *Action*

   Hampir semua video *game* yang dibuat pada masa-masa awal merupakan genre *action*, seperti *Asteroids*, *Pac-Man*, *Space Invaders* menjadi representatif dari genre *action*. Genre ini membutuhkan koordinasi antara tangan dan mata yang baik, dan biasanya reaksi yang cepat pula. Pada jenis ini pemain tidak memiliki waktu yang cukup untuk berpikir membuat strategi atau merencanakannya, beberapa *game* juga membutuhkan kemampuan fisik seperti *aim* yang akurat, ritme, atau *timing* yang tepat, atau kemampuan untuk mengeksekusi gerakan *combo* sampai serangkaian perintah yang rumit. Yang paling popular dan sering dikenal oleh pemain dari genre *action* adalah *shooting games*, namun ada juga *platform games*, *fighting games*, *puzzle games* yang cepat, dan masih banyak lagi.

2. *Strategy*

   Game strategi menantang pemainnya untuk memenangkan *game* melalui perencanaan, strategi untuk mengurangi jumlah pasukan lawan biasanya diterapkan pada *game* jenis ini. Jadi kebanyakan *game* strategi adalah *game* perang antara negara, kerajaan, dan sebagainya. *Game* strategi biasanya bermain dengan dadu, kartu, dan permainan papan ini dikembangkan untuk *PC games* pada umumnya. *Game* strategi dibagi menjadi dua bagian, yaitu *classical turn-based games* dan *real-time strategy games.* Dalam sebuah *turn-based game*, pemain memikirkan langkah apa yang selanjutnya diambil, dengan mempertimbangkan keuntungan yang akan didapat. *RTS* atau *real time strategy game* dikembangkan setelah *turn-based game*, jenis ini menambahkan waktu sebagai elemen agar pemain merasa tertekan secara mental karena pemain tidak memiliki banyak kesempatan untuk memikirkan langkah yang akan diambil seperti pada bagian *turn-based*.

3. *Role Playing Game*

   Game RPG menyajikan pemain interaksi yang lebih luas dengan dunia *game* dibandingkan genre lain, dan juga pemain bisa memilih peran yang lebih banyak dari *game* biasanya. Kebanyakan *game RPG* juga memberikan sensasi yang menarik ketika pemain berkembang dari seseorang yang biasa saja menjadi pahlawan super dengan kekuatan yang mengagumkan, padahal biasanya genre lain langsung memberikan kekuatan super pada saat awal bermain. Jadi kesimpulan yang bisa diambil adalah genre *RPG* memberikan kepada pemain sensasi terjun ke dalam dunia yang kompleks dengan beragam pilihan cara bermain.

4. *Sports Game*

   Sports *game* memberikan tantangan khusus kepada desainer *game*, karena orang-orang yang suka melihat pertandingan olahraga atau olahragawan akan berekspektasi tinggi terhadap video *game* yang dibuat, jadi desainer *game* harus bisa memenuhi ekspektasi tersebut. Berbeda dengan kebanyakan *game* lain, yang memiliki latar

belakang yang asing bagi pemain, sport *game*s mensimulasikan sebuah dunia dimana pemain sangat mengenal dunia tersebut, bermain *game* namun seolah-olah mereka melakukan aktivitas olahraga pada dunia nyata. Memang tidak semua *sports game* sangat realistis, ada juga *game* fantasi tapi berbasis olahraga seperti *Super Shot Soccer*, dimana setiap negara dalam *game* tersebut memiliki kemampuan khusus dan biasanya *game* seperti ini ditujukan untuk anak-anak karena terlihat lebih menarik bagi mereka. Kebanyakan *sports game* berkonsentrasi pada mensimulasikan pertandingan yang aktual, namun banyak juga yang memasukan fitur manajemen seperti menjadi seorang manajer dari sebuah tim olahraga, atau mengatur karir dari seorang atlit.

5. *Vehicle Simulations*

   Game dengan mekanisme seperti ini membuat pemain merasakan sensasi dari mengendarai kendaraan, baik kendaraan darat maupun udara. Termasuk juga berlomba dengan pemain lain atau melawan komputer, selain itu melakukan eksplorasi dengan kendaraan, atau hanya sekedar merasakan sensasi berkendara. Yang diharapkan dari memainkan *game* seperti ini adalah pemain bisa benar-benar merasakan bagaimana mengendalikan sebuah kendaraan.

6. *Construction and Management Simulations*

   Construction and management simulations atau disingkat *CMS* menawarkan pemain kesempatan untuk membangun berbagai hal, seperti gedung, kota, sebagai contoh adalah *game* berjudul *SimCity* merupakan *game CMS* pertama yang sukses. Penekanan pada *game* CMS adalah mengenai proses, tujuan utama pemain bukanlah mengalahkan musuh seperti genre *action* atau lainnya, namun untuk membangun sesuatu di dalam konteks dari proses yang berkelanjutan. Tantangan besar dari *game CMS* yaitu adalah tantangan ekonomi dan urusan perkembangan.

7. *Adventure Game*

   Adventure game cukup berbeda dari kebanyakan *game* lainnya yang ada di pasaran, sebuah *adventure game* bukanlah mengenai kompetisi atau sebuah simulasi. Sebuah

adventure game tidak menawarkan proses untuk mengelola atau mengalahkan musuh dengan menyusun strategi dan taktik, tetapi sebuah *adventure game* merupakan cerita yang interaktif mengenai suatu karakter yang dikendalikan oleh pemain. Dia merupakan protagonis, seorang pahlawan dalam cerita tersebut. Memanipulasi atau memaksimalkan sumber daya bukanlah bagian pengalaman dari bermain *adventure game*, inilah yang membedakan *adventure game* dengan *role playing game*.

8. *Artificial Life*

   Genre ini memiliki lebih sedikit *game* dibanding kategori lainnya, di pasaran tidak banyak produk yang mirip, hal ini mendorong lebih banyak peluang untuk terciptanya sesuatu yang benar-benar baru. *The Sims* merupakan contoh dari *artificial life game*. Tipikal *artificial life game* berfokus pada menjaga, dan mengembangkan sebuah populasi makhluk hidup yang bisa dikelola, dan masing-masingnya bersifat unik.

9. *Online Gaming*

   Online gaming lebih mengacu kepada teknologi, dibandingkan kepada genre sebuah *game*, merupakan sebuah mekanisme untuk menghubungkan pemain secara bersama-sama. *Online gaming* telah berkembang dari bisnis hiburan kecil menjadi besar seperti saat ini. Menggunakan sebuah jaringan yang menghubungkan para pemainnya untuk bermain *game* menimbulkan kesempatan bagi pemain untuk berinteraksi satu sama lain, mendirikan komunitas, atau hanya sekedar bersosialisasi.

2.2.3 *Game* sebagai Edukasi

Game sebagai konten yang mengedukasi pemainnya atau memberikan pengetahuan merupakan media pembelajaran yang cukup efektif. Menurut buku "*Educational Game Design Fundamentals*" pembelajaran berarti : "*Learning is an ongoing process of acquiring existing knowledge that leads to internal change. Learning is continuous and involves every aspect of someone's life.*" (Kalmpourtzis, 2018:41). Orang belajar karena mereka ingin memenuhi kebutuhannya, dengan mengidentifikasi kebutuhan tersebut kita

bisa membangun *game* yang memiliki manfaat bagi pengguna, sekaligus memotivasi mereka.

Teknologi komunikasi yang berkembang saat ini memberikan nuansa yang berbeda dalam proses belajar seseorang, tempat belajar tidak hanya dalam kelas seperti dahulu namun bisa dilakukan dimana saja selama seseorang memiliki niat dan sumber daya untuk belajar.

2.3 Kimia

Menurut buku "*Chemistry, Twelfth Edition*" kimia adalah : "*Chemistry is the study of matter and the changes it undergoes*." (Chang & *Gold*sby, 2016:2). Kimia sering disebut sebagai pusat ilmu, karena pengetahuan dasar dari kimia penting bagi pelajar ilmu biologi, fisika, geologi, ekologi, dan mata pelajaran lainnya. Kimia juga merupakan pusat dari cara hidup kita sehari-hari, tanpa kimia kita masih hidup dengan kondisi primitif, tanpa kendaraan, energi listrik, komputer, dan masih banyak benda-benda keseharian lainnya. Dibandingkan dengan mata pelajaran lain, kimia umumnya dirasa lebih sulit, setidaknya pada tahap pengenalan.

Meskipun begitu, istilah kimia sudah banyak digunakan dalam kehidupan sehari-hari tanpa disadari misalnya elektronik, oksigen, karbondioksida, bahkan ketika kita memasak kita sedang mempelajari kimia. Dari pengalaman di dapur, kita mengetahui bahwa air dan minyak tidak bisa bercampur dan air mendidih yang ditinggal diatas kompor akan mengalami penguapan.

Materi pelajaran yang akan digunakan pada pembuatan *game* ini adalah tentang zat aditif dan zat adiktif. Zat aditif merupakan bahan yang ditambahkan ke dalam makanan atau minuman, yang bertujuan agar menambah cita rasa, penampilan, daya tahan dari makanan atau minuman. Penambahan zat aditif pada makanan juga dapat mempengaruhi nilai kandungan gizi yang terdapat dalam makanan dan minuman seperti vitamin, mineral, dan protein. Zat aditif dalam makanan dikelompokkan menjadi dua, yaitu zat aditif alami dan

buatan. Zat aditif alami umumnya tidak menimbulkan efek yang membahayakan kesehatan manusia karena biasanya diperoleh dari makhluk hidup. Sedangkan zat aditif buatan diperoleh melalui proses reaksi kimia. Sebagai contoh pewarna tekstil jika digunakan untuk pewarna makanan akan sangat berbahaya untuk kesehatan.

Zat adiktif adalah zat-zat yang jika dikonsumsi akan menyebabkan ketergantungan atau kecanduan. Zat adiktif yang umum dikonsumsi adalah kafein yang terdapat dalam kopi, orang yang terbiasa minum kopi secara rutin, kemudian suatu ketika tidak minum kopi akan muncul gejala-gejala ketergantungan kafein seperti merasa pusing. Zat adiktif dikelompokkan menjadi tiga, yaitu narkotika, psikotropika, serta zat psiko-aktif lainnya.

Materi pelajaran diperoleh dari buku pelajaran siswa yang berjudul "*ILMU PENGETAHUAN ALAM*" diharapkan mampu membantu siswa mempelajari pengertian tentang zat aditif, zat adiktif, serta manfaat dan kerugian yang diperoleh jika seseorang mengkonsumsi zat-zat tersebut dalam kehidupan sehari-hari. Setelah mengenal dampak baik dan buruk dari penggunaan zat aditif dan zat adiktif, siswa diharapkan mampu mengaplikasikan pengetahuan yang diperoleh dengan cara memilih makanan dan minuman yang sehat, serta menjauhi pengaruh buruk dari narkotika dan obat-obatan terlarang (Zubaidah, 2017).

2.4 RPG Maker MV

2.4.1 Definisi RPG Maker MV

Menurut buku "*Beginning RPG Maker MV*" definisi *RPG Maker MV* yaitu : "*RMMV is the latest version of the roleplaying game development engine published by Degica and developed by Kadokawa Games.*" (Perez, 2016:xxiii). *RPG Maker MV* dirilis secara internasional pada tanggal 23 Oktober 2015, *RPG Maker* merupakan sebuah *game development engine* untuk membuat *game* RPG 2 dimensi, proses pembuatan dibantu oleh alat-alat yang telah disediakan dalam program. *RPG Maker* awalnya dirilis di Jepang, kemudian Asia Timur, Amerika Utara, Eropa, dan Australia. *RPG Maker* merupakan perangkat lunak yang popular digunakan untuk mengembangkan *game*.

2.4.2 Fitur-fitur pada *RPG Maker MV*

Berikut ini adalah beberapa komponen dalam perangkat lunak *RPG Maker* :

1. *Database*

   Database digunakan untuk tempat penyimpanan bermacam-macam komponen pada *game*. *Database* dibedakan berdasarkan bentuk *tab*, setiap *tab* memiliki sebuah kategori. *Tab* yang terdapat pada *Database* antara lain :

   a. *Actor*

      Tab bagian actor berisi data untuk membuat karakter.

   b. *Classes*

      Tab bagian *classes* mengatur kelas-kelas yang akan digunakan pada karakter.

   c. *Skill, Animation, State*

      Tab bagian ini mengatur kemampuan yang dimiliki oleh karakter, serta animasi dan efek dari kemampuan tersebut.

   d. *Item, Weapon, Armor*

      Tab bagian ini mengatur barang, senjata, dan perlengkapan karakter.

e. *System and Term*

Tab bagian ini berisi pengaturan standar, seperti efek suara dan menu dalam *game*.

f. *Common Event and Tileset*

Tab bagian ini mengatur kejadian-kejadian yang terjadi dalam *game* dan aksesoris untuk mendukung pembuatan *game*.

2. *Map editor*

Map editor merupakan tempat untuk membuat peta untuk dijelajahi oleh karakter. Peta dapat dibuat dengan memilih *tileset* yang berada di sebelah kiri *map editor*. Pada *map editor*, pihak pengembang dapat menentukan daerah mana saja yang dapat dijelajahi.

3. *Event*

Event memiliki peran yang signifikan untuk memberikan sebuah interaksi antara karakter dengan NPC, kendaraan, perpindahan *map*, dan sebagainya. Selain itu *event* juga mengatur alur adegan yang ada dalam *game* seperti perubahan dialog antara aktor dengan beberapa NPC.

4. *Switch and Variables*

Switch and Variables dalam *RPG Maker MV* mirip seperti kotak di kehidupan nyata. Kotak-kotak ini menyimpan informasi yang akan kita gunakan nantinya. Berikut ini adalah beberapa penggunaan dari *Switch* dan *Variables* :

a. *Variables* memiliki kemampuan untuk mengetahui berapa banyak peti harta karun yang sudah dibuka dalam *game.*

b. *Switch* memiliki kemampuan untuk menentukan apakah sebuah *boss* sudah dikalahkan atau belum.

c. *Switch* bisa membatasi pemain menjelajahi area tertentu dalam *map*, misalnya jika pemain tersebut tidak memiliki kunci atau benda lainnya.

d. *Variables* dapat digunakan untuk mengganti hadiah dari *quest*, misalnya pertama kali kita menyelesaikan sebuah *quest* dari membunuh musuh kita

e. mendapatkan 300 *gold*, dan mendapat hadiah yang dikurangi jika kita hanya berhasil mengusir musuh, atau jika gagal.

f. *Variables* dapat digunakan untuk meng*hit*ung musuh yang tersisa dalam layar.

g. *Variables* dapat meng*hit*ung jumlah langkah yang pemain sudah jalani, ini bisa diaplikasikan untuk menentukan *game over* yang tidak biasa, ketika pemain mencapai jumlah langkah tertentu, maka *game* berakhir.

Dengan demikian, penggunaan *Switch* dan *Variables* memang bisa dikatakan hampir mirip, namun memiliki perbedaan. Untuk *Switch*, dapat dianalogikan seperti saklar lampu, memiliki kondisi menyala dan kondisi mati, artinya bisa mengatur kondisi tambahan yang akan terpicu hanya jika saklar dalam kondisi menyala atau kondisi mati. Sedangkan *variables* dapat digunakan untuk menyimpan angka-angka. Sebaiknya *switch* digunakan ketika ingin membuat tombol, atau tuas, atau benda-benda yang hanya memiliki dua buah kondisi. Penggunaan *variables* paling baik adalah ketika situasi tidak dapat dipenuhi hanya dengan mengandalkan dua buah kondisi sederhana, kondisi menyala dan mati seperti *switch*. Contohnya ketika ingin meng*hit*ung jumlah *monster* yang sudah kita bunuh, dan ingin menyimpan jumlah tersebut pada sebuah *variable*.

5. *JavaScript*

RPG Maker MV menggunakan *JavaScript*, dimana *JavaScript* merupakan *scripting language*, yang biasa dijalankan pada *browser*, orang-orang lebih mengenalnya dengan *client side programming*. *Client* merupakan sebuah *browser* seperti *Internet Explorer, Firefox, Safari*, dan sebagainya. *JavaScript* didesain untuk menambah interaktifitas dari sebuah aplikasi *web*. Kode *JavaScript* biasanya disisipkan diantara kode-kode *HTML*. *JavaScript* menggunakan *text editor* sebagai tempat untuk menulis kode-kodenya, seperti *Notepad, Atom, Visual Studio Code*, dan sebagainya. Sintaksis dari *JavaScript*

serupa dengan bahasa pemrograman *C* atau *java*. *JavaScript* bersifat *case sensitive*, ini berarti adanya perbedaan jika menggunakan huruf besar dan huruf kecil.

6. *Plugin*

   Plugin merupakan kode tambahan untuk menambah fungsi suatu program. *RPG Maker MV* memiliki fitur *plugin* yang dapat digunakan oleh pengembang untuk menambah berbagai hal dalam *game* seperti membuat tampilan antar muka menjadi lebih halus, menampilkan HP musuh, dan berbagai hal lain. Pengguna dapat memilih atau menghapus *plugin* yang ingin digunakan menggunakan menu yang disediakan pada *RPG Maker MV*.

2.5 Perbandingan dengan Penelitian Terdahulu

Tabel 2.1 merupakan tabel pembanding yang membedakan antara penelitian yang sudah pernah ada, kemudian dibandingkan dengan penelitian ini. Adapun perbedaan dari penelitian terdahulu, yaitu mengenai tujuan penelitian, karena data yang dibandingkan diperoleh dari penelitian pembuatan *game* menggunakan perangkat lunak *RPG Maker.*

Tabel 2.1
Tabel Pembanding Tujuan Penelitian

Judul Penelitian	Tujuan Penelitian	Kegunaan Hasil
PENGEMBANGAN *GAME* EDUKASI KIMIA BERBASIS ROLE PLAYING *GAME* (RPG) PADA MATERI STRUKTUR ATOM SEBAGAI MEDIA PEMBELAJARAN MANDIRI UNTUK SISWA KELAS X SMA DI KABUPATEN PURWOREJO (Sari, Saputro and Hastuti, 2014).	Menjelaskan materi pelajaran kimia mengenai struktur atom untuk siswa kelas X di SMA Negeri 2 Purworejo dan SMA Negeri 8 Purworejo. Menghasilkan media pembelajaran mandiri bagi para siswa.	*Game* edukasi berbasis Role Playing *Game* (RPG) pada materi struktur atom untuk siswa kelas X SMA dapat dipakai sebagai media pembelajaran mandiri bagi para siswa.
PENGEMBANGAN MEDIA *GAME* EDUKASI KIMIA MENGGUNAKAN SCRATCH PADA ANAK	Menjelaskan materi kimia tentang aliran energi di sekitar kita. Penelitian ditujukan untuk	Diharapkan setelah bermain *game* edukasi kimia yang dikembangkan, siswa

TAHAPAN OPERASIONAL FORMAL (Hariyati, 2017)	anak berusia 12 sampai 15 tahun di kota Jambi.	menjadi lebih paham mengenai materi aliran energi yang disajikan.
PENGEMBANGAN *GAME* EDUKASI KIMIA ADVENTURE OF LITTLE ALCHEMIST	Membuat aplikasi *game* edukasi kimia untuk siswa SMP kelas 8 menggunakan *RPG Maker MV*. Menjelaskan materi kimia tentang zat aditif dan zat adiktif.	Dapat menambah wawasan siswa dengan cara yang menyenangkan serta materi pelajaran dapat tersampaikan dengan baik kepada siswa.

BAB III

ANALISIS DAN PERANCANGAN PERANGKAT LUNAK

3.1 Gambaran Umum Perangkat Lunak

Dalam penelitian ini perangkat lunak yang akan dirancang akan berbentuk sebuah *game* berjenis *role playing game*, dengan tampilan grafis dua dimensi dan sudut pandang yang digunakan adalah orang ketiga. Pemain berperan sebagai tokoh pahlawan dalam cerita, dimana awal mulanya pemain merupakan seorang anak kecil berusia 12 tahun bernama Vincent yang hidup di tahun 2020. *Game* dimulai pada sebuah ruang kelas, tempat bagi pemain untuk mempelajari zat aditif dan zat adiktif. Pemain harus memenuhi seluruh kondisi belajar seperti membaca seluruh buku yang berisi pengetahuan, jika ingin melanjutkan ke bagian selanjutnya. Proses belajar ini juga berguna bagi pemain untuk memenuhi uji kemampuan yang diberikan oleh NPC sebagai bagian dari proses bermain *game*. Setelah jam pelajaran sekolah berakhir, pemain harus menemui NPC guru yang terletak di depan papan tulis sekolah untuk mendapatkan ijin keluar kelas. Skenario berlanjut sampai ketika Vincent sampai di rumah, Vincent masuk ke dalam kamar kemudian tertidur, namun dia bermimpi kembali ke masa lalu dan bertemu seorang Dewi bernama Ramayan. Sang Dewi pun menjelaskan bahwa dia yang telah memanggil Vincent ke masa lampau dan memberi tugas untuk mengalahkan musuh serta membantu para penduduk desa yang kesulitan. Namun kekuatan sang Dewi sudah habis terpakai untuk memanggil Vincent ke masa lampau, sehingga agar bisa mengembalikan Vincent ke tahun 2020 dia memerlukan kekuatan dari empat batu elemen yang tersebar di seluruh penjuru dunia ini. Terdapat tiga batu yang sudah diketahui keberadaannya dan mereka mewakili elemen dari masing-masing naga yang harus dikalahkan, yaitu naga api Fira, naga angin Zephyr, dan naga air Aqua. Untuk menambah unsur edukasi dalam *game*, pemain juga akan diberikan uji kemampuan yang berhubungan dengan edukasi zat aditif dan zat adiktif yang ada dalam *game* sebagai ujian dari NPC. Syarat dari uji kemampuan dinyatakan berhasil adalah ketika pemain berhasil

membawa benda-benda yang diminta oleh NPC, seperti contoh dari benda stimulan yaitu kafein. Jika pemain berhasil menyelesaikan uji kemampuan tersebut maka akan ada keuntungan yang didapat seperti penambahan *skill* khusus karakter untuk menyelesaikan *game* lebih mudah. Setelah keempat batu elemen didapatkan, pemain bisa kembali ke dunia dia berasal.

Pemain akan diarahkan untuk mengembangkan kemampuan bertahan hidup sekaligus mendapatkan perlengkapan dengan cara mengalahkan *monster* yang ada di luar kota dan dengan cara menjalankan *quest* yang diberikan penduduk desa. *Quest* yang diberikan dalam *game* adalah pemain harus membuat barang yang termasuk ke dalam zat aditif atau zat adiktif yang berguna bagi warga desa, seperti gula aren yang merupakan pemanis alami. Pemain akan diberikan *quest* jika berinteraksi dengan NPC kakek tua, kemudian akan dijelaskan mengenai tugas yang harus diselesaikan pemain. Setelah pemain berhasil menyelesaikan tugas, NPC akan memberikan petunjuk mengenai cara membuat gula aren melalui penayangan video, setelahnya pemain akan diberikan tugas yang baru sampai berhasil membuat gula aren. Tidak ada fitur *stage* dalam *game* ini, namun jika pemain ingin melanjutkan narasi maka pemain harus menyelesaikan *main quest* yang diberikan oleh NPC kakek tua dan juga mengalahkan naga di masing-masing desa. *Game* ini dirancang agar pemain bisa belajar sambil bermain, konten-konten yang dibuat seperti bagian awal *game* ketika pemain harus belajar terlebih dahulu, atau ketika pemain mendapatkan uji kemampuan yang diberikan NPC akan memperkaya unsur belajar sambil bermain, yang diharapkan mampu menambah wawasan pemain dengan cara yang menyenangkan.

Pada awal *game* sang Dewi Ramayan akan menjelaskan perbedaan kemampuan yang dimiliki setiap elemen kepada pemain. Pada akhir pembicaraan, Dewi Ramayan akan bertanya kepada pemain untuk memilih satu diantara tiga elemen dasar yang tersedia yaitu elemen api, elemen air, dan elemen tanah. Masing-masing elemen tersebut memiliki musuh alaminya dan dapat dengan mudah saling mengalahkan satu dengan yang lainnya,

contohnya elemen api mendapat tambahan serangan terhadap elemen tanah, elemen tanah akan lebih mudah mengalahkan elemen air, dan elemen air dapat lebih mudah mengalahkan elemen api. Berikut merupakan daftar kendali untuk karakter pada cerita :

Tabel 3.1
Daftar Kendali Karakter

Kendali	Keterangan
Arrow up, Numpad 8	Digunakan untuk menggerakkan karakter maju sesuai dengan arah pandangan.
Arrow down, Numpad 2	Digunakan untuk menggerakkan karakter maju berlawanan dengan arah pandangan.
Arrow left, Numpad 4	Digunakan untuk menggerakkan karakter maju ke arah kiri dari arah pandangan.
Arrow right, Numpad 6	Digunakan untuk menggerakkan karakter maju ke arah kanan dari arah pandangan.
Z, *Enter*, *Space*	Digunakan untuk memicu tombol aksi.
X, *Esc*, *Numpad* 0	Digunakan untuk memicu tombol batal.
Shift	Digunakan untuk membuat karakter berlari.
Q, *Page up*	Digunakan untuk berpindah ke halaman sebelumnya.
W, Page down	Digunakan untuk berpindah ke halaman selanjutnya.

Terdapat beberapa pekerjaan berbeda dalam *game* ini yang sesuai dengan elemen yang dipilih oleh pemain, yaitu *blacksmith*, *fisherman*, dan *farmer*. Berikut ini adalah penjelasan mengenai pekerjaan :

1. *Blacksmith* merupakan kegiatan pada saat pemain melakukan *crafting* senjata atau perlengkapan, pekerjaan ini khusus dimiliki oleh elemen api. Setelah sukses membuat barang maka pemain akan mendapatkan poin yang nantinya dapat ditukar menjadi status tambahan bagi pemain. NPC Master Blacksmith dapat d*item*ukan hampir di

2. setiap kota dan berguna ketika pemain ingin membeli resep untuk *crafting* atau ketika pemain ingin menukarkan poin *blacksmith* yang dimiliki.
3. *Fisherman* merupakan pekerjaan untuk elemen air, akan disediakan beberapa tempat dalam *game* yang bisa pemain gunakan untuk memancing ikan, setelah mendapatkan ikan, pemain dapat menggunakannya untuk berbagai manfaat seperti menyembuhkan *hit point*, dapat dijual untuk mendapatkan uang, atau ditukar sebagai hadiah kepada NPC.
4. *Farmer* merupakan pekerjaan untuk elemen tanah, akan disediakan tempat dalam *game* yang bisa pemain gunakan untuk bertani dan mendapatkan berbagai macam hasil bumi, yang nantinya dapat dimanfaatkan sebagai makanan untuk menyembuhkan *hit point* karakter, dijual untuk mendapatkan tambahan uang, atau ditukar sebagai hadiah kepada NPC.

Ketika pemain memiliki poin yang cukup untuk ditukar kepada NPC masing-masing elemen, maka pemain dapat memiliki beberapa pilihan status yang ingin dia tingkatkan seperti :

1. *Hit point* yaitu nilai yang melekat pada karakter sebagai nyawa karakter tersebut, ketika *hit point* berkurang sampai nilai nol maka karakter tersebut akan terkena status *knockout*, dan tidak dapat berpartisipasi dalam pertarungan untuk sementara. Jika seluruh karakter yang terdapat dalam tim *knockout*, pemain akan mengulang kembali ke data *save* paling akhir.
2. Maximum *hit point* merupakan nilai maksimal dari *hit point* yang dimiliki karakter dalam *game*.
3. *Magic point* penggunaan *magic point* mirip seperti *hit point*, namun *magic point* berguna untuk menunjukan jumlah mana yang dimiliki karakter pada suatu waktu. *Magic point* dibutuhkan jika karakter ingin menggunakan *skill*.

4. Maximum *magic point* yaitu jumlah maksimal *magic point* yang dimiliki oleh karakter di dalam *game*.
5. *Technique point* pengunaan dari *technique point* mirip seperti *magic point*, jika pemain ingin mengeluarkan *skill* tertentu maka pemain harus memiliki *technique point* dalam jumlah tertentu, *technique point* didapat saat karakter diserang oleh musuh.
6. *Attack* merupakan nilai dari serangan terhadap musuh yang dapat dikeluarkan oleh karakter dalam *game*.
7. *Defense* merupakan nilai pertahanan yang digunakan untuk mengurangi serangan dari musuh.
8. *Magic Attack* merupakan nilai dari serangan *magic* yang dimiliki karakter, *magic Attack* lebih banyak berpengaruh terhadap serangan dari *skill* yang dikeluarkan oleh karakter.
9. *Magic Defense* merupakan nilai pertahanan dari serangan *magic*, digunakan untuk mengurangi serangan *magic* dari musuh.
10. *Agility* merupakan status yang menentukan urutan serangan masing-masing karakter, semakin besar *Agility* yang dimiliki maka semakin cepat karakter melakukan serangan terhadap musuh.
11. *Luck* mempengaruhi kemungkinan status berhasil diaplikasikan atau dihindarkan kepada target, juga mempengaruhi kesempatan untuk pemain mendapat serangan *critical*.

Di bawah ini merupakan daftar dari status yang dimiliki oleh musuh dalam *game*, serta *Element Rate* yang dimiliki. *Element Rate* merupakan penambahan damage yang diterima musuh sesuai elemen yang dimiliki, semakin tinggi jumlahnya maka semakin tinggi juga damage yang diterima. Parameter yang digunakan untuk musuh juga sama seperti parameter yang terdapat pada karakter dalam *game*, yaitu :

Tabel 3.2
Status Musuh dalam *Game*

No	Nama	Parameter		*Element Rate*
1	*Bat*	Max HP : 200 Max MP : 0 *Attack* : 30 *Defense* : 30	M.*Attack* : 30 M.*Defense* : 30 *Agility* : 30 *Luck* : 30	*Fire* * 120%
2	*Slime*	Max HP : 250 Max MP : 0 *Attack* : 30 *Defense* : 30	M.*Attack* : 30 M.*Defense* : 30 *Agility* : 30 *Luck* : 30	*Fire* * 120%
3	*Orc*	Max HP : 400 Max MP : 0 *Attack* : 40 *Defense* : 30	M.*Attack* : 30 M.*Defense* : 30 *Agility* : 30 *Luck* : 30	-
4	*Minotaur*	Max HP : 1000 Max MP : 0 *Attack* : 40 *Defense* : 30	M.*Attack* : 30 M.*Defense* : 30 *Agility* : 30 *Luck* : 30	-
5	*Desert Slime*	Max HP : 500 Max MP : 0 *Attack* : 40 *Defense* : 40	M.*Attack* : 40 M.*Defense* : 40 *Agility* : 40 *Luck* : 40	*Water* * 120%
6	*Desert Rat*	Max HP : 500 Max MP : 0 *Attack* : 40 *Defense* : 40	M.*Attack* : 40 M.*Defense* : 40 *Agility* : 40 *Luck* : 40	*Water* * 120%
7	*Ice Slime*	Max HP : 1500 Max MP : 300 *Attack* : 50 *Defense* : 50	M.*Attack* : 50 M.*Defense* : 50 *Agility* : 50 *Luck* : 50	*Earth* * 120%
8	*Water Spirit*	Max HP : 1500 Max MP : 2000 *Attack* : 50 *Defense* : 50	M.*Attack* : 70 M.*Defense* : 50 *Agility* : 50 *Luck* : 50	*Earth* * 120%
9	*Ogre*	Max HP : 3000 Max MP : 2000 *Attack* : 50 *Defense* : 50	M.*Attack* : 70 M.*Defense* : 50 *Agility* : 50 *Luck* : 50	*Earth* * 120%
10	*Zephyr*	Max HP : 800 Max MP : 0 *Attack* : 30	M.*Attack* : 40 M.*Defense* : 40 *Agility* : 40	*Fire* * 120%

		Defense : 40	*Luck* : 40	
11	*Fira*	Max HP : 2000 Max MP : 1000 *Attack* : 50 *Defense* : 50	M.*Attack* : 50 M.*Defense* : 50 *Agility* : 50 *Luck* : 50	*Water* * 120% *Fire* * 70% *Earth* * 70%
12	*Aqua*	Max HP : 5000 Max MP : 1500 *Attack* : 70 *Defense* : 70	M.*Attack* : 70 M.*Defense* : 70 *Agility* : 70 *Luck* : 70	*Water* * 70% *Fire* * 70% *Earth* * 120%
13	*Fire Mage*	Max HP : 2500 Max MP : 500 *Attack* : 50 *Defense* : 50	M.*Attack* : 50 M.*Defense* : 50 *Agility* : 50 *Luck* : 50	*Water* * 120% *Fire* * 70% *Earth* * 70%

Di bawah ini merupakan tabel untuk daftar *item-item* yang terdapat dalam *game*. Terdapat deskripsi serta harga dari *item* tersebut seperti :

Tabel 3.3
Item dalam *game*

No	Nama *Item*	Deskripsi	Harga
1	*Potion*	Dapat digunakan untuk menyembuhkan 500 HP	100
2	*Mana Potion*	Dapat digunakan untuk menyembuhkan 50 MP	100
3	Umpan ikan	Umpan digunakan untuk memancing	50
4	Joran	alat untuk memancing ikan	0
5	*Great Sword*	Pedang yang lebih kuat daripada pedang biasa	1000
6	*Great Axe*	Kapak yang kuat	1000
7	*Staff*	Tongkat sihir yang umum digunakan	1000
8	*Great Bow*	Busur panah yang sering digunakan pemburu	1000
9	*BroadSword*	Pedang yang sudah menghilangkan banyak nyawa *monster*	1000

No	Nama *Item*	Deskripsi	Harga
10	*Claymore*	Sebuah pedang kuat hasil *crafting*	1000

Di bawah ini merupakan tabel untuk daftar *item-item* zat aditif maupun zat adiktif yang digunakan sebagai bahan pembelajaran dalam *game*. Terdapat deskripsi serta harga dari *item* tersebut seperti :

Tabel 3.4
Item zat aditif dan zat adiktif

No	Nama *Item*	Deskripsi	Harga
1	Gula	Salah satu bahan pemanis makanan	0
2	Kopi	Kopi mengandung kafein, orang-orang biasa meminumnya untuk menghilangkan rasa kantuk	0
3	Alkohol	Hasil fermentasi dari glukosa dengan ragi	0
4	Garam	Salah satu bahan penyedap makanan	0

3.2 Spesifikasi Kebutuhan Perangkat Lunak

Spesifikasi kebutuhan perangkat lunak untuk *game* ini dibagi menjadi dua bagian, yaitu kebutuhan fungsional dan non fungsional. Pada kebutuhan fungsional terdiri dari daftar kebutuhan yang berisi beberapa proses yang perlu disediakan oleh sistem. Sedangkan kebutuhan non fungsional meliputi perilaku sistem yang berhubungan dengan kinerja, operasional, platform sistem, hukum.

3.2.1 Kebutuhan Fungsional

Berikut merupakan kebutuhan fungsional dari *game* yang dirancang :

1. *Game* ini hanya dapat dimainkan secara *offline* dan *single player*.
2. *Game* ini hanya dapat berjalan pada PC yang memiliki sistem operasi *Windows* versi 7/8/8.1/10.
3. Pemain dapat memilih tiga jenis elemen yaitu elemen api, elemen air, dan elemen tanah.
4. Karakter yang terdapat dalam *game* memiliki *skill*.
5. Pemain memiliki *inventory* yang berguna sebagai tempat menyimpan barang.
6. Karakter dapat menggunakan *equipment*.
7. Pemain dapat menyelesaikan *quest*.

3.2.2 Kebutuhan Non Fungsional

Spesifikasi kebutuhan non fungsional perangkat lunak terdiri dari kebutuhan perangkat keras, menunjukan perangkat keras minimum yang dibutuhkan agar perangkat lunak dapat berjalan dan juga kebutuhan dari perangkat lunak yang dibutuhkan oleh *game* agar dapat berjalan. Berikut merupakan spesifikasi kebutuhan non fungsional perangkat lunak :

1. Kebutuhan perangkat keras :

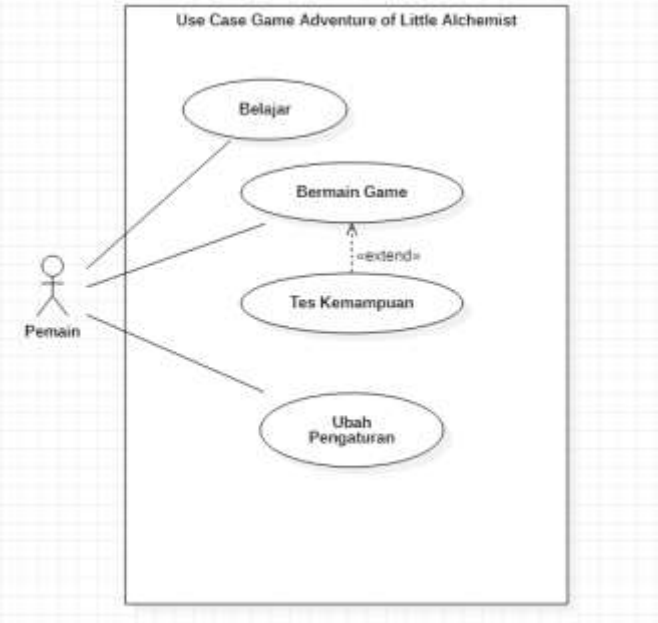
Tabel 3.5
Kebutuhan non fungsional perangkat keras

Perangkat Keras	Keterangan Spesifikasi
Processor	Intel Core2 Duo
Storage	Membutuhkan 1 GB tempat penyimpanan
Memory	4 *GB RAM*
Graphics Card	AMD Radeon HD 6250
Resolusi	1280 x 768

Berdasarkan tabel 3.5, merupakan *minimum requiment* untuk pengembangan perangkat lunak game "***PENGEMBANGAN GAME EDUKASI KIMIA ADVENTURE OF LITTLE ALCHEMIST***".

2. Kebutuhan perangkat lunak :
    a. *Operating system* : *Windows* 7
    b. Perangkat lunak untuk pengembang : *RPG Maker MV Version* 1.6.1

3.3 *Use case* Diagram



Gambar 3.1
Use Case Game RPG

Use case diagram diatas merupakan gambaran mengenai interaksi antara aktor yang merupakan pemain, dengan sistem *game*. Pada *use case* Belajar, pemain mendapatkan pengetahuan mengenai zat aditif dan zat adiktif yang diperoleh dari interaksi aktor dalam *game*. Sedangkan use case Bermain *Game* adalah ketika pemain melakukan aktivitas-aktivitas yang berhubungan dengan permainan, seperti membunuh *monster*, membeli barang, dan sebagainya. *Use case* Tes Kemampuan merupakan bagian dari bermain *game*, dimana nantinya pemain akan diberikan ujian dari NPC yang berhubungan dengan materi pembelajaran. *Use case* Ubah Pengaturan berguna jika pemain ingin merubah pengaturan yang berhubungan dengan *game*, contohnya untuk mengatur volume musik.

3.4 Skenario Diagram

3.4.1 Skenario *Use case* Bermain *Game*

Skenario Bermain *game* utama

Kondisi : Pemain belum memiliki data *save game* sebelumnya.

Deskripsi : Pemain ingin memulai *game*

Tabel 3.6
Skenario Bermain *Game* utama

Aksi Actor	Sistem	Database
1. Menekan tombol *New Game*.		
	2. Melakukan load scene dari *game*	
3. Mulai bermain.		
	4. Menampilkan dialog pembuka	
5. Menekan tombol *arrow up* atau *numpad* 8 pada *keyboard*.		
	6. Karakter menjalankan animasi berjalan ke depan sesuai arah pandangan.	
7. Menekan tombol Z untuk berinteraksi dengan NPC.		
	8. Menampilkan dialog percakapan	
	9. Menampilkan dialog *quest*	
10. Menekan tombol Z untuk melakukan konfirmasi		
	11. Mengganti kelas elemen sesuai pilihan	
		12. Mengambil data aktor dan kelas sesuai pilihan
	13. Mengganti aktor dan kelas sesuai *database*	
	14. Melakukan perubahan *map*	
15. Menekan tombol Z untuk berinteraksi dengan NPC		
	16. Melakukan progres *quest*	
	17. Memeriksa status *quest* dan status selesai	
	18. Memberikan *EXP* kepada karakter sesuai ketentuan	
	19. Memeriksa *EXP* yang dimiliki karakter dan melebihi *EXP Curve*	

	20. Menambah *level* karakter sebanyak satu	
		21. Memperbarui data *level* karakter
	22. Menampilkan dialog *LevelUp* karakter	
	23. Melakukan update *quest*	
	24. Menampilkan data *quest* baru	
25. Menekan tombol Z untuk berinteraksi dengan *loot chest*		
	26. Membuat navigasi menjadi tidak aktif	
		27. Membaca data yang berada dalam *loot chest*
	28. Menarik data dalam *loot chest*	
	29. Mengubah *item* pada *inventory* sesuai data *loot chest*	
		30. Memperbarui data *inventory*
	31. Menampilkan animasi *chest* yang sudah di *loot*	
	32. Menampilkan dialog *chest* berhasil di *loot*	
33. Menekan tombol X		
34. Memilih menu barang		
	35. Membuat navigasi menjadi tidak aktif	
		36. Membaca daftar barang yang dimiliki
	37. Menampilkan data barang yang dimiliki	
38. Memilih menu *skill*		
	39. Membuat navigasi menjadi tidak aktif	
		40. Membaca daftar *skill* yang dimiliki
	41. Menampilkan daftar *skill* yang dimiliki	
42. Memilih menu *equip*		
	43. Membuat navigasi menjadi tidak aktif	
		44. Membaca daftar *equip* yang dimiliki
	45. Menampilkan daftar *equip* yang dimiliki	
46. Memilih menu status		
	47. Membuat navigasi menjadi tidak aktif	

		48. Membaca daftar status karakter
	49. Menampilkan status dari karakter	
50. Memilih menu *formation*		
51. Memilih dua karakter		
	52. Menukar posisi antara dua karakter yang dipilih	
53. Menekan tombol Z untuk berinteraksi dengan NPC.		
	54. Menampilkan dialog percakapan	
	55. Menampilkan dialog *quest*	
56. Menekan tombol Z untuk melakukan konfirmasi		
	57. Melakukan progres *quest*	
	58. Memeriksa status *quest* dan status selesai	
	59. Memberikan *EXP* kepada karakter sesuai ketentuan	
	60. Memeriksa *EXP* yang dimiliki karakter dan melebihi *EXP Curve*	
	61. Menambah *level* karakter sebanyak satu	
		62. Memperbarui data *level* karakter
	63. Menampilkan dialog *Level*Up karakter	
	64. Melakukan update *quest*	
	65. Menampilkan dialog *end game*	
66. Menekan tombol Z untuk melakukan konfirmasi		
	67. Melakukan perubahan *map*	
	68. Menampilkan dialog *easter egg*	
	69. Meng*hit*ung jumlah karakter mati dalam *game*	
	70. Mengurangi jumlah *IQ* dengan jumlah mati karakter dalam *game*	
	71. Menampilkan dialog penutup	
72. Memilih menu *game* end		
73. Memilih kembali ke layar judul		
	74. Menampilkan menu utama	
75. Melihat tampilan menu utama		

Skenario Bermain *game* alternatif 1

Kondisi : Pemain sudah memiliki data *game* sebelumnya

Deskripsi : Pemain ingin melanjutkan *game*

Tabel 3.7
Skenario Bermain *Game* Alternatif 1

Aksi Actor	Sistem	Database
1. Menekan tombol *Continue*.		
	2. Membuka menu pemilihan data	
		3. Membaca data *game* yang dimiliki
	4. Menampilkan data *game*	
5. Memilih data yang akan dimainkan		
6. Mulai bermain.		
Nomor 7-75 sama seperti skenario *Bermain game* utama.		

Skenario Bermain *game* alternatif 2

Kondisi : Pemain sedang membuka menu *game*

Deskripsi : Pemain ingin melakukan *save game*

Tabel 3.8
Skenario Bermain *Game* Alternatif 2

Aksi Actor	Sistem	Database
Nomor 1-33 sama seperti skenario Bermain *game* utama		
34. Memilih menu *save*		
	35. Menampilkan menu untuk *save game*	
		36. Membaca data penyimpanan karakter
37. Memilih *slot* untuk *save game*		
	38. Menampilkan *slot* sesuai pilihan pemain	
		39. Menyimpan data permainan pada *slot* yang dipilih

Skenario Bermain *game* alternatif 3

Kondisi : Pemain sedang membuka menu *game*

Deskripsi : Pemain ingin membatalkan *save*

Tabel 3.9
Skenario Bermain *Game* Alternatif 3

Aksi Actor	Sistem	Database
Nomor 1-33 sama seperti skenario Bermain *game* utama		
34.Memilih menu *save*		
	35.Menampilkan menu untuk *save game*	
		36.Membaca data penyimpanan karakter
37.Menekan tombol X		
	38.Kembali ke menu *game*	

Skenario Bermain *game* alternatif 5

Kondisi : Pemain sedang berada dalam menu pertarungan

Deskripsi : Proses pemain sedang melakukan pertarungan dengan musuh

Tabel 3.10
Skenario Bermain *Game* Alternatif 5

Aksi Actor	Sistem	Database
Nomor 1-24 sama seperti skenario Bermain *game* utama		
	25.Menampilkan dialog pertarungan	
26.Memilih menu serang		
	27.Mengurangi *HP* musuh	
28.Memilih menu barang		
	29.Menampilkan menu *inventory*	
		30.Membaca daftar barang yang dimiliki
	31.Menampilkan daftar barang	
32.Menggunakan barang		
	33.Memperbarui kondisi karakter	
		34.Memperbarui *inventory* pemain
	35.Menampilkan daftar barang yang telah diperbarui	
36.Berhasil mengalahkan		

Aksi Actor	Sistem	Database
semua musuh		
	37. Menambahkan *drop item* ke dalam *inventory* jika tersedia	
		38. Memperbarui *inventory* pemain
	39. Menambah *EXP* untuk karakter	
		40. Memperbarui jumlah EXP karakter
	41. Muncul dialog musuh berhasil dikalahkan	

3.4.2 Skenario *Use case* Ubah Pengaturan

Skenario ubah pengaturan utama

Kondisi : Pemain sedang berada dalam menu utama

Deskripsi : Pemain ingin mengganti pengaturan dalam *game*

Tabel 3.11
Skenario ubah pengaturan utama

Aksi Actor	Reaksi Sistem
1. Memilih menu pengaturan	
	2. Menampilkan pengaturan dalam *game*
3. Mengganti pengaturan dalam *game*	
	4. Menyimpan pengaturan yang dipilih pemain
5. Melihat pengaturan yang berhasil diubah	

Skenario mengganti pengaturan alternatif 1

Kondisi : Pemain sedang berada dalam menu utama

Deskripsi : Pemain ingin membatalkan pengaturan

Tabel 3.12
Skenario mengganti pengaturan alternatif 1

Aksi Actor	Reaksi Sistem
Nomor 1-2 sama dengan skenario pengaturan utama	

Aksi Actor	Reaksi Sistem
2. Menekan tombol X / *Esc* / *Numpad* 0	
	3. Kembali menampilkan menu utama
4. Melihat tampilan menu utama	

3.4.3 Skenario *Use Case* Belajar

Skenario belajar

Kondisi : Pemain berada dalam kelas

Deskripsi : Pemain ingin melanjutkan narasi *game*

Tabel 3.13
Skenario belajar

Aksi Actor	Sistem	Database
1. Menekan tombol Z		
	2. Memeriksa data objek buku	
		3. Membaca data objek buku
	4. Menampilkan data objek buku	
5. Membaca pelajaran dalam buku		

3.4.4 Skenario *Use Case* Tes Kemampuan

Skenario Tes Kemampuan

Kondisi : Pemain belum selesai melakukan tes kemampuan yang diberikan NPC

Deskripsi : Pemain ingin menyelesaikan tes kemampuan dari NPC dalam *game*

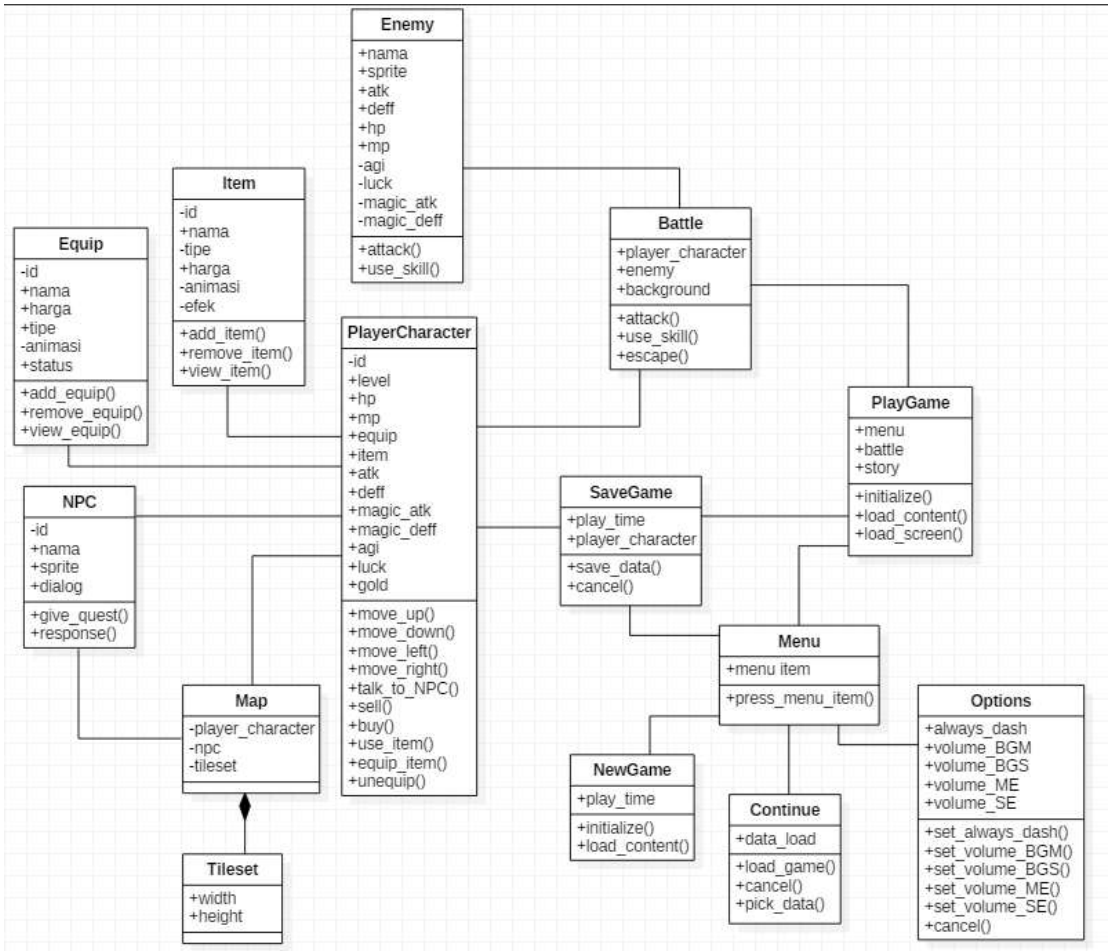
Tabel 3.14
Skenario tes kemampuan

Aksi Actor	Sistem	Database
1. Menekan tombol Z		
	2. Menampilkan tugas yang harus diselesaikan pemain	
3. Menyelesaikan permintaan dari NPC		
	4. Menampilkan dialog untuk memeriksa barang	
		5. Membaca data barang yang dimiliki pemain

Aksi Actor	Sistem	Database
	6. Memeriksa data barang sesuai dengan tugas yang diberikan	
	7. Menampilkan dialog berhasil melewati ujian	
8. Melihat dialog berhasil melewati ujian yang diberikan		

3.5 Class diagram

Class diagram menggambarkan keadaan suatu sistem dengan menjelaskan keterhubungan antara suatu kelas dengan kelas lain yang terdapat pada sistem tersebut. Gambar 3.2 merupakan *class diagram* dari *game* yang dibuat :



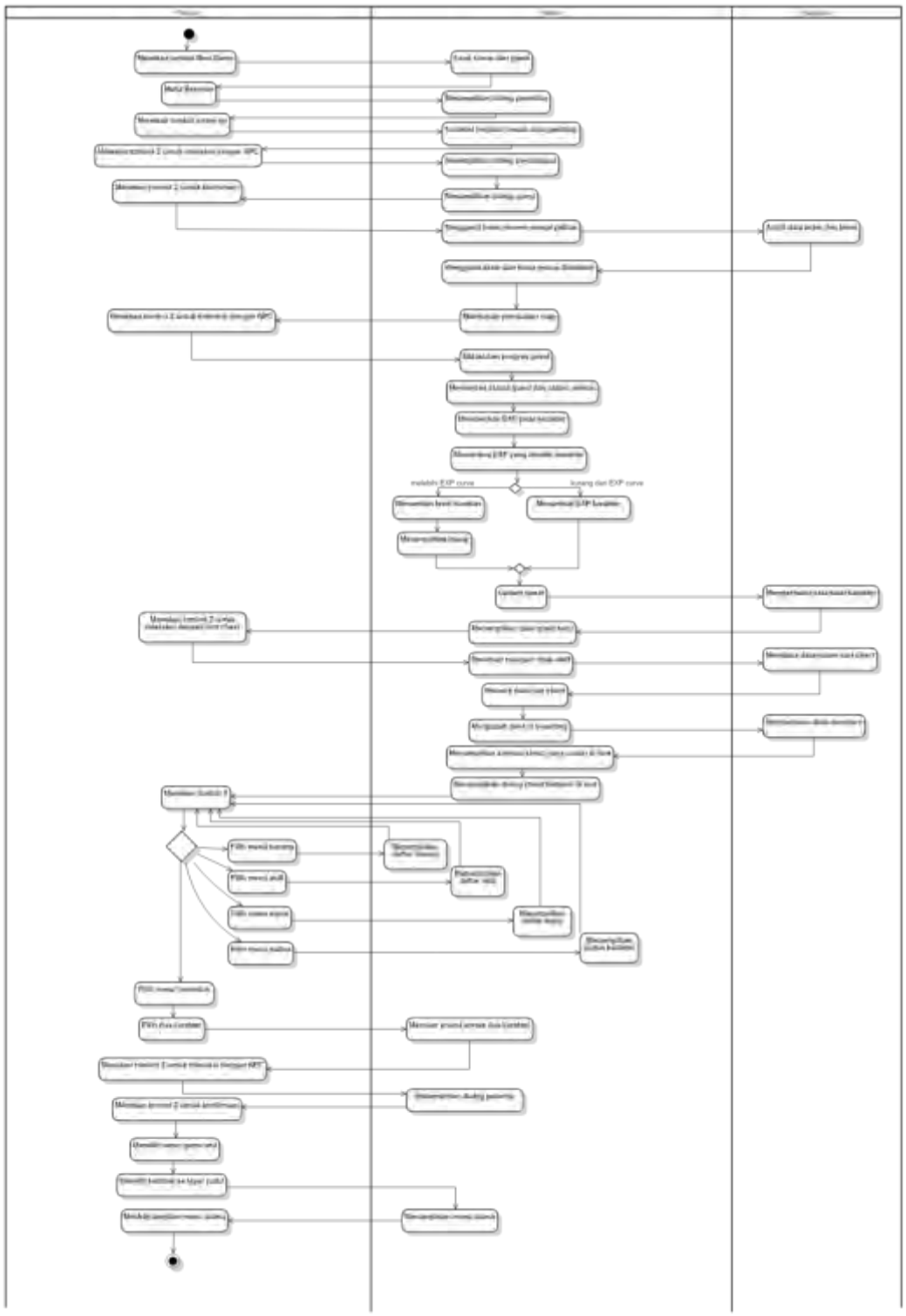
Gambar 3.2
Class Diagram

3.6 Activity diagram

Activity diagram merupakan diagram yang menjelaskan tentang alur kegiatan dalam program yang sedang dirancang, mengenai bagaimana proses berawal, keputusan-keputusan dalam sistem, dan bagaimana sistem berakhir. Berikut beberapa *activity diagram* yang dibuat :

1. *Activity diagram* bermain *game*

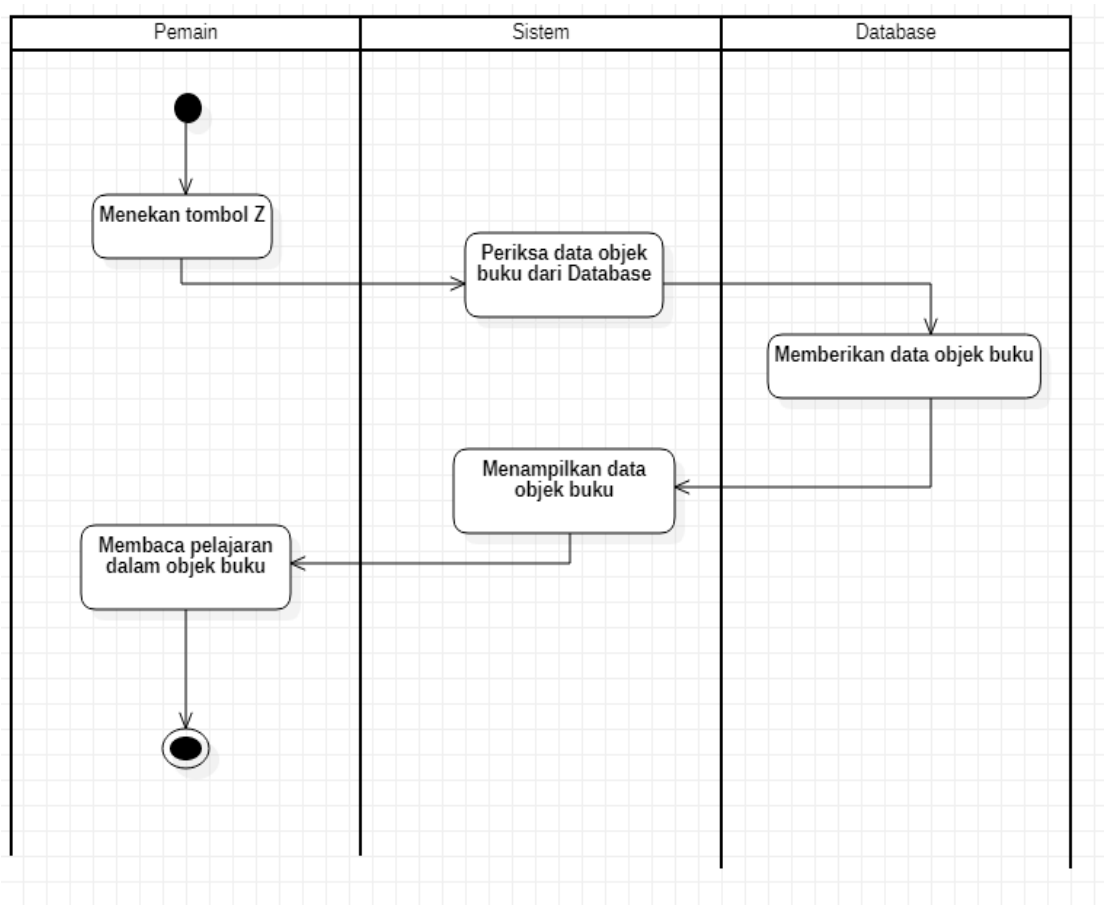
   Gambar 3.3 dibawah merupakan penjelasan ketika pemain memilih menu bermain *game*, memilih menu dalam *game*, serta ketika pemain berhasil menyelesaikan *quest* yang diberikan NPC.



Gambar 3.3
Activity diagram Bermain *Game*

2. *Activity diagram* belajar

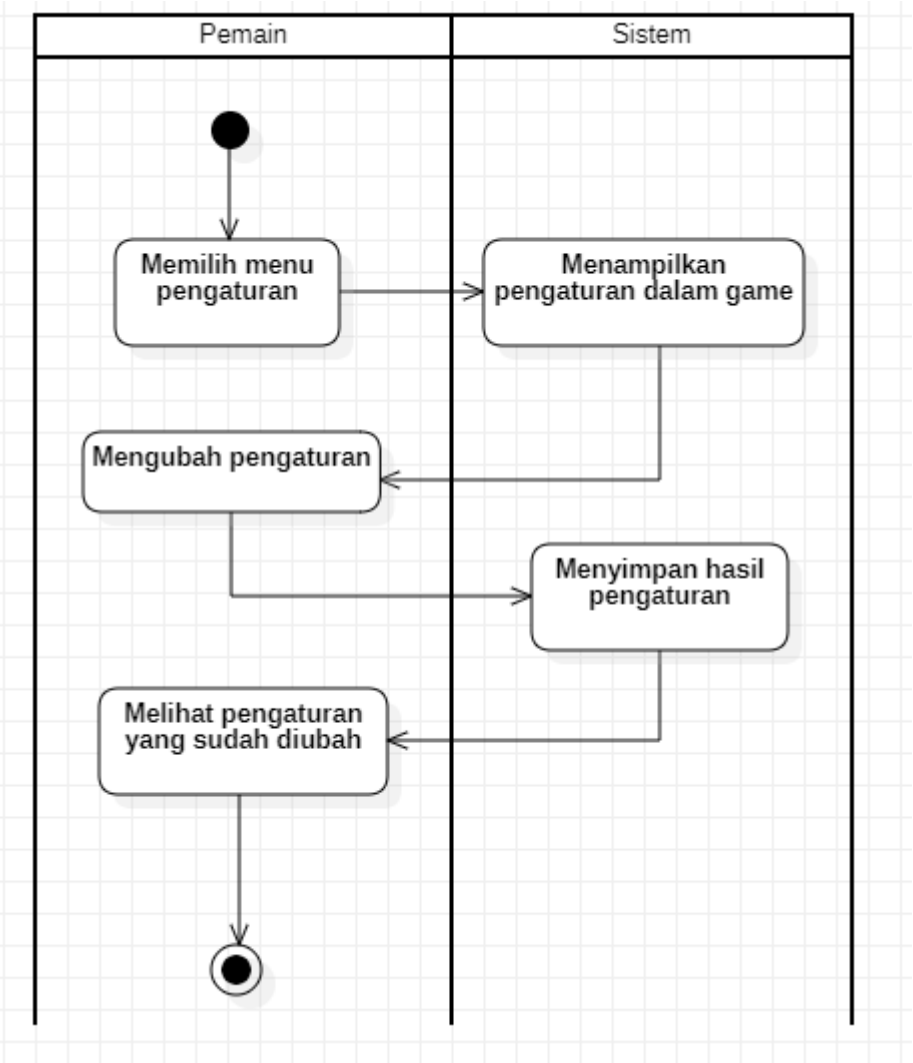
Gambar 3.4 dibawah merupakan penjelasan ketika pemain berada dalam kelas dan diharuskan untuk belajar terlebih dahulu agar bisa melanjutkan narasi dari *game*.



Gambar 3.4
Activity diagram Belajar

3. *Activity diagram* ubah pengaturan

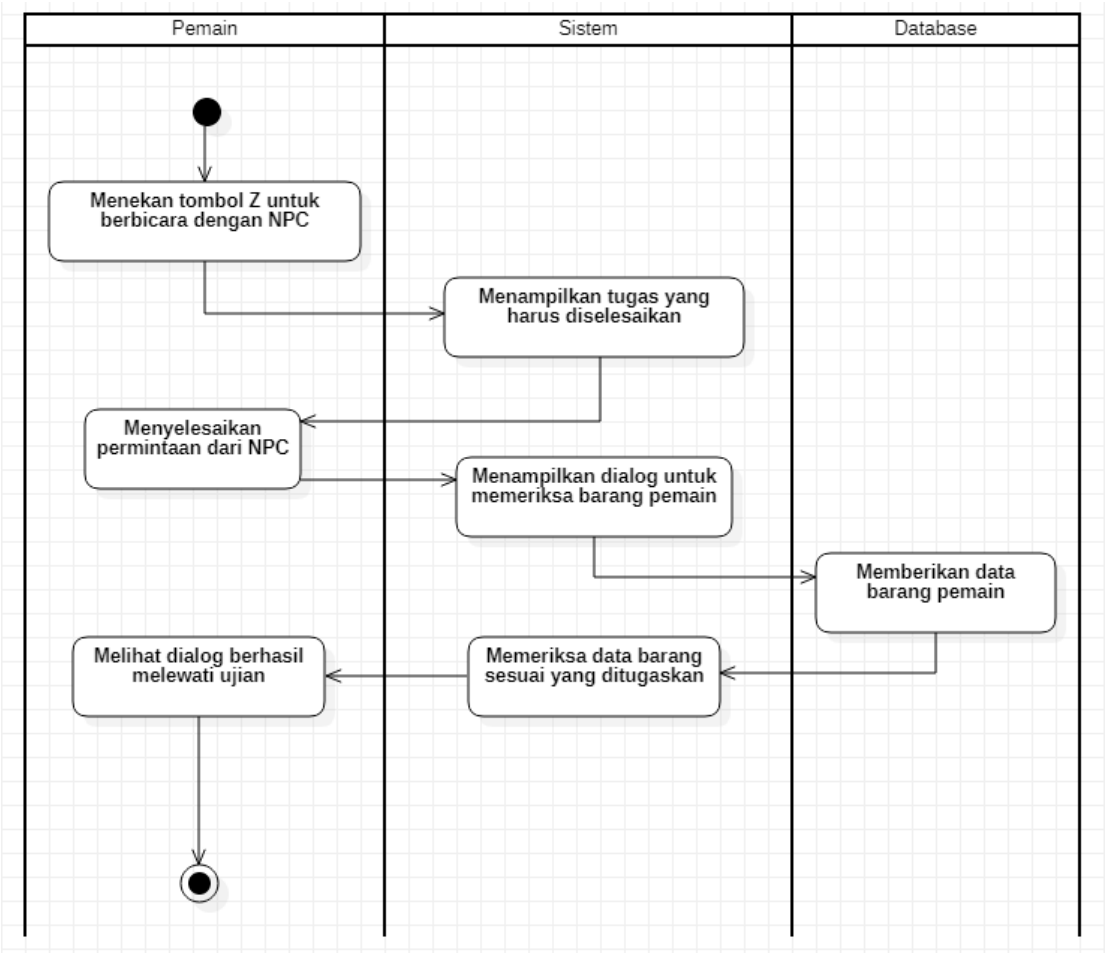
Gambar 3.5 dibawah merupakan penjelasan ketika pemain memilih menu *options* pada tampilan menu utama.



Gambar 3.5
Activity diagram Ubah Pengaturan

4. *Activity diagram* tes kemampuan

Gambar 3.6 dibawah merupakan penjelasan ketika pemain harus menyelesaikan ujian yang diberikan oleh NPC.



Gambar 3.6
Activity diagram Tes Kemampuan

3.7 Rancangan Antar Muka

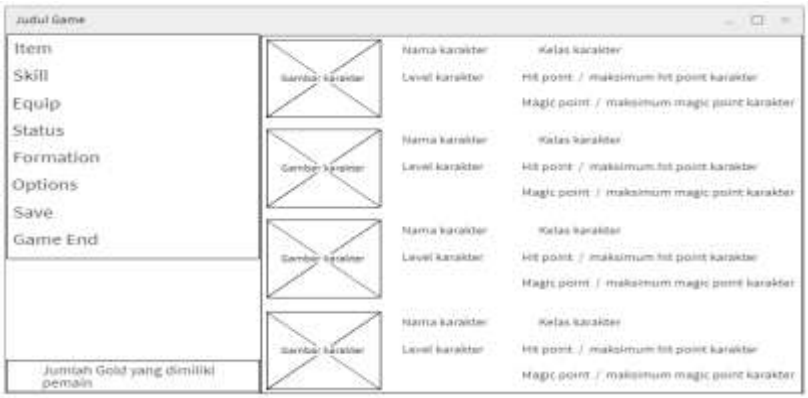
3.7.1 Rancangan Antar Muka Menu Utama

Di bawah ini merupakan rancangan menu utama dimana nantinya pemain dapat memilih menu new *game*, *continue*, atau *options*.

Gambar 3.7
Rancangan Antar Muka Menu Utama

3.7.2 Rancangan Antar Muka Menu dalam *Game*

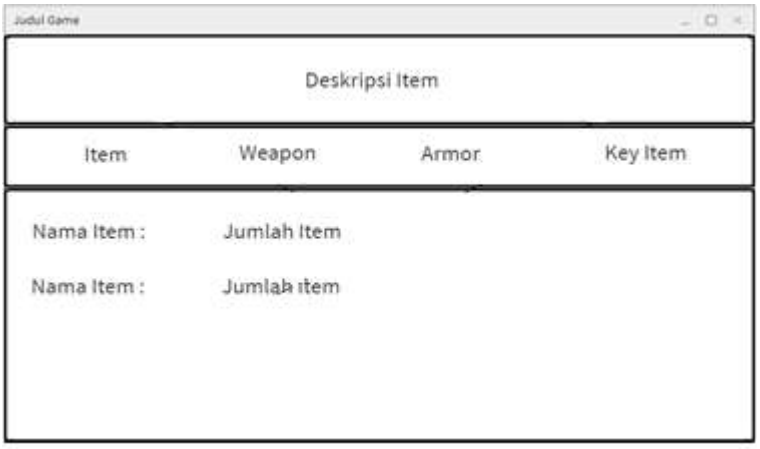
Di bawah ini merupakan rancangan menu yang terdapat dalam *game*, pemain dapat memilih menu *item*, *skill*, *equip*, dan sebagainya.



Gambar 3.8
Rancangan Antar Muka Menu dalam *Game*

3.7.3 Rancangan Antar Muka Menu *Item*

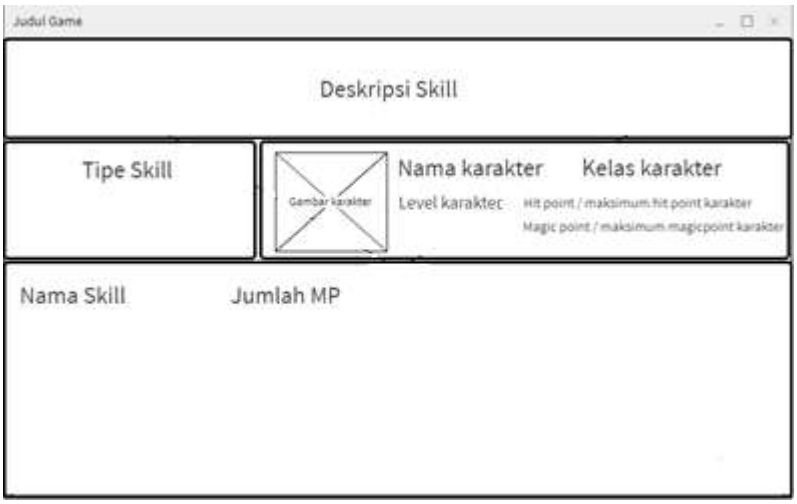
Di bawah ini merupakan rancangan antar muka untuk menu *item* dalam *game*, pemain bisa melihat jumlah *item*, deskripsi *item* pada tampilan menu.



Gambar 3.9
Rancangan Antar Muka Menu *Item*

3.7.4 Rancangan Antar Muka Menu *Skill*

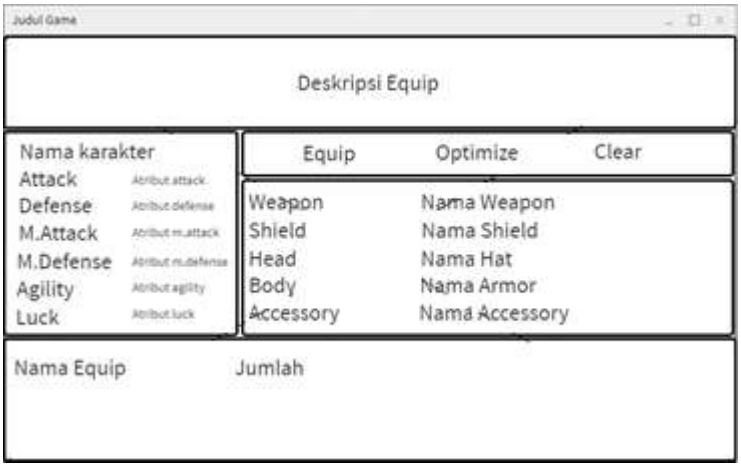
Di bawah ini merupakan rancangan antar muka untuk menu *skill* dalam *game*, *skill* yang pemain miliki, tipe *skill*, jumlah MP yang dibutuhkan dapat dilihat oleh pemain.



Gambar 3.10
Rancangan Antar Muka Menu *Skill*

3.7.5 Rancangan Antar Muka Menu *Equip*

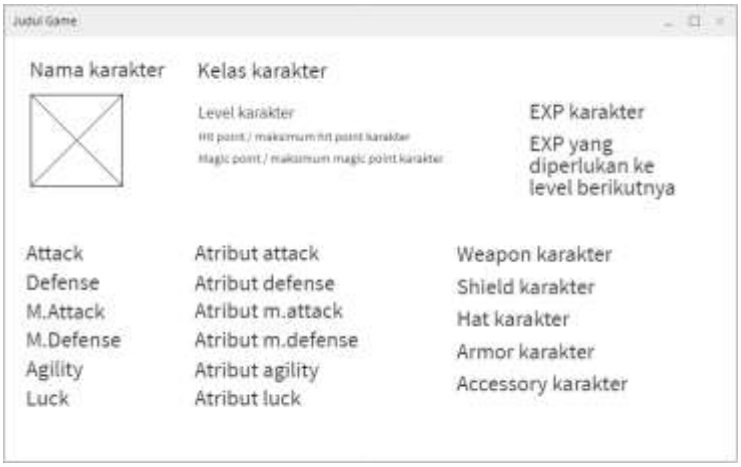
Di bawah ini merupakan rancangan antar muka untuk menu *equip*, pemain akan bisa mengubah *equip* yang digunakan karakter pada menu ini.



Gambar 3.11
Rancangan Antar Muka Menu *Equip*

3.7.6 Rancangan Antar Muka Menu Status

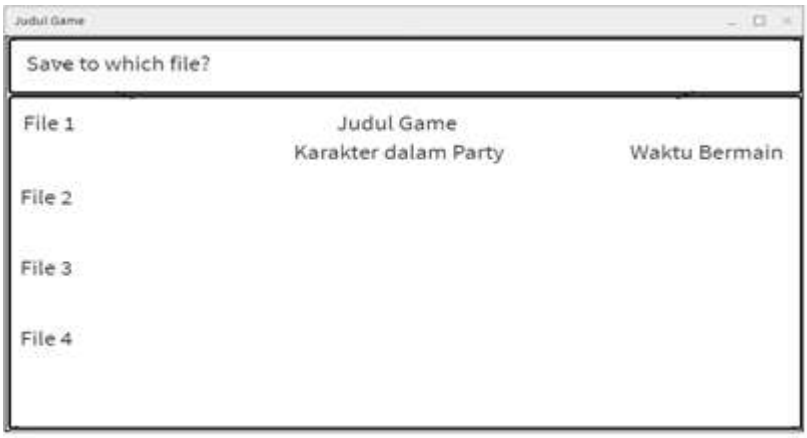
Di bawah ini merupakan rancangan antar muka untuk menu status karakter, pemain dapat melihat secara detail status apa saja yang dimiliki oleh masing-masing karakter, mulai dari *attack* sampai jumlah exp yang dibutuhkan untuk mencapai *level* berikutnya.



Gambar 3.12
Rancangan Antar Muka Menu Status

3.7.7 Rancangan Antar Muka Menu *Save*

Di bawah ini merupakan rancangan tampilan untuk pemain ketika ingin menyimpan data *game*. Pemain diberikan *slot* yang berbeda untuk menyimpan data sebanyak dua puluh.



Gambar 3.13
Rancangan Antar Muka Menu *Save*

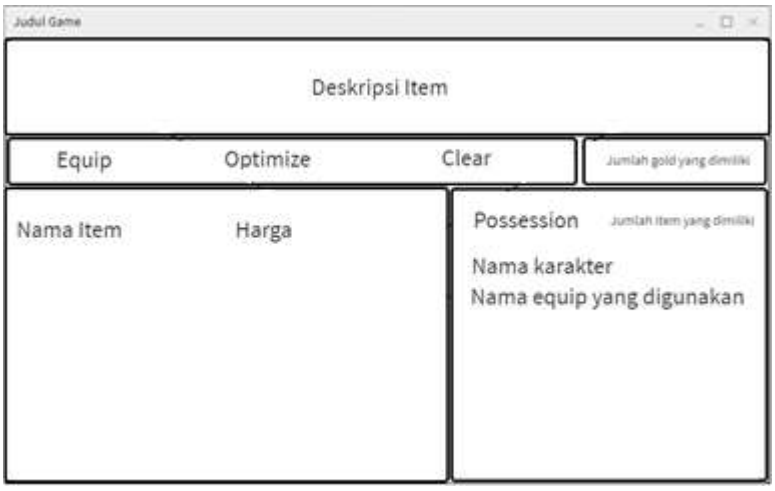
3.7.8 Rancangan Antar Muka Menu *Game* End

Di bawah ini merupakan rancangan tampilan ketika pemain selesai bermain *game* dan ingin kembali ke menu utama.

Gambar 3.14
Rancangan Antar Muka Menu *Game* End

3.7.9 Rancangan Antar Muka Toko Barang

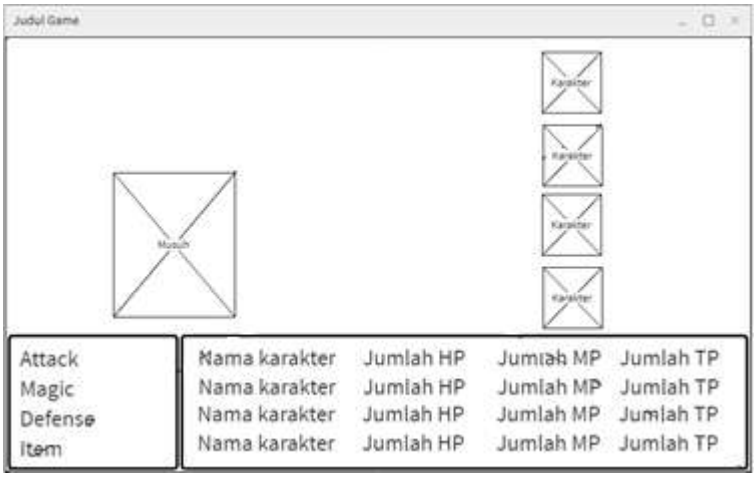
Di bawah ini merupakan rancangan tampilan ketika pemain memasuki toko barang, pemain dapat membeli atau menjual perlengkapan yang dimiliki.



Gambar 3.15
Rancangan Antar Muka Toko Barang

3.7.10 Rancangan Antar Muka *Battle*

Di bawah ini merupakan rancangan antar muka ketika karakter memasuki *battle* dan melawan musuh.



Gambar 3.16
Rancangan Antar Muka *Battle*

BAB IV
IMPLEMENTASI DAN PENGUJIAN PERANGKAT LUNAK

4.1 Spesifikasi Kebutuhan Perangkat Keras

Spesifikasi kebutuhan perangkat keras bertujuan untuk memberikan gambaran mengenai perangkat keras yang digunakan pada pengujian *game RPG* yang dibuat. Berikut ini merupakan spesifikasi perangkat keras yang digunakan dalam pengujian :

Tabel 4.1
Spesifikasi Perangkat Keras

Perangkat Keras	Keterangan Spesifikasi
Processor	AMD A6-6400K APU 3.9GHz (2 *CPU*s)
Motherboard	Gigabyte
Hard *Disk*	500 *GB HDD*
Memory	1 x 4 *GB DDR3*
VGA	Radeon *HD Graphics*
Resolusi	1360 x 768

4.2 Pengujian Antar Muka

4.2.1 Tampilan Menu Utama

Gambar 4.1
Tampilan Menu Utama

Tampilan menu utama merupakan tampilan menu yang pertama kali dilihat oleh pemain pada saat permainan dijalankan. Menu utama memiliki empat buah tombol, berikut penjelasan mengenai masing-masing tombol yang bisa dipilih pemain pada menu utama :

1. *New Game* adalah tombol yang digunakan ketika pemain belum memiliki data penyimpanan *game* sebelumnya, dan ingin memulai *game*. Setelah menekan tombol ini, sistem akan menampilkan bagian awal permainan pada gambar 4.2.
2. *Continue* merupakan tombol yang digunakan pemain, jika pemain sudah memiliki data penyimpanan *game* sebelumnya, dan ingin melanjutkan *game* yang telah disimpan. Jika pemain belum memiliki data permainan, maka tombol ini akan dinonaktifkan seperti pada gambar 4.1. Apabila pemain sudah memiliki data penyimpanan *game* sebelumnya, maka tampilan akan terlihat seperti gambar 4.3.
3. *Options* merupakan tombol yang digunakan pemain untuk mengganti pengaturan *game,* pemain dapat mengatur volume suara, volume musik, dan sebagainya jika menekan tombol ini. Tampilan pengaturan akan terlihat seperti gambar 4.4.
4. *Exit* adalah tombol yang digunakan pemain jika ingin berhenti bermain *game.*

Gambar 4.2
Tampilan Menu Utama ketika pemain menekan tombol *New Game*

Gambar 4.3
Tampilan Menu Utama ketika pemain menekan tombol *Continue*

Gambar 4.4
Tampilan Menu Utama ketika pemain menekan tombol *Options*

4.2.2 Tampilan Menu dalam *Game*

Gambar 4.5
Tampilan Menu dalam *Game*

Tampilan Menu dalam *Game* ini adalah tampilan ketika pemain menekan tombol *New Game* lalu pemain menekan tombol batal, yaitu merupakan tombol pemicu untuk membuka menu dalam *game*. Terdapat beberapa pilihan tampilan menu yang dapat dipilih oleh pemain seperti :

1. Menu *item* berfungsi untuk melihat barang yang dimiliki oleh pemain. Pemain dapat melihat jumlah barang yang dimiliki, deskripsi barang, kategori barang, dan menggunakan barang.
2. Menu *skill* berfungsi untuk melihat *skill* yang dimiliki oleh pemain, jenis *skill* dari masing-masing karakter, nama, kelas yang dimiliki, status *hit point*, *magic point*, dan juga *level* karakter terlihat di menu ini. Pemain juga dapat menggunakan *skill* yang berguna seperti *heal*.
3. Menu *equip* memiliki fungsi untuk menampilkan perlengkapan yang digunakan, memakai perlengkapan yang bisa dipakai oleh karakter, atau melepas perlengkapan yang sudah terpasang. Status seperti *attack*, *defense*, dan sebagainya juga terlihat sesuai dengan perlengkapan yang digunakan oleh karakter.

4. Menu status memiliki fungsi untuk melihat informasi mengenai karakter, seperti nama, kelas, *level*, atribut dasar seperti *attack*, *defense*, dan sebagainya. Menu ini juga menampilkan jumlah *experience* yang dimiliki, serta jumlah *experience* yang dibutuhkan untuk mencapai *level* berikutnya.
5. *Formation* digunakan pemain jika dalam *party* terdapat dua karakter atau lebih, menu ini digunakan untuk menukar urutan bertarung dari masing-masing karakter.
6. Menu *options* digunakan jika pemain ingin mengganti pengaturan dalam *game* seperti volume suara musik, efek suara, dan sebagainya.
7. Menu *save* berfungsi agar pemain dapat menyimpan data permainan, jumlah maksimal *slot* yang dimiliki pemain untuk menyimpan data adalah sebanyak dua puluh.
8. *Game end* memiliki fungsi untuk pemain kembali ke menu utama.

4.2.3 Tampilan Menu *Item*

Gambar 4.6
Tampilan Menu *Item*

Tampilan menu *item* merupakan tampilan untuk menunjukan barang apa saja yang pemain miliki, jumlah dari *item* yang dimiliki, serta terdapat deskripsi dari *item* yang dimiliki pemain. Kategori dalam menu *item* dibagi menjadi empat yaitu *item*, *weapon*, *armor*, dan *key item*.

4.2.4 Tampilan Menu *Skill*

Gambar 4.7
Tampilan Utama *Skill*

Gambar 4.7 merupakan tampilan ketika pemain membuka menu *skill*. *Skill* yang dimiliki oleh masing-masing karakter akan terlihat jika pemain membuka menu ini, setiap *skill* mempunyai deskripsi serta jumlah MP atau TP yang diperlukan untuk mengeluarkan *skill* terlihat.

Gambar 4.8
Tampilan Alternatif *Skill*

Gambar 4.8 merupakan tampilan ketika karakter memiliki jenis *skill* yang dapat digunakan di luar pertarungan, setelah pemain memilih *skill* yang akan digunakan dan memilih target, kemudian efek dari *skill* akan aktif dan MP dari karakter akan berkurang.

4.2.5 Tampilan Menu *Equip*

Gambar 4.9
Tampilan *Equip* Karakter

Gambar 4.9 merupakan tampilan saat pemain membuka menu *equip*, ketika pemain memiliki jenis perlengkapan yang sesuai dengan kelas karakter, maka akan ditampilkan pada bagian bawah, ketika pemain memilih perlengkapan akan ditampilkan juga perubahan status dari karakter jika mengenakan perlengkapan tersebut. Pilihan *optimize* berfungsi untuk mengganti seluruh perlengkapan pemain dengan atribut yang paling tinggi. Menu *clear* akan melepas semua perlengkapan yang sedang digunakan oleh karakter.

4.2.6 Tampilan Menu Status

Gambar 4.10
Tampilan Status Karakter

Gambar 4.9 merupakan tampilan ketika pemain membuka menu status, kemudian memilih karakter yang diinginkan. Pemain dapat melihat informasi dari karakter secara detail, seperti jumlah *Attack*, *Defense*, sampai exp yang dibutuhkan pemain untuk mencapai *level* berikutnya.

4.2.7 Tampilan Menu *Save*

Gambar 4.11
Tampilan *Save*

Gambar 4.10 merupakan tampilan saat pemain ingin menyimpan proses bermain *game*, kemudian pemain dapat memilih *slot* yang diinginkan untuk menyimpan data permainan. Jumlah maksimal dari *slot* penyimpanan yang dimiliki pemain adalah sebanyak dua puluh.

4.2.8 Tampilan Menu *Game End*

Gambar 4.12
Tampilan *Game End*

Gambar 4.11 merupakan tampilan ketika pemain memilih menu *game end*, menu ini berfungsi untuk mengembalikan pemain ke menu awal permainan. Setelah menekan menu ini pemain bisa memilih untuk kembali ke menu utama atau membatalkan aksi.

4.2.9 Tampilan Toko Barang

Gambar 4.13
Tampilan *Buy Item*

Gambar 4.12 merupakan tampilan ketika pemain berinteraksi dengan NPC penjual *item*, pemain dapat membeli berbagai jenis perlengkapan dan *item* yang tersedia. Jika pemain tidak memiliki jumlah *gold* yang cukup, warna *item* akan menjadi redup menandakan pemain tidak dapat membeli barang tersebut. Saat pemain ingin membeli perlengkapan, terdapat juga informasi mengenai perbedaan status dari perlengkapan yang dimiliki.

Gambar 4.14
Tampilan Sell *Item*

Gambar 4.13 merupakan tampilan saat pemain ingin menjual *item* yang dimiliki, jumlah *item* serta harga akan ditampilkan pada layar pemain. Setelah pemain memilih *item*, kemudian pemain bisa menentukan berapa banyak *item* yang ingin dijual kepada NPC.

4.2.10 Tampilan *Battle*

Gambar 4.15
Tampilan *Battle*

Gambar 4.14 merupakan tampilan pada saat pemain bertarung melawan musuh. Pemain dapat memilih untuk menyerang, menggunakan *skill*, bertahan, atau menggunakan *item* dalam pertarungan. Terdapat dua jenis pertarungan melawan *monster* yaitu ketika pemain berada dalam *dungeon*, maka pemain akan menemukan *monster* yang menghadang jalan pemain, jika pemain menyentuh *monster* maka pemain tidak dapat melarikan diri dari pertarungan. Jenis kedua adalah ketika pemain sedang berjalan, maka terdapat kesempatan bagi pemain untuk bertarung melawan *monster*, pada jenis pertarungan ini pemain dapat memilih untuk melarikan diri dari *monster*.

4.3 Pengujian Fungsi

4.3.1 Pengujian Menu Utama

Tabel 4.2
Pengujian Menu Utama

No Uji	Data yang diuji	Hasil yang diharapkan	Hasil Pengujian	Status
PMU-01	Tombol *New Game.*	Membuka tampilan awal *game*.	Membuka tampilan awal *game*.	Sesuai
PMU-02	Tombol *Continue*, terdapat data penyimpanan.	Menampilkan *slot* yang berisi data penyimpanan beserta waktu bermain.	Menampilkan *slot* yang berisi data penyimpanan beserta waktu bermain.	Sesuai
PMU-03	Tombol *Continue*, tidak terdapat data penyimpanan.	Tombol tidak dapat ditekan dan tidak terjadi apapun.	Tombol tidak dapat ditekan dan tidak terjadi apapun.	Sesuai
PMU-04	Tombol *Options.*	Menampilkan pengaturan karakter otomatis lari, serta pengaturan volume	Menampilkan pengaturan karakter otomatis lari, serta pengaturan volume	Sesuai
PMU-05	Tombol Exit	Menutup aplikasi *game*.	Menutup aplikasi *game*.	Sesuai

Berdasarkan tabel 4.2, maka dapat disimpulkan bahwa pengujian menu utama sudah sesuai dengan hasil yang diharapkan.

4.3.2 Pengujian Menu dalam *Game*

Tabel 4.3
Pengujian Menu dalam *Game*

No Uji	Data yang diuji	Hasil yang diharapkan	Hasil Pengujian	Status
PMG-01	Tombol *Item,* pemain tidak memiliki *item.*	Menampilkan *inventory* yang tidak berisi *item* .	Menampilkan *inventory* yang tidak berisi *item* .	Sesuai
PMG-02	Tombol *Item,* pemain memiliki *item.*	Menampilkan *inventory* yang berisi *item*, serta jumlah dari masing-masing *item*.	Menampilkan *inventory* yang berisi *item*, serta jumlah dari masing-masing *item*.	Sesuai

PMG-03	Tombol *Skill,* karakter tidak memiliki *skill.*	Menampilkan data karakter yang tidak memiliki *skill*	Menampilkan data karakter yang tidak memiliki *skill*	Sesuai
PMG-04	Tombol *Skill,* karakter memiliki *skill.*	Menampilkan *skill* yang dapat digunakan karakter, serta jumlah MP yang dibutuhkan.	Menampilkan *skill* yang dapat digunakan karakter, serta jumlah MP yang dibutuhkan.	Sesuai
PMG-05	Tombol *Equip,* karakter tidak memiliki *equip*ment.	Menampilkan status karakter, dan *equip*ment tidak tersedia.	Menampilkan status karakter, dan *equip*ment tidak tersedia.	Sesuai
PMG-06	Tombol *Equip,* karakter memiliki *equip*ment.	Menampilkan status karakter, dan *equip*ment yang tersedia.	Menampilkan status karakter, dan *equip*ment yang tersedia.	Sesuai
PMG-07	Tombol *Status*	Menampilkan status karakter	Menampilkan status karakter	Sesuai
PMG-08	Tombol *Formation,* pemain hanya memiliki satu karakter dalam *party*.	Tombol tidak dapat ditekan dan tidak terjadi apapun.	Tombol tidak dapat ditekan dan tidak terjadi apapun.	Sesuai
PMG-09	Tombol *Formation,* pemain memiliki minimal dua karakter dalam *party*.	Menampilkan pilihan untuk menukar posisi antara dua karakter	Menampilkan pilihan untuk menukar posisi antara dua karakter	Sesuai
PMG-10	Tombol *Options.*	Menampilkan pengaturan karakter otomatis lari, serta pengaturan volume	Menampilkan pengaturan karakter otomatis lari, serta pengaturan volume	Sesuai
PMG-11	Tombol *Save,* pemain belum memiliki data *game.*	Menampilkan *slot* kosong untuk menyimpan data *game.*	Menampilkan *slot* kosong untuk menyimpan data *game.*	Sesuai
PMG-12	Tombol *Save,* pemain memiliki data *game.*	Menampilkan *slot* yang berisi data penyimpanan beserta waktu bermain.	Menampilkan *slot* yang berisi data penyimpanan beserta waktu bermain.	Sesuai
PMG-13	Tombol *Game End.*	Menampilkan pilihan To Title atau Cancel	Menampilkan pilihan To Title atau Cancel	Sesuai

Berdasarkan tabel 4.3, maka dapat disimpulkan bahwa pengujian menu dalam *game* sudah sesuai dengan hasil yang diharapkan.

4.3.3 Pengujian Menu *Item*

Tabel 4.4
Pengujian Menu *Item*

No Uji	Data yang diuji	Hasil yang diharapkan	Hasil Pengujian	Status
PMI-01	Tombol *Item,* pemain memilih *item* pada *inventory.*	Menampilkan deskripsi dari *item* yang dipilih pemain beserta jumlah yang pemain miliki.	Menampilkan deskripsi dari *item* yang dipilih pemain beserta jumlah yang pemain miliki.	Sesuai
PMI-02	Tombol *Weapon,* pemain memiliki senjata pada *inventory.*	Menampilkan senjata yang tidak digunakan oleh karakter beserta deskripsi.	Menampilkan senjata yang tidak digunakan oleh karakter beserta deskripsi.	Sesuai
PMI-03	Tombol *Weapon,* pemain memilih senjata pada *inventory* dan menekan tombol aksi.	Senjata tidak dapat digunakan dan tidak terjadi apapun.	Senjata tidak dapat digunakan dan tidak terjadi apapun.	Sesuai
PMI-04	Tombol *Armor,* pemain memiliki *armor* pada *inventory.*	Menampilkan berbagai jenis armor yang tidak digunakan oleh karakter beserta deskripsi.	Menampilkan berbagai jenis armor yang tidak digunakan oleh karakter beserta deskripsi.	Sesuai
PMI-05	Tombol *Armor,* pemain memilih senjata pada *inventory* dan menekan tombol aksi.	Armor tidak dapat digunakan dan tidak terjadi apapun.	Armor tidak dapat digunakan dan tidak terjadi apapun.	Sesuai
PMI-06	Tombol *Key Item,* pemain memiliki *key item* pada *inventory.*	Menampilkan berbagai jenis key *item* yang dimiliki oleh pemain beserta deskripsi.	Menampilkan berbagai jenis key *item* yang dimiliki oleh pemain beserta deskripsi.	Sesuai

Berdasarkan tabel 4.4, maka dapat disimpulkan bahwa pengujian menu *item* sudah sesuai dengan hasil yang diharapkan.

4.3.4 Pengujian Menu *Skill*

Tabel 4.5
Pengujian Menu *Skill*

No Uji	Data yang diuji	Hasil yang diharapkan	Hasil Pengujian	Status
PMSK-01	Tombol *Skill* dan pemain memilih jenis *skill* yang dimiliki karakter.	Menampilkan semua *skill* yang karakter miliki beserta deskripsi dan MP yang dibutuhkan.	Menampilkan semua *skill* yang karakter miliki beserta deskripsi dan MP yang dibutuhkan.	Sesuai
PMSK-02	Tombol *Skill* dan pemain menekan tombol aksi pada combat *skill.*	*Skill* tidak dapat digunakan dan tidak terjadi apapun.	*Skill* tidak dapat digunakan dan tidak terjadi apapun.	Sesuai
PMSK-03	Tombol *Skill* dan pemain menekan tombol aksi pada jenis *skill* yang dapat digunakan di luar combat.	Menampilkan pilihan target dari *skill* yang akan digunakan.	Menampilkan pilihan target dari *skill* yang akan digunakan.	Sesuai

Berdasarkan tabel 4.5, maka dapat disimpulkan bahwa pengujian menu *skill* sudah sesuai dengan hasil yang diharapkan.

4.3.5 Pengujian Menu *Equip*

Tabel 4.6
Pengujian Menu *Equip*

No Uji	Data yang diuji	Hasil yang diharapkan	Hasil Pengujian	Status
PME-01	Tombol *Equip* dan pemain melepas *equip*ment yang terpasang pada karakter.	Menampilkan perubahan atribut yang akan karakter miliki jika melepas *equip*ment.	Menampilkan perubahan atribut yang akan karakter miliki jika melepas *equip*ment.	Sesuai
PME-02	Tombol *Equip,* pemain memiliki *equip*ment yang dapat dipasang pada karakter.	Menampilkan *equipment* yang dapat dipasang pada karakter, dan terlihat perubahan status karakter jika menggunakan *equipment* baru.	Menampilkan *equipment* yang dapat dipasang pada karakter, dan terlihat perubahan status karakter jika menggunakan *equipment* baru.	Sesuai
PME-03	Tombol *Equip*, karakter memiliki *equip*ment, serta pemain menekan tombol *Optimize*	Menampilkan status karakter, dan mengganti *equip*ment dengan status paling tinggi.	Menampilkan status karakter, dan mengganti *equip*ment dengan status paling tinggi.	Sesuai
PME-04	Tombol *Equip*, pemain menekan tombol *Clear*	Melepas *equip*ment yang terpasang pada karakter, kemudian menampilkan status dari karakter.	Melepas *equip*ment yang terpasang pada karakter, kemudian menampilkan status dari karakter.	Sesuai

Berdasarkan tabel 4.6, maka dapat disimpulkan bahwa pengujian menu *equip* sudah sesuai dengan hasil yang diharapkan.

4.3.6 Pengujian Menu Status

Tabel 4.7
Pengujian Menu Status

No Uji	Data yang diuji	Hasil yang diharapkan	Hasil Pengujian	Status
PMS-01	Tombol *Status,* pemain memilih karakter.	Menampilkan status karakter.	Menampilkan status karakter.	Sesuai
PMS-02	Tombol *Status,* pemain mengubah *equip*ment karakter.	Menampilkan status karakter yang baru.	Menampilkan status karakter yang baru.	Sesuai
PMS-03	Tombol *Status,* setelah pemain mengalahkan *monster*.	Menampilkan pembaharuan status EXP karakter.	Menampilkan pembaharuan status EXP karakter.	Sesuai
PMS-04	Tombol *Status,* pemain memilih karakter, dan menekan tombol halaman selanjutnya.	Menampilkan status karakter berikutnya.	Menampilkan status karakter berikutnya.	Sesuai
PMS-05	Tombol *Status,* pemain memilih karakter, dan menekan tombol halaman selanjutnya.	Tombol *Status,* pemain memilih karakter, dan menekan tombol halaman sebelumnya.	Tombol *Status,* pemain memilih karakter, dan menekan tombol halaman sebelumnya.	Sesuai

Berdasarkan tabel 4.7, maka dapat disimpulkan bahwa pengujian menu status sudah sesuai dengan hasil yang diharapkan.

4.3.7 Pengujian Menu *Save*

Tabel 4.8
Pengujian Menu *Save*

No Uji	Data yang diuji	Hasil yang diharapkan	Hasil Pengujian	Status
PMSV-01	Tombol *Save,* pemain belum memiliki data *game*, dan memilih *slot* satu untuk menyimpan.	Data *game* berhasil disimpan dalam *slot* satu, dan kembali ke menu dalam *game*	Data *game* berhasil disimpan, dan kembali ke menu dalam *game*	Sesuai
PMSV-02	Tombol *Save,* pemain sudah memiliki data *game* pada *slot* satu, dan memilih *slot* satu untuk menyimpan.	Data *game* berhasil ditimpa dan disimpan dalam *slot* satu, kemudian kembali ke menu dalam *game*	Data *game* berhasil ditimpa dan disimpan dalam *slot* satu, kemudian kembali ke menu dalam *game*	Sesuai
PMSV-03	Tombol *Save,* pemain belum memiliki data *game*, dan menekan tombol batal.	Kembali ke menu dalam *game*.	Kembali ke menu dalam *game*.	Sesuai
PMSV-04	Tombol *Save,* pemain memiliki data *game*, dan menekan tombol batal.	Kembali ke menu dalam *game*.	Kembali ke menu dalam *game*.	Sesuai

Berdasarkan tabel 4.8, maka dapat disimpulkan bahwa pengujian menu utama *save* sesuai dengan hasil yang diharapkan.

4.3.8 Pengujian Menu *Game* End

Tabel 4.9
Pengujian Menu *Game* End

No Uji	Data yang diuji	Hasil yang diharapkan	Hasil Pengujian	Status
PMG-01	Tombol *Game End* dan pemain memilih To Title.	Kembali ke menu utama *game*.	Kembali ke menu utama *game*.	Sesuai
PMG-02	Tombol *Game End* dan pemain menekan tombol batal.	Kembali ke menu dalam *game*.	Kembali ke menu dalam *game*.	Sesuai
PMG-03	Tombol *Game End* dan pemain memilih *Cancel.*	Kembali ke menu dalam *game*.	Kembali ke menu dalam *game*.	Sesuai

Berdasarkan tabel 4.9, maka dapat disimpulkan bahwa pengujian menu *game end* sudah sesuai dengan hasil yang diharapkan.

4.3.9 Pengujian Toko Barang

Tabel 4.10
Pengujian Toko Barang

No Uji	Data yang diuji	Hasil yang diharapkan	Hasil Pengujian	Status
PTB-01	Pemain berinteraksi dengan NPC penjual barang.	Menampilkan dialogue, membuka menu transaksi.	Menampilkan dialogue, membuka menu transaksi.	Sesuai
PTB-02	Tombol *buy.*	Menampilkan barang-barang yang tersedia untuk dibeli pemain.	Menampilkan barang-barang yang tersedia untuk dibeli pemain.	Sesuai
PTB-03	Tombol *buy,* pemain tidak memiliki cukup *gold* untuk	Barang tidak dapat dibeli oleh pemain.	Barang tidak dapat dibeli oleh pemain.	Sesuai

	membeli barang.			
PTB-04	Tombol *buy,* pemain memiliki cukup *gold* untuk membeli barang.	Barang berhasil dibeli oleh pemain dan masuk ke dalam *inventory*.	Barang berhasil dibeli oleh pemain dan masuk ke dalam *inventory*.	Sesuai
PTB-05	Tombol *sell.*	Menampilkan barang-barang yang tersedia untuk dijual pemain.	Menampilkan barang-barang yang tersedia untuk dijual pemain.	Sesuai
PTB-06	Tombol *sell*, pemain menjual *item* yang dimiliki.	Barang dalam *inventory* berhasil berkurang, *gold* yang dimiliki bertambah.	Barang dalam *inventory* berhasil berkurang, *gold* yang dimiliki bertambah.	Sesuai
PTB-07	Tombol *Cancel*.	Menutup menu transaksi, dan kembali ke dalam permainan.	Menutup menu transaksi, dan kembali ke dalam permainan.	Sesuai

Berdasarkan tabel 4.10, maka dapat disimpulkan bahwa pengujian menu toko barang sudah sesuai dengan hasil yang diharapkan.

4.3.10 Pengujian *Battle*

Tabel 4.11
Pengujian *Battle*

No Uji	Data yang diuji	Hasil yang diharapkan	Hasil Pengujian	Status
PB-01	Pemain dihampiri *monster* dalam *dungeon*.	Masuk ke dalam *battle*, tombol *escape* tidak aktif.	Masuk ke dalam *battle*, tombol *escape* tidak aktif.	Sesuai
PB-02	Tombol *Fight*.	Menampilkan pilihan *Attack*, *magic*, *guard*, dan *item*.	Menampilkan pilihan *Attack*, *magic*, *guard*, dan *item*.	Sesuai
PB-03	Tombol *Fight*, memilih *Attack* dan memilih target.	Karakter menyerang *monster* sesuai target pilihan.	Karakter menyerang *monster* sesuai target pilihan.	Sesuai
PB-04	Tombol *Fight*, memilih *magic*, karakter tidak memiliki MP /	*Skill* tidak dapat digunakan.	*Skill* tidak dapat digunakan.	Sesuai

	TP yang cukup.			
PB-05	Tombol *Fight*, memilih *magic*, karakter memiliki MP / TP yang cukup.	Karakter dapat menggunakan *skill*.	Karakter dapat menggunakan *skill*.	Sesuai
PB-06	Tombol *Fight*, dan memilih *guard*.	Karakter masuk ke dalam mode bertahan dan mendapatkan TP.	Karakter masuk ke dalam mode bertahan dan mendapatkan TP.	Sesuai
PB-07	Tombol *Fight*, dan memilih *item* yang dapat digunakan dalam pertarungan.	Efek dari *item* berhasil digunakan, *item* berkurang dari *inventory*.	Efek dari *item* berhasil digunakan, *item* berkurang dari *inventory*.	Sesuai
PB-08	Tombol *Fight*, dan memilih *item* yang tidak dapat digunakan dalam pertarungan.	Karakter menggunakan *item* dan tidak mendapat efek apapun, *item* tidak berkurang dari *inventory*.	Karakter menggunakan *item* dan tidak mendapat efek apapun, *item* tidak berkurang dari *inventory*.	Sesuai

Berdasarkan tabel 4.11, maka dapat disimpulkan bahwa pengujian *battle* sudah sesuai dengan hasil yang diharapkan.

Rangkuman dari seluruh pengujian fungsionalitas yang ada, akan disajikan di tabel 4.12 di bawah ini :

Tabel 4.12
Rangkuman Pengujian Fungsionalitas

No.	Nama Pengujian	% Hasil Pengujian
1	Menu Utama	100%
2	Menu dalam *Game*	100%
3	Menu *Item*	100%
4	Menu *Skill*	100%
5	Menu *Equip*	100%

6	Menu Status	100%
7	Menu *Save*	100%
8	Menu *Game End*	100%
9	Toko Barang	100%
10	*Battle*	100%
	Rata-rata	100%

Berdasarkan hasil pengujian pada tabel 4.12, diperoleh hasil rata-rata 100% maka dapat disimpulkan bahwa seluruh fungsi-fungsi yang ada dalam *game* sudah berjalan dengan baik tanpa adanya kesalahan.

4.4 Pengujian Performa

Pengujian performa dilakukan untuk menguji hal-hal yang berkaitan dengan performa *game* ketika dimainkan. Beberapa data yang akan diuji meliputi waktu *loading* sampai memasuki menu utama, waktu *loading* setiap memasuki desa, waktu *loading* sampai video berhasil diputar. Untuk pengujian waktu *loading* diperlukan kondisi khusus, yaitu setiap kali percobaan selesai, PC yang dipakai untuk pengujian di *restart* kembali.

Tabel 4.13
Pengujian Performa

No Uji	Data yang diuji	Hasil yang diharapkan	Hasil Pengujian	Status
PF-01	Waktu *loading* masuk menu utama.	Waktu *loading* tidak melebihi satu detik.	Waktu *loading* adalah 0.74 detik	Sesuai
PF-02	Waktu *loading* masuk menu utama.	Waktu *loading* tidak melebihi satu detik.	Waktu *loading* adalah 0.68 detik	Sesuai
PF-03	Waktu *loading* masuk menu utama.	Waktu *loading* tidak melebihi satu detik.	Waktu *loading* adalah 0.76 detik	Sesuai
PF-04	Waktu *loading* masuk desa pertama.	Waktu *loading* tidak melebihi dua detik.	Waktu *loading* adalah 1.12 detik	Sesuai

PF-05	Waktu *loading* masuk desa kedua.	Waktu *loading* tidak melebihi dua detik.	Waktu *loading* adalah 1.00 detik	Sesuai
PF-06	Waktu *loading* masuk desa ketiga.	Waktu *loading* tidak melebihi dua detik.	Waktu *loading* adalah 1.18 detik	Sesuai
PF-07	Waktu *loading* pemutaran video pembuatan gula aren.	Waktu *loading* tidak melebihi satu detik.	Waktu *loading* adalah 0.67 detik	Sesuai
PF-08	Waktu *loading* pemutaran video pembuatan garam.	Waktu *loading* tidak melebihi satu detik.	Waktu *loading* adalah 0.57 detik	Sesuai

Berdasarkan hasil pengujian pada tabel 4.13, maka dapat disimpulkan bahwa performa aplikasi sudah sesuai dengan hasil yang diharapkan, dimana rata-rata waktu loading dari masing-masing pengujian berada dibawah satu detik.

4.5 Pengujian Sumber Daya

Pengujian sumber daya diperlukan untuk mengetahui seberapa besar aplikasi *game* menghabiskan sumber daya yang tersedia. Pengujian ini hanya berfokus kepada seberapa besar sumber daya *memory* yang diperlukan ketika *game* dimainkan.

Tabel 4.14
Pengujian Sumber Daya

No Uji	Data yang diuji	Hasil yang diharapkan	Hasil Pengujian	Status
PSD-01	Penggunaan *memory* pada menu utama.	Penggunaan *memory* tidak melebihi 150 MB.	Penggunaan *memory* mencapai 78 MB.	Sesuai
PSD-02	Penggunaan *memory* pada awal permainan.	Penggunaan *memory* tidak melebihi 150 MB.	Penggunaan *memory* mencapai 104 MB.	Sesuai
PSD-03	Penggunaan *memory* saat pemain masuk desa.	Penggunaan *memory* tidak melebihi 150 MB.	Penggunaan *memory* mencapai 117 MB.	Sesuai

PSD-04	Penggunaan *memory* saat pemain bertarung.	Penggunaan *memory* tidak melebihi 200 MB.	Penggunaan *memory* mencapai 140 MB.	Sesuai
PSD-05	Penggunaan *memory* saat pemain melihat video pembelajaran.	Penggunaan *memory* tidak melebihi 200 MB.	Penggunaan *memory* mencapai 165 MB.	Sesuai

Berdasarkan hasil pengujian pada tabel 4.14, maka dapat disimpulkan bahwa nggunaan sumber daya pada aplikasi sudah sesuai dengan hasil yang diharapkan, dimana penggunaan *memory* ketika *game* berjalan tidak melebihi dari hasil yang diharapkan, yaitu tidak melebihi 200 MB.

4.6 Pengujian Kuesioner

Selain pengujian-pengujian yang telah dilakukan oleh pihak pengembang, pengujian dari pihak *expertise* yang dilakukan oleh mahasiswa teknik kimia dari Universitas Parahyangan pun telah dilakukan guna menambah kualitas dari *game* yang dibuat. Pengukuran berupa skor yang diberikan oleh responden terhadap pernyataan-pernyataan untuk menguji *user interface*, dan juga menguji *user experience*. Rentang kategori nilai dikelompokkan sebagai berikut :

1. Penilaian 0 – 25 % termasuk dalam kategori tidak baik.
2. Penilaian 25 – 50 % termasuk dalam kategori kurang baik.
3. Penilaian 50 – 75 % termasuk dalam kategori baik.
4. Penilaian 75 – 100 % termasuk dalam kategori sangat baik

Perhitungan persentase diperoleh dengan cara :

$$\% = \frac{nilai}{total\ nilai} x\ 100\%$$

Tabel 4.15
Pengujian Kuesioner

Variabel	Total Nilai	Nilai	%	Kategori
User Interface	90	67	74%	Baik
User Experience	90	69	77%	Sangat Baik

Berdasarkan tabel 4.15, persentase *user interface* dinyatakan baik karena responden menilai tampilan, karakter, simbol-simbol, dan bahasa dalam *game* mudah dipahami oleh pengguna, namun masih ada kekurangan dalam hal fitur-fitur dalam *game*. Untuk *user experience*, persentase dinyatakan sangat baik karena responden menilai *game* yang dibuat memiliki instruksi, alur cerita, dan konten edukasi mudah dipahami oleh pengguna.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan hasil perancangan dan pengujian yang telah dilakukan, maka diperoleh kesimpulan sebagai berikut :

1. Penerapan zat aditif dan zat adiktif sebagai bahan pembelajaran siswa berhasil dibentuk menjadi *item-item* yang terdapat dalam *game* dan berguna sebagai *main quest* bagi pemain.
2. Perancangan *game* edukasi yang bertujuan untuk membantu siswa dalam mempelajari materi zat aditif dan zat adiktif dapat dibuat dan semua fungsionalitas yang ada di dalamnya sesuai dengan hasil yang diharapkan.
3. *Game* edukasi yang ditujukan untuk membantu belajar siswa mudah dimengerti oleh pengguna, dimana hasil yang didapatkan berdasarkan kuesioner menyatakan bahwa variabel *user interface* mendapat nilai persentase sebanyak 74% dan termasuk dalam kategori baik.
4. *Game* edukasi yang ditujukan untuk membantu belajar siswa mudah digunakan, berdasarkan hasil pengujian terhadap pengguna dimana hasil yang didapatkan menyatakan bahwa variabel *user experience* mendapatkan nilai persentase sebanyak 77% dan termasuk dalam kategori sangat baik.

5.2 Saran

Berdasarkan implementasi dan uji coba yang telah dilakukan, masih terdapat beberapa hal yang menjadi kekurangan pada perangkat lunak yang dibuat dan berikut pengembangan yang dapat dilakukan :

1. Menambah *quest* yang dapat berjalan berdasarkan *trigger* tertentu agar *game* yang dibuat menjadi lebih menarik.

2. Menambah aktivitas NPC dalam desa guna memberi informasi tambahan kepada pemain.
3. Membuat proses belajar dalam *game* menjadi lebih interaktif bagi pemain.
4. Menambah konten edukasi dalam *game* yang dibuat.

DAFTAR PUSTAKA

Adams, E. (2010) *Fundamentals Of Game Design Second Edition*, *Journal of Graphic Design and Modeling*.

Bruegge, B. and Dutoit, A. H. (2014) *Object-oriented software engineering : using UML, patterns and Java*.

Chang, R. and Goldsby, K. (2016) *Chemistry, Twelfth Edition*, *Journal of Chemical Information and Modeling*.

Hariyati, 2017 (2017) 'PENGEMBANGAN MEDIA GAME EDUKASI KIMIA MENGGUNAKAN SCRATCH PADA ANAK TAHAPAN OPERASIONAL FORMAL'.

Kalmpourtzis, G. (2018) *Educational Game Design Fundamentals: A Journey to Creating Intrinsically Motivating Learning Experiences*, *Educational Game Design Fundamentals*.

news.detik.com *Survei Kemdikbud: Siswa Sulit Pahami Pelajaran Saat Belajar Jarak Jauh*. (Accessed: 5 April 2021).

Perez, D. (2016) *Beginning RPG Maker MV*, *Beginning RPG Maker MV*.

Pressman, R. S. and Maxim, B. R. (2015) *SOFTWARE ENGINEERING: A PRACTITIONER'S APPROACH*.

Rina Hayati (2021) *Pengertian Rating Scale, Ciri, dan Contohnya*. (Accessed: 8 May 2021).

Sari, K., Saputro, S. and Hastuti, B. (2014) 'Pengembangan Game Edukasi Kimia Berbasis Role Playing Game (Rpg) Pada Materi Struktur Atom Sebagai Media Pembelajaran Mandiri Untuk Siswa Kelas X Sma Di Kabupaten Purworejo'.

Seidl, M. (2015) *UML@Classroom: An introduction to object-oriented modeling*.

Setiyani, L. (2019) [Software Engineering] .

Wibawanto, W. (2020) *Game Edukasi RPG (Role Playing Game)*.

Zubaidah, S. dkk. (2017) *ILMU PENGETAHUAN ALAM*. Pusat Kurikulum dan Perbukuan, Balitbang, Kemendikbud.

LAMPIRAN

Listing Program :

```
function Scene_Base() {
    this.initialize.apply(this, arguments);
}

Scene_Base.prototype = Object.create(Stage.prototype);
Scene_Base.prototype.constructor = Scene_Base;



Scene_Base.prototype.initialize = function() {
    Stage.prototype.initialize.call(this);
    this._active = false;
    this._fadeSign = 0;
    this._fadeDuration = 0;
    this._fadeSprite = null;
    this._imageReservationId = Utils.generateRuntimeId();
};

Scene_Base.prototype.attachReservation = function() {
    ImageManager.setDefaultReservationId(this._imageReservationId);
};

Scene_Base.prototype.detachReservation = function() {
    ImageManager.releaseReservation(this._imageReservationId);
};

Scene_Base.prototype.create = function() {
};

Scene_Base.prototype.isActive = function() {
    return this._active;
};

Scene_Base.prototype.isReady = function() {
    return ImageManager.isReady();
};

Scene_Base.prototype.start = function() {
    this._active = true;
};

Scene_Base.prototype.update = function() {
    this.updateFade();
    this.updateChildren();
};



Scene_Base.prototype.stop = function() {
    this._active = false;
};
```

```
Scene_Base.prototype.createWindowLayer = function() {
    var width = Graphics.boxWidth;
    var height = Graphics.boxHeight;
    var x = (Graphics.width - width) / 2;
    var y = (Graphics.height - height) / 2;
    this._windowLayer = new WindowLayer();
    this._windowLayer.move(x, y, width, height);
    this.addChild(this._windowLayer);
};

Scene_Base.prototype.addWindow = function(window) {
    this._windowLayer.addChild(window);
};

Scene_Base.prototype.startFadeIn = function(duration, white) {
    this.createFadeSprite(white);
    this._fadeSign = 1;
    this._fadeDuration = duration || 30;
    this._fadeSprite.opacity = 255;
};

Scene_Base.prototype.startFadeOut = function(duration, white) {
    this.createFadeSprite(white);
    this._fadeSign = -1;
    this._fadeDuration = duration || 30;
    this._fadeSprite.opacity = 0;
};

Scene_Base.prototype.createFadeSprite = function(white) {
    if (!this._fadeSprite) {
        this._fadeSprite = new ScreenSprite();
        this.addChild(this._fadeSprite);
    }
    if (white) {
        this._fadeSprite.setWhite();
    } else {
        this._fadeSprite.setBlack();
    }
};

Scene_Base.prototype.updateFade = function() {
    if (this._fadeDuration > 0) {
        var d = this._fadeDuration;
        if (this._fadeSign > 0) {
            this._fadeSprite.opacity -= this._fadeSprite.opacity / d;
        } else {
            this._fadeSprite.opacity += (255 - this._fadeSprite.opacity) / d;
        }
        this._fadeDuration--;
    }
};

Scene_Base.prototype.updateChildren = function() {
    this.children.forEach(function(child) {
        if (child.update) {
            child.update();
        }
    });
};
```

```
Scene_Base.prototype.popScene = function() {
    SceneManager.pop();
};

Scene_Base.prototype.checkGameover = function() {
    if ($gameParty.isAllDead()) {
        SceneManager.goto(Scene_Gameover);
    }
};

Scene_Base.prototype.fadeOutAll = function() {
    var time = this.slowFadeSpeed() / 60;
    AudioManager.fadeOutBgm(time);
    AudioManager.fadeOutBgs(time);
    AudioManager.fadeOutMe(time);
    this.startFadeOut(this.slowFadeSpeed());
};

Scene_Base.prototype.fadeSpeed = function() {
    return 24;
};

Scene_Base.prototype.slowFadeSpeed = function() {
    return this.fadeSpeed() * 2;
};


function Scene_Boot() {
    this.initialize.apply(this, arguments);
}

Scene_Boot.prototype = Object.create(Scene_Base.prototype);
Scene_Boot.prototype.constructor = Scene_Boot;

Scene_Boot.prototype.initialize = function() {
    Scene_Base.prototype.initialize.call(this);
    this._startDate = Date.now();
};

Scene_Boot.prototype.create = function() {
    Scene_Base.prototype.create.call(this);
    DataManager.loadDatabase();
    ConfigManager.load();
    this.loadSystemWindowImage();
};

Scene_Boot.prototype.loadSystemWindowImage = function() {
    ImageManager.reserveSystem('Window');
};

Scene_Boot.loadSystemImages = function() {
    ImageManager.reserveSystem('IconSet');
    ImageManager.reserveSystem('Balloon');
    ImageManager.reserveSystem('Shadow1');
    ImageManager.reserveSystem('Shadow2');
    ImageManager.reserveSystem('Damage');
    ImageManager.reserveSystem('States');
    ImageManager.reserveSystem('Weapons1');
    ImageManager.reserveSystem('Weapons2');
    ImageManager.reserveSystem('Weapons3');
```

```
    ImageManager.reserveSystem('ButtonSet');
};

Scene_Boot.prototype.isReady = function() {
    if (Scene_Base.prototype.isReady.call(this)) {
        return DataManager.isDatabaseLoaded() && this.isGameFontLoaded();
    } else {
        return false;
    }
};

Scene_Boot.prototype.isGameFontLoaded = function() {
    if (Graphics.isFontLoaded('GameFont')) {
        return true;
    } else if (!Graphics.canUseCssFontLoading()){
        var elapsed = Date.now() - this._startDate;
        if (elapsed >= 60000) {
            throw new Error('Failed to load GameFont');
        }
    }
};

Scene_Boot.prototype.start = function() {
    Scene_Base.prototype.start.call(this);
    SoundManager.preloadImportantSounds();
    if (DataManager.isBattleTest()) {
        DataManager.setupBattleTest();
        SceneManager.goto(Scene_Battle);
    } else if (DataManager.isEventTest()) {
        DataManager.setupEventTest();
        SceneManager.goto(Scene_Map);
    } else {
        this.checkPlayerLocation();
        DataManager.setupNewGame();
        SceneManager.goto(Scene_Title);
        Window_TitleCommand.initCommandPosition();
    }
    this.updateDocumentTitle();
};

Scene_Boot.prototype.updateDocumentTitle = function() {
    document.title = $dataSystem.gameTitle;
};

Scene_Boot.prototype.checkPlayerLocation = function() {
    if ($dataSystem.startMapId === 0) {
        throw new Error('Player\'s starting position is not set');
    }
};


function Scene_Title() {
    this.initialize.apply(this, arguments);
}

Scene_Title.prototype = Object.create(Scene_Base.prototype);
Scene_Title.prototype.constructor = Scene_Title;

Scene_Title.prototype.initialize = function() {
    Scene_Base.prototype.initialize.call(this);
```

```
};

Scene_Title.prototype.create = function() {
    Scene_Base.prototype.create.call(this);
    this.createBackground();
    this.createForeground();
    this.createWindowLayer();
    this.createCommandWindow();
};

Scene_Title.prototype.start = function() {
    Scene_Base.prototype.start.call(this);
    SceneManager.clearStack();
    this.centerSprite(this._backSprite1);
    this.centerSprite(this._backSprite2);
    this.playTitleMusic();
    this.startFadeIn(this.fadeSpeed(), false);
};

Scene_Title.prototype.update = function() {
    if (!this.isBusy()) {
        this._commandWindow.open();
    }
    Scene_Base.prototype.update.call(this);
};

Scene_Title.prototype.isBusy = function() {
    return this._commandWindow.isClosing() || Scene_Base.prototype.isBusy.call(this);
};

Scene_Title.prototype.terminate = function() {
    Scene_Base.prototype.terminate.call(this);
    SceneManager.snapForBackground();
};

Scene_Title.prototype.createBackground = function() {
    this._backSprite1 = new Sprite(ImageManager.loadTitle1($dataSystem.title1Name));
    this._backSprite2 = new Sprite(ImageManager.loadTitle2($dataSystem.title2Name));
    this.addChild(this._backSprite1);
    this.addChild(this._backSprite2);
};

Scene_Title.prototype.createForeground = function() {
    this._gameTitleSprite = new Sprite(new Bitmap(Graphics.width, Graphics.height));
    this.addChild(this._gameTitleSprite);
    if ($dataSystem.optDrawTitle) {
        this.drawGameTitle();
    }
};

Scene_Title.prototype.drawGameTitle = function() {
    var x = 20;
    var y = Graphics.height / 4;
    var maxWidth = Graphics.width - x * 2;
    var text = $dataSystem.gameTitle;
    this._gameTitleSprite.bitmap.outlineColor = 'black';
    this._gameTitleSprite.bitmap.outlineWidth = 8;
    this._gameTitleSprite.bitmap.fontSize = 72;
    this._gameTitleSprite.bitmap.drawText(text, x, y, maxWidth, 48, 'center');
};
```

```
Scene_Title.prototype.centerSprite = function(sprite) {
    sprite.x = Graphics.width / 2;
    sprite.y = Graphics.height / 2;
    sprite.anchor.x = 0.5;
    sprite.anchor.y = 0.5;
};

Scene_Title.prototype.createCommandWindow = function() {
    this._commandWindow = new Window_TitleCommand();
    this._commandWindow.setHandler('newGame',  this.commandNewGame.bind(this));
    this._commandWindow.setHandler('continue', this.commandContinue.bind(this));
    this._commandWindow.setHandler('options',  this.commandOptions.bind(this));
    this.addWindow(this._commandWindow);
};

Scene_Title.prototype.commandNewGame = function() {
    DataManager.setupNewGame();
    this._commandWindow.close();
    this.fadeOutAll();
    SceneManager.goto(Scene_Map);
};

Scene_Title.prototype.commandContinue = function() {
    this._commandWindow.close();
    SceneManager.push(Scene_Load);
};

Scene_Title.prototype.commandOptions = function() {
    this._commandWindow.close();
    SceneManager.push(Scene_Options);
};

Scene_Title.prototype.playTitleMusic = function() {
    AudioManager.playBgm($dataSystem.titleBgm);
    AudioManager.stopBgs();
    AudioManager.stopMe();
};


function Scene_Map() {
    this.initialize.apply(this, arguments);
}

Scene_Map.prototype = Object.create(Scene_Base.prototype);
Scene_Map.prototype.constructor = Scene_Map;

Scene_Map.prototype.initialize = function() {
    Scene_Base.prototype.initialize.call(this);
    this._waitCount = 0;
    this._encounterEffectDuration = 0;
    this._mapLoaded = false;
    this._touchCount = 0;
};

Scene_Map.prototype.create = function() {
    Scene_Base.prototype.create.call(this);
    this._transfer = $gamePlayer.isTransferring();
    var mapId = this._transfer ? $gamePlayer.newMapId() : $gameMap.mapId();
    DataManager.loadMapData(mapId);
```

```
};

Scene_Map.prototype.isReady = function() {
    if (!this._mapLoaded && DataManager.isMapLoaded()) {
        this.onMapLoaded();
        this._mapLoaded = true;
    }
    return this._mapLoaded && Scene_Base.prototype.isReady.call(this);
};

Scene_Map.prototype.onMapLoaded = function() {
    if (this._transfer) {
        $gamePlayer.performTransfer();
    }
    this.createDisplayObjects();
};

Scene_Map.prototype.start = function() {
    Scene_Base.prototype.start.call(this);
    SceneManager.clearStack();
    if (this._transfer) {
        this.fadeInForTransfer();
        this._mapNameWindow.open();
        $gameMap.autoplay();
    } else if (this.needsFadeIn()) {
        this.startFadeIn(this.fadeSpeed(), false);
    }
    this.menuCalling = false;
};

Scene_Map.prototype.update = function() {
    this.updateDestination();
    this.updateMainMultiply();
    if (this.isSceneChangeOk()) {
        this.updateScene();
    } else if (SceneManager.isNextScene(Scene_Battle)) {
        this.updateEncounterEffect();
    }
    this.updateWaitCount();
    Scene_Base.prototype.update.call(this);
};

Scene_Map.prototype.updateMainMultiply = function() {
    this.updateMain();
    if (this.isFastForward()) {
        this.updateMain();
    }
};

Scene_Map.prototype.updateMain = function() {
    var active = this.isActive();
    $gameMap.update(active);
    $gamePlayer.update(active);
    $gameTimer.update(active);
    $gameScreen.update();
};

Scene_Map.prototype.isFastForward = function() {
    return ($gameMap.isEventRunning() && !SceneManager.isSceneChanging() &&
            (Input.isLongPressed('ok') || TouchInput.isLongPressed()));
```

```
};

Scene_Map.prototype.stop = function() {
    Scene_Base.prototype.stop.call(this);
    $gamePlayer.straighten();
    this._mapNameWindow.close();
    if (this.needsSlowFadeOut()) {
        this.startFadeOut(this.slowFadeSpeed(), false);
    } else if (SceneManager.isNextScene(Scene_Map)) {
        this.fadeOutForTransfer();
    } else if (SceneManager.isNextScene(Scene_Battle)) {
        this.launchBattle();
    }
};

Scene_Map.prototype.isBusy = function() {
    return ((this._messageWindow && this._messageWindow.isClosing()) ||
            this._waitCount > 0 || this._encounterEffectDuration > 0 ||
            Scene_Base.prototype.isBusy.call(this));
};

Scene_Map.prototype.terminate = function() {
    Scene_Base.prototype.terminate.call(this);
    if (!SceneManager.isNextScene(Scene_Battle)) {
        this._spriteset.update();
        this._mapNameWindow.hide();
        SceneManager.snapForBackground();
    } else {
        ImageManager.clearRequest();
    }

    if (SceneManager.isNextScene(Scene_Map)) {
        ImageManager.clearRequest();
    }

    $gameScreen.clearZoom();

    this.removeChild(this._fadeSprite);
    this.removeChild(this._mapNameWindow);
    this.removeChild(this._windowLayer);
    this.removeChild(this._spriteset);
};

Scene_Map.prototype.needsFadeIn = function() {
    return (SceneManager.isPreviousScene(Scene_Battle) ||
            SceneManager.isPreviousScene(Scene_Load));
};

Scene_Map.prototype.needsSlowFadeOut = function() {
    return (SceneManager.isNextScene(Scene_Title) ||
            SceneManager.isNextScene(Scene_Gameover));
};

Scene_Map.prototype.updateWaitCount = function() {
    if (this._waitCount > 0) {
        this._waitCount--;
        return true;
    }
    return false;
};
```

```
Scene_Map.prototype.updateDestination = function() {
    if (this.isMapTouchOk()) {
        this.processMapTouch();
    } else {
        $gameTemp.clearDestination();
        this._touchCount = 0;
    }
};

Scene_Map.prototype.isMapTouchOk = function() {
    return this.isActive() && $gamePlayer.canMove();
};

Scene_Map.prototype.processMapTouch = function() {
    if (TouchInput.isTriggered() || this._touchCount > 0) {
        if (TouchInput.isPressed()) {
            if (this._touchCount === 0 || this._touchCount >= 15) {
                var x = $gameMap.canvasToMapX(TouchInput.x);
                var y = $gameMap.canvasToMapY(TouchInput.y);
                $gameTemp.setDestination(x, y);
            }
            this._touchCount++;
        } else {
            this._touchCount = 0;
        }
    }
};

Scene_Map.prototype.isSceneChangeOk = function() {
    return this.isActive() && !$gameMessage.isBusy();
};

Scene_Map.prototype.updateScene = function() {
    this.checkGameover();
    if (!SceneManager.isSceneChanging()) {
        this.updateTransferPlayer();
    }
    if (!SceneManager.isSceneChanging()) {
        this.updateEncounter();
    }
    if (!SceneManager.isSceneChanging()) {
        this.updateCallMenu();
    }
    if (!SceneManager.isSceneChanging()) {
        this.updateCallDebug();
    }
};

Scene_Map.prototype.createDisplayObjects = function() {
    this.createSpriteset();
    this.createMapNameWindow();
    this.createWindowLayer();
    this.createAllWindows();
};

Scene_Map.prototype.createSpriteset = function() {
    this._spriteset = new Spriteset_Map();
    this.addChild(this._spriteset);
};
```

```
Scene_Map.prototype.createAllWindows = function() {
    this.createMessageWindow();
    this.createScrollTextWindow();
};

Scene_Map.prototype.createMapNameWindow = function() {
    this._mapNameWindow = new Window_MapName();
    this.addChild(this._mapNameWindow);
};

Scene_Map.prototype.createMessageWindow = function() {
    this._messageWindow = new Window_Message();
    this.addWindow(this._messageWindow);
    this._messageWindow.subWindows().forEach(function(window) {
        this.addWindow(window);
    }, this);
};

Scene_Map.prototype.createScrollTextWindow = function() {
    this._scrollTextWindow = new Window_ScrollText();
    this.addWindow(this._scrollTextWindow);
};

Scene_Map.prototype.updateTransferPlayer = function() {
    if ($gamePlayer.isTransferring()) {
        SceneManager.goto(Scene_Map);
    }
};

Scene_Map.prototype.updateEncounter = function() {
   if ($gamePlayer.executeEncounter()) {
       SceneManager.push(Scene_Battle);
   }
};

Scene_Map.prototype.updateCallMenu = function() {
    if (this.isMenuEnabled()) {
        if (this.isMenuCalled()) {
            this.menuCalling = true;
        }
        if (this.menuCalling && !$gamePlayer.isMoving()) {
            this.callMenu();
        }
    } else {
        this.menuCalling = false;
    }
};

Scene_Map.prototype.isMenuEnabled = function() {
    return $gameSystem.isMenuEnabled() && !$gameMap.isEventRunning();
};

Scene_Map.prototype.isMenuCalled = function() {
    return Input.isTriggered('menu') || TouchInput.isCancelled();
};

Scene_Map.prototype.callMenu = function() {
    SoundManager.playOk();
    SceneManager.push(Scene_Menu);
```

```
    Window_MenuCommand.initCommandPosition();
    $gameTemp.clearDestination();
    this._mapNameWindow.hide();
    this._waitCount = 2;
};

Scene_Map.prototype.updateCallDebug = function() {
    if (this.isDebugCalled()) {
        SceneManager.push(Scene_Debug);
    }
};

Scene_Map.prototype.isDebugCalled = function() {
    return Input.isTriggered('debug') && $gameTemp.isPlaytest();
};

Scene_Map.prototype.fadeInForTransfer = function() {
    var fadeType = $gamePlayer.fadeType();
    switch (fadeType) {
    case 0: case 1:
        this.startFadeIn(this.fadeSpeed(), fadeType === 1);
        break;
    }
};

Scene_Map.prototype.fadeOutForTransfer = function() {
    var fadeType = $gamePlayer.fadeType();
    switch (fadeType) {
    case 0: case 1:
        this.startFadeOut(this.fadeSpeed(), fadeType === 1);
        break;
    }
};

Scene_Map.prototype.launchBattle = function() {
    BattleManager.saveBgmAndBgs();
    this.stopAudioOnBattleStart();
    SoundManager.playBattleStart();
    this.startEncounterEffect();
    this._mapNameWindow.hide();
};

Scene_Map.prototype.stopAudioOnBattleStart = function() {
    if (!AudioManager.isCurrentBgm($gameSystem.battleBgm())) {
        AudioManager.stopBgm();
    }
    AudioManager.stopBgs();
    AudioManager.stopMe();
    AudioManager.stopSe();
};

Scene_Map.prototype.startEncounterEffect = function() {
    this._spriteset.hideCharacters();
    this._encounterEffectDuration = this.encounterEffectSpeed();
};

Scene_Map.prototype.updateEncounterEffect = function() {
    if (this._encounterEffectDuration > 0) {
        this._encounterEffectDuration--;
        var speed = this.encounterEffectSpeed();
```

```
        var n = speed - this._encounterEffectDuration;
        var p = n / speed;
        var q = ((p - 1) * 20 * p + 5) * p + 1;
        var zoomX = $gamePlayer.screenX();
        var zoomY = $gamePlayer.screenY() - 24;
        if (n === 2) {
            $gameScreen.setZoom(zoomX, zoomY, 1);
            this.snapForBattleBackground();
            this.startFlashForEncounter(speed / 2);
        }
        $gameScreen.setZoom(zoomX, zoomY, q);
        if (n === Math.floor(speed / 6)) {
            this.startFlashForEncounter(speed / 2);
        }
        if (n === Math.floor(speed / 2)) {
            BattleManager.playBattleBgm();
            this.startFadeOut(this.fadeSpeed());
        }
    }
};

Scene_Map.prototype.snapForBattleBackground = function() {
    this._windowLayer.visible = false;
    SceneManager.snapForBackground();
    this._windowLayer.visible = true;
};

Scene_Map.prototype.startFlashForEncounter = function(duration) {
    var color = [255, 255, 255, 255];
    $gameScreen.startFlash(color, duration);
};

Scene_Map.prototype.encounterEffectSpeed = function() {
    return 60;
};


function Scene_MenuBase() {
    this.initialize.apply(this, arguments);
}

Scene_MenuBase.prototype = Object.create(Scene_Base.prototype);
Scene_MenuBase.prototype.constructor = Scene_MenuBase;

Scene_MenuBase.prototype.initialize = function() {
    Scene_Base.prototype.initialize.call(this);
};

Scene_MenuBase.prototype.create = function() {
    Scene_Base.prototype.create.call(this);
    this.createBackground();
    this.updateActor();
    this.createWindowLayer();
};

Scene_MenuBase.prototype.actor = function() {
    return this._actor;
};

Scene_MenuBase.prototype.updateActor = function() {
```

```
    this._actor = $gameParty.menuActor();
};

Scene_MenuBase.prototype.createBackground = function() {
    this._backgroundSprite = new Sprite();
    this._backgroundSprite.bitmap = SceneManager.backgroundBitmap();
    this.addChild(this._backgroundSprite);
};

Scene_MenuBase.prototype.setBackgroundOpacity = function(opacity) {
    this._backgroundSprite.opacity = opacity;
};

Scene_MenuBase.prototype.createHelpWindow = function() {
    this._helpWindow = new Window_Help();
    this.addWindow(this._helpWindow);
};

Scene_MenuBase.prototype.nextActor = function() {
    $gameParty.makeMenuActorNext();
    this.updateActor();
    this.onActorChange();
};

Scene_MenuBase.prototype.previousActor = function() {
    $gameParty.makeMenuActorPrevious();
    this.updateActor();
    this.onActorChange();
};

Scene_MenuBase.prototype.onActorChange = function() {
};


function Scene_Menu() {
    this.initialize.apply(this, arguments);
}

Scene_Menu.prototype = Object.create(Scene_MenuBase.prototype);
Scene_Menu.prototype.constructor = Scene_Menu;

Scene_Menu.prototype.initialize = function() {
    Scene_MenuBase.prototype.initialize.call(this);
};

Scene_Menu.prototype.create = function() {
    Scene_MenuBase.prototype.create.call(this);
    this.createCommandWindow();
    this.createGoldWindow();
    this.createStatusWindow();
};

Scene_Menu.prototype.start = function() {
    Scene_MenuBase.prototype.start.call(this);
    this._statusWindow.refresh();
};

Scene_Menu.prototype.createCommandWindow = function() {
    this._commandWindow = new Window_MenuCommand(0, 0);
    this._commandWindow.setHandler('item',      this.commandItem.bind(this));
```

```
    this._commandWindow.setHandler('skill',     this.commandPersonal.bind(this));
    this._commandWindow.setHandler('equip',     this.commandPersonal.bind(this));
    this._commandWindow.setHandler('status',    this.commandPersonal.bind(this));
    this._commandWindow.setHandler('formation', this.commandFormation.bind(this));
    this._commandWindow.setHandler('options',   this.commandOptions.bind(this));
    this._commandWindow.setHandler('save',      this.commandSave.bind(this));
    this._commandWindow.setHandler('gameEnd',   this.commandGameEnd.bind(this));
    this._commandWindow.setHandler('cancel',    this.popScene.bind(this));
    this.addWindow(this._commandWindow);
};

Scene_Menu.prototype.createGoldWindow = function() {
    this._goldWindow = new Window_Gold(0, 0);
    this._goldWindow.y = Graphics.boxHeight - this._goldWindow.height;
    this.addWindow(this._goldWindow);
};

Scene_Menu.prototype.createStatusWindow = function() {
    this._statusWindow = new Window_MenuStatus(this._commandWindow.width, 0);
    this._statusWindow.reserveFaceImages();
    this.addWindow(this._statusWindow);
};

Scene_Menu.prototype.commandItem = function() {
    SceneManager.push(Scene_Item);
};

Scene_Menu.prototype.commandPersonal = function() {
    this._statusWindow.setFormationMode(false);
    this._statusWindow.selectLast();
    this._statusWindow.activate();
    this._statusWindow.setHandler('ok',     this.onPersonalOk.bind(this));
    this._statusWindow.setHandler('cancel', this.onPersonalCancel.bind(this));
};

Scene_Menu.prototype.commandFormation = function() {
    this._statusWindow.setFormationMode(true);
    this._statusWindow.selectLast();
    this._statusWindow.activate();
    this._statusWindow.setHandler('ok',     this.onFormationOk.bind(this));
    this._statusWindow.setHandler('cancel', this.onFormationCancel.bind(this));
};

Scene_Menu.prototype.commandOptions = function() {
    SceneManager.push(Scene_Options);
};

Scene_Menu.prototype.commandSave = function() {
    SceneManager.push(Scene_Save);
};

Scene_Menu.prototype.commandGameEnd = function() {
    SceneManager.push(Scene_GameEnd);
};

Scene_Menu.prototype.onPersonalOk = function() {
    switch (this._commandWindow.currentSymbol()) {
    case 'skill':
        SceneManager.push(Scene_Skill);
        break;
```

```
    case 'equip':
        SceneManager.push(Scene_Equip);
        break;
    case 'status':
        SceneManager.push(Scene_Status);
        break;
    }
};

Scene_Menu.prototype.onPersonalCancel = function() {
    this._statusWindow.deselect();
    this._commandWindow.activate();
};

Scene_Menu.prototype.onFormationOk = function() {
    var index = this._statusWindow.index();
    var actor = $gameParty.members()[index];
    var pendingIndex = this._statusWindow.pendingIndex();
    if (pendingIndex >= 0) {
        $gameParty.swapOrder(index, pendingIndex);
        this._statusWindow.setPendingIndex(-1);
        this._statusWindow.redrawItem(index);
    } else {
        this._statusWindow.setPendingIndex(index);
    }
    this._statusWindow.activate();
};

Scene_Menu.prototype.onFormationCancel = function() {
    if (this._statusWindow.pendingIndex() >= 0) {
        this._statusWindow.setPendingIndex(-1);
        this._statusWindow.activate();
    } else {
        this._statusWindow.deselect();
        this._commandWindow.activate();
    }
};


function Scene_ItemBase() {
    this.initialize.apply(this, arguments);
}

Scene_ItemBase.prototype = Object.create(Scene_MenuBase.prototype);
Scene_ItemBase.prototype.constructor = Scene_ItemBase;

Scene_ItemBase.prototype.initialize = function() {
    Scene_MenuBase.prototype.initialize.call(this);
};

Scene_ItemBase.prototype.create = function() {
    Scene_MenuBase.prototype.create.call(this);
};

Scene_ItemBase.prototype.createActorWindow = function() {
    this._actorWindow = new Window_MenuActor();
    this._actorWindow.setHandler('ok',     this.onActorOk.bind(this));
    this._actorWindow.setHandler('cancel', this.onActorCancel.bind(this));
    this.addWindow(this._actorWindow);
};
```

```
Scene_ItemBase.prototype.item = function() {
    return this._itemWindow.item();
};

Scene_ItemBase.prototype.user = function() {
    return null;
};

Scene_ItemBase.prototype.isCursorLeft = function() {
    return this._itemWindow.index() % 2 === 0;
};

Scene_ItemBase.prototype.showSubWindow = function(window) {
    window.x = this.isCursorLeft() ? Graphics.boxWidth - window.width : 0;
    window.show();
    window.activate();
};

Scene_ItemBase.prototype.hideSubWindow = function(window) {
    window.hide();
    window.deactivate();
    this.activateItemWindow();
};

Scene_ItemBase.prototype.onActorOk = function() {
    if (this.canUse()) {
        this.useItem();
    } else {
        SoundManager.playBuzzer();
    }
};

Scene_ItemBase.prototype.onActorCancel = function() {
    this.hideSubWindow(this._actorWindow);
};

Scene_ItemBase.prototype.determineItem = function() {
    var action = new Game_Action(this.user());
    var item = this.item();
    action.setItemObject(item);
    if (action.isForFriend()) {
        this.showSubWindow(this._actorWindow);
        this._actorWindow.selectForItem(this.item());
    } else {
        this.useItem();
        this.activateItemWindow();
    }
};

Scene_ItemBase.prototype.useItem = function() {
    this.playSeForItem();
    this.user().useItem(this.item());
    this.applyItem();
    this.checkCommonEvent();
    this.checkGameover();
    this._actorWindow.refresh();
};

Scene_ItemBase.prototype.activateItemWindow = function() {
```

```
    this._itemWindow.refresh();
    this._itemWindow.activate();
};

Scene_ItemBase.prototype.itemTargetActors = function() {
    var action = new Game_Action(this.user());
    action.setItemObject(this.item());
    if (!action.isForFriend()) {
        return [];
    } else if (action.isForAll()) {
        return $gameParty.members();
    } else {
        return [$gameParty.members()[this._actorWindow.index()]];
    }
};

Scene_ItemBase.prototype.canUse = function() {
    return this.user().canUse(this.item()) && this.isItemEffectsValid();
};

Scene_ItemBase.prototype.isItemEffectsValid = function() {
    var action = new Game_Action(this.user());
    action.setItemObject(this.item());
    return this.itemTargetActors().some(function(target) {
        return action.testApply(target);
    }, this);
};

Scene_ItemBase.prototype.applyItem = function() {
    var action = new Game_Action(this.user());
    action.setItemObject(this.item());
    this.itemTargetActors().forEach(function(target) {
        for (var i = 0; i < action.numRepeats(); i++) {
            action.apply(target);
        }
    }, this);
    action.applyGlobal();
};

Scene_ItemBase.prototype.checkCommonEvent = function() {
    if ($gameTemp.isCommonEventReserved()) {
        SceneManager.goto(Scene_Map);
    }
};


function Scene_Item() {
    this.initialize.apply(this, arguments);
}

Scene_Item.prototype = Object.create(Scene_ItemBase.prototype);
Scene_Item.prototype.constructor = Scene_Item;

Scene_Item.prototype.initialize = function() {
    Scene_ItemBase.prototype.initialize.call(this);
};

Scene_Item.prototype.create = function() {
    Scene_ItemBase.prototype.create.call(this);
    this.createHelpWindow();
```

```
    this.createCategoryWindow();
    this.createItemWindow();
    this.createActorWindow();
};

Scene_Item.prototype.createCategoryWindow = function() {
    this._categoryWindow = new Window_ItemCategory();
    this._categoryWindow.setHelpWindow(this._helpWindow);
    this._categoryWindow.y = this._helpWindow.height;
    this._categoryWindow.setHandler('ok',     this.onCategoryOk.bind(this));
    this._categoryWindow.setHandler('cancel', this.popScene.bind(this));
    this.addWindow(this._categoryWindow);
};

Scene_Item.prototype.createItemWindow = function() {
    var wy = this._categoryWindow.y + this._categoryWindow.height;
    var wh = Graphics.boxHeight - wy;
    this._itemWindow = new Window_ItemList(0, wy, Graphics.boxWidth, wh);
    this._itemWindow.setHelpWindow(this._helpWindow);
    this._itemWindow.setHandler('ok',     this.onItemOk.bind(this));
    this._itemWindow.setHandler('cancel', this.onItemCancel.bind(this));
    this.addWindow(this._itemWindow);
    this._categoryWindow.setItemWindow(this._itemWindow);
};

Scene_Item.prototype.user = function() {
    var members = $gameParty.movableMembers();
    var bestActor = members[0];
    var bestPha = 0;
    for (var i = 0; i < members.length; i++) {
        if (members[i].pha > bestPha) {
            bestPha = members[i].pha;
            bestActor = members[i];
        }
    }
    return bestActor;
};

Scene_Item.prototype.onCategoryOk = function() {
    this._itemWindow.activate();
    this._itemWindow.selectLast();
};

Scene_Item.prototype.onItemOk = function() {
    $gameParty.setLastItem(this.item());
    this.determineItem();
};

Scene_Item.prototype.onItemCancel = function() {
    this._itemWindow.deselect();
    this._categoryWindow.activate();
};

Scene_Item.prototype.playSeForItem = function() {
    SoundManager.playUseItem();
};

Scene_Item.prototype.useItem = function() {
    Scene_ItemBase.prototype.useItem.call(this);
    this._itemWindow.redrawCurrentItem();
```

```
};


function Scene_Skill() {
    this.initialize.apply(this, arguments);
}

Scene_Skill.prototype = Object.create(Scene_ItemBase.prototype);
Scene_Skill.prototype.constructor = Scene_Skill;

Scene_Skill.prototype.initialize = function() {
    Scene_ItemBase.prototype.initialize.call(this);
};

Scene_Skill.prototype.create = function() {
    Scene_ItemBase.prototype.create.call(this);
    this.createHelpWindow();
    this.createSkillTypeWindow();
    this.createStatusWindow();
    this.createItemWindow();
    this.createActorWindow();
};

Scene_Skill.prototype.start = function() {
    Scene_ItemBase.prototype.start.call(this);
    this.refreshActor();
};

Scene_Skill.prototype.createSkillTypeWindow = function() {
    var wy = this._helpWindow.height;
    this._skillTypeWindow = new Window_SkillType(0, wy);
    this._skillTypeWindow.setHelpWindow(this._helpWindow);
    this._skillTypeWindow.setHandler('skill',    this.commandSkill.bind(this));
    this._skillTypeWindow.setHandler('cancel',   this.popScene.bind(this));
    this._skillTypeWindow.setHandler('pagedown', this.nextActor.bind(this));
    this._skillTypeWindow.setHandler('pageup',   this.previousActor.bind(this));
    this.addWindow(this._skillTypeWindow);
};

Scene_Skill.prototype.createStatusWindow = function() {
    var wx = this._skillTypeWindow.width;
    var wy = this._helpWindow.height;
    var ww = Graphics.boxWidth - wx;
    var wh = this._skillTypeWindow.height;
    this._statusWindow = new Window_SkillStatus(wx, wy, ww, wh);
    this._statusWindow.reserveFaceImages();
    this.addWindow(this._statusWindow);
};

Scene_Skill.prototype.createItemWindow = function() {
    var wx = 0;
    var wy = this._statusWindow.y + this._statusWindow.height;
    var ww = Graphics.boxWidth;
    var wh = Graphics.boxHeight - wy;
    this._itemWindow = new Window_SkillList(wx, wy, ww, wh);
    this._itemWindow.setHelpWindow(this._helpWindow);
    this._itemWindow.setHandler('ok',     this.onItemOk.bind(this));
    this._itemWindow.setHandler('cancel', this.onItemCancel.bind(this));
    this._skillTypeWindow.setSkillWindow(this._itemWindow);
    this.addWindow(this._itemWindow);
```

```
};

Scene_Skill.prototype.refreshActor = function() {
    var actor = this.actor();
    this._skillTypeWindow.setActor(actor);
    this._statusWindow.setActor(actor);
    this._itemWindow.setActor(actor);
};

Scene_Skill.prototype.user = function() {
    return this.actor();
};

Scene_Skill.prototype.commandSkill = function() {
    this._itemWindow.activate();
    this._itemWindow.selectLast();
};

Scene_Skill.prototype.onItemOk = function() {
    this.actor().setLastMenuSkill(this.item());
    this.determineItem();
};

Scene_Skill.prototype.onItemCancel = function() {
    this._itemWindow.deselect();
    this._skillTypeWindow.activate();
};

Scene_Skill.prototype.playSeForItem = function() {
    SoundManager.playUseSkill();
};

Scene_Skill.prototype.useItem = function() {
    Scene_ItemBase.prototype.useItem.call(this);
    this._statusWindow.refresh();
    this._itemWindow.refresh();
};

Scene_Skill.prototype.onActorChange = function() {
    this.refreshActor();
    this._skillTypeWindow.activate();
};


function Scene_Equip() {
    this.initialize.apply(this, arguments);
}

Scene_Equip.prototype = Object.create(Scene_MenuBase.prototype);
Scene_Equip.prototype.constructor = Scene_Equip;

Scene_Equip.prototype.initialize = function() {
    Scene_MenuBase.prototype.initialize.call(this);
};

Scene_Equip.prototype.create = function() {
    Scene_MenuBase.prototype.create.call(this);
    this.createHelpWindow();
    this.createStatusWindow();
    this.createCommandWindow();
```

```
    this.createSlotWindow();
    this.createItemWindow();
    this.refreshActor();
};

Scene_Equip.prototype.createStatusWindow = function() {
    this._statusWindow = new Window_EquipStatus(0, this._helpWindow.height);
    this.addWindow(this._statusWindow);
};

Scene_Equip.prototype.createCommandWindow = function() {
    var wx = this._statusWindow.width;
    var wy = this._helpWindow.height;
    var ww = Graphics.boxWidth - this._statusWindow.width;
    this._commandWindow = new Window_EquipCommand(wx, wy, ww);
    this._commandWindow.setHelpWindow(this._helpWindow);
    this._commandWindow.setHandler('equip',    this.commandEquip.bind(this));
    this._commandWindow.setHandler('optimize', this.commandOptimize.bind(this));
    this._commandWindow.setHandler('clear',    this.commandClear.bind(this));
    this._commandWindow.setHandler('cancel',   this.popScene.bind(this));
    this._commandWindow.setHandler('pagedown', this.nextActor.bind(this));
    this._commandWindow.setHandler('pageup',   this.previousActor.bind(this));
    this.addWindow(this._commandWindow);
};

Scene_Equip.prototype.createSlotWindow = function() {
    var wx = this._statusWindow.width;
    var wy = this._commandWindow.y + this._commandWindow.height;
    var ww = Graphics.boxWidth - this._statusWindow.width;
    var wh = this._statusWindow.height - this._commandWindow.height;
    this._slotWindow = new Window_EquipSlot(wx, wy, ww, wh);
    this._slotWindow.setHelpWindow(this._helpWindow);
    this._slotWindow.setStatusWindow(this._statusWindow);
    this._slotWindow.setHandler('ok',       this.onSlotOk.bind(this));
    this._slotWindow.setHandler('cancel',   this.onSlotCancel.bind(this));
    this.addWindow(this._slotWindow);
};

Scene_Equip.prototype.createItemWindow = function() {
    var wx = 0;
    var wy = this._statusWindow.y + this._statusWindow.height;
    var ww = Graphics.boxWidth;
    var wh = Graphics.boxHeight - wy;
    this._itemWindow = new Window_EquipItem(wx, wy, ww, wh);
    this._itemWindow.setHelpWindow(this._helpWindow);
    this._itemWindow.setStatusWindow(this._statusWindow);
    this._itemWindow.setHandler('ok',     this.onItemOk.bind(this));
    this._itemWindow.setHandler('cancel', this.onItemCancel.bind(this));
    this._slotWindow.setItemWindow(this._itemWindow);
    this.addWindow(this._itemWindow);
};

Scene_Equip.prototype.refreshActor = function() {
    var actor = this.actor();
    this._statusWindow.setActor(actor);
    this._slotWindow.setActor(actor);
    this._itemWindow.setActor(actor);
};

Scene_Equip.prototype.commandEquip = function() {
```

```
    this._slotWindow.activate();
    this._slotWindow.select(0);
};

Scene_Equip.prototype.commandOptimize = function() {
    SoundManager.playEquip();
    this.actor().optimizeEquipments();
    this._statusWindow.refresh();
    this._slotWindow.refresh();
    this._commandWindow.activate();
};

Scene_Equip.prototype.commandClear = function() {
    SoundManager.playEquip();
    this.actor().clearEquipments();
    this._statusWindow.refresh();
    this._slotWindow.refresh();
    this._commandWindow.activate();
};

Scene_Equip.prototype.onSlotOk = function() {
    this._itemWindow.activate();
    this._itemWindow.select(0);
};

Scene_Equip.prototype.onSlotCancel = function() {
    this._slotWindow.deselect();
    this._commandWindow.activate();
};

Scene_Equip.prototype.onItemOk = function() {
    SoundManager.playEquip();
    this.actor().changeEquip(this._slotWindow.index(), this._itemWindow.item());
    this._slotWindow.activate();
    this._slotWindow.refresh();
    this._itemWindow.deselect();
    this._itemWindow.refresh();
    this._statusWindow.refresh();
};

Scene_Equip.prototype.onItemCancel = function() {
    this._slotWindow.activate();
    this._itemWindow.deselect();
};

Scene_Equip.prototype.onActorChange = function() {
    this.refreshActor();
    this._commandWindow.activate();
};


function Scene_Status() {
    this.initialize.apply(this, arguments);
}

Scene_Status.prototype = Object.create(Scene_MenuBase.prototype);
Scene_Status.prototype.constructor = Scene_Status;

Scene_Status.prototype.initialize = function() {
    Scene_MenuBase.prototype.initialize.call(this);
```

```
};

Scene_Status.prototype.create = function() {
    Scene_MenuBase.prototype.create.call(this);
    this._statusWindow = new Window_Status();
    this._statusWindow.setHandler('cancel',   this.popScene.bind(this));
    this._statusWindow.setHandler('pagedown', this.nextActor.bind(this));
    this._statusWindow.setHandler('pageup',   this.previousActor.bind(this));
    this._statusWindow.reserveFaceImages();
    this.addWindow(this._statusWindow);
};

Scene_Status.prototype.start = function() {
    Scene_MenuBase.prototype.start.call(this);
    this.refreshActor();
};

Scene_Status.prototype.refreshActor = function() {
    var actor = this.actor();
    this._statusWindow.setActor(actor);
};

Scene_Status.prototype.onActorChange = function() {
    this.refreshActor();
    this._statusWindow.activate();
};


function Scene_Options() {
    this.initialize.apply(this, arguments);
}

Scene_Options.prototype = Object.create(Scene_MenuBase.prototype);
Scene_Options.prototype.constructor = Scene_Options;

Scene_Options.prototype.initialize = function() {
    Scene_MenuBase.prototype.initialize.call(this);
};

Scene_Options.prototype.create = function() {
    Scene_MenuBase.prototype.create.call(this);
    this.createOptionsWindow();
};

Scene_Options.prototype.terminate = function() {
    Scene_MenuBase.prototype.terminate.call(this);
    ConfigManager.save();
};

Scene_Options.prototype.createOptionsWindow = function() {
    this._optionsWindow = new Window_Options();
    this._optionsWindow.setHandler('cancel', this.popScene.bind(this));
    this.addWindow(this._optionsWindow);
};


function Scene_File() {
    this.initialize.apply(this, arguments);
}
```

```
Scene_File.prototype = Object.create(Scene_MenuBase.prototype);
Scene_File.prototype.constructor = Scene_File;

Scene_File.prototype.initialize = function() {
    Scene_MenuBase.prototype.initialize.call(this);
};

Scene_File.prototype.create = function() {
    Scene_MenuBase.prototype.create.call(this);
    DataManager.loadAllSavefileImages();
    this.createHelpWindow();
    this.createListWindow();
};

Scene_File.prototype.start = function() {
    Scene_MenuBase.prototype.start.call(this);
    this._listWindow.refresh();
};

Scene_File.prototype.savefileId = function() {
    return this._listWindow.index() + 1;
};

Scene_File.prototype.createHelpWindow = function() {
    this._helpWindow = new Window_Help(1);
    this._helpWindow.setText(this.helpWindowText());
    this.addWindow(this._helpWindow);
};

Scene_File.prototype.createListWindow = function() {
    var x = 0;
    var y = this._helpWindow.height;
    var width = Graphics.boxWidth;
    var height = Graphics.boxHeight - y;
    this._listWindow = new Window_SavefileList(x, y, width, height);
    this._listWindow.setHandler('ok',     this.onSavefileOk.bind(this));
    this._listWindow.setHandler('cancel', this.popScene.bind(this));
    this._listWindow.select(this.firstSavefileIndex());
    this._listWindow.setTopRow(this.firstSavefileIndex() - 2);
    this._listWindow.setMode(this.mode());
    this._listWindow.refresh();
    this.addWindow(this._listWindow);
};

Scene_File.prototype.mode = function() {
    return null;
};

Scene_File.prototype.activateListWindow = function() {
    this._listWindow.activate();
};

Scene_File.prototype.helpWindowText = function() {
    return '';
};

Scene_File.prototype.firstSavefileIndex = function() {
    return 0;
};
```

```
Scene_File.prototype.onSavefileOk = function() {
};

function Scene_Save() {
    this.initialize.apply(this, arguments);
}

Scene_Save.prototype = Object.create(Scene_File.prototype);
Scene_Save.prototype.constructor = Scene_Save;

Scene_Save.prototype.initialize = function() {
    Scene_File.prototype.initialize.call(this);
};

Scene_Save.prototype.mode = function() {
    return 'save';
};

Scene_Save.prototype.helpWindowText = function() {
    return TextManager.saveMessage;
};

Scene_Save.prototype.firstSavefileIndex = function() {
    return DataManager.lastAccessedSavefileId() - 1;
};

Scene_Save.prototype.onSavefileOk = function() {
    Scene_File.prototype.onSavefileOk.call(this);
    $gameSystem.onBeforeSave();
    if (DataManager.saveGame(this.savefileId())) {
        this.onSaveSuccess();
    } else {
        this.onSaveFailure();
    }
};

Scene_Save.prototype.onSaveSuccess = function() {
    SoundManager.playSave();
            StorageManager.cleanBackup(this.savefileId());
    this.popScene();
};

Scene_Save.prototype.onSaveFailure = function() {
    SoundManager.playBuzzer();
    this.activateListWindow();
};

function Scene_Load() {
    this.initialize.apply(this, arguments);
}

Scene_Load.prototype = Object.create(Scene_File.prototype);
Scene_Load.prototype.constructor = Scene_Load;

Scene_Load.prototype.initialize = function() {
    Scene_File.prototype.initialize.call(this);
    this._loadSuccess = false;
};

Scene_Load.prototype.terminate = function() {
```

```
    Scene_File.prototype.terminate.call(this);
    if (this._loadSuccess) {
        $gameSystem.onAfterLoad();
    }
};

Scene_Load.prototype.mode = function() {
    return 'load';
};

Scene_Load.prototype.helpWindowText = function() {
    return TextManager.loadMessage;
};

Scene_Load.prototype.firstSavefileIndex = function() {
    return DataManager.latestSavefileId() - 1;
};

Scene_Load.prototype.onSavefileOk = function() {
    Scene_File.prototype.onSavefileOk.call(this);
    if (DataManager.loadGame(this.savefileId())) {
        this.onLoadSuccess();
    } else {
        this.onLoadFailure();
    }
};

Scene_Load.prototype.onLoadSuccess = function() {
    SoundManager.playLoad();
    this.fadeOutAll();
    this.reloadMapIfUpdated();
    SceneManager.goto(Scene_Map);
    this._loadSuccess = true;
};

Scene_Load.prototype.onLoadFailure = function() {
    SoundManager.playBuzzer();
    this.activateListWindow();
};

Scene_Load.prototype.reloadMapIfUpdated = function() {
    if ($gameSystem.versionId() !== $dataSystem.versionId) {
        $gamePlayer.reserveTransfer($gameMap.mapId(), $gamePlayer.x, $gamePlayer.y);
        $gamePlayer.requestMapReload();
    }
};


function Scene_GameEnd() {
    this.initialize.apply(this, arguments);
}

Scene_GameEnd.prototype = Object.create(Scene_MenuBase.prototype);
Scene_GameEnd.prototype.constructor = Scene_GameEnd;

Scene_GameEnd.prototype.initialize = function() {
    Scene_MenuBase.prototype.initialize.call(this);
};

Scene_GameEnd.prototype.create = function() {
```

```
    Scene_MenuBase.prototype.create.call(this);
    this.createCommandWindow();
};

Scene_GameEnd.prototype.stop = function() {
    Scene_MenuBase.prototype.stop.call(this);
    this._commandWindow.close();
};

Scene_GameEnd.prototype.createBackground = function() {
    Scene_MenuBase.prototype.createBackground.call(this);
    this.setBackgroundOpacity(128);
};

Scene_GameEnd.prototype.createCommandWindow = function() {
    this._commandWindow = new Window_GameEnd();
    this._commandWindow.setHandler('toTitle',  this.commandToTitle.bind(this));
    this._commandWindow.setHandler('cancel',   this.popScene.bind(this));
    this.addWindow(this._commandWindow);
};

Scene_GameEnd.prototype.commandToTitle = function() {
    this.fadeOutAll();
    SceneManager.goto(Scene_Title);
};

function Scene_Shop() {
    this.initialize.apply(this, arguments);
}

Scene_Shop.prototype = Object.create(Scene_MenuBase.prototype);
Scene_Shop.prototype.constructor = Scene_Shop;

Scene_Shop.prototype.initialize = function() {
    Scene_MenuBase.prototype.initialize.call(this);
};

Scene_Shop.prototype.prepare = function(goods, purchaseOnly) {
    this._goods = goods;
    this._purchaseOnly = purchaseOnly;
    this._item = null;
};

Scene_Shop.prototype.create = function() {
    Scene_MenuBase.prototype.create.call(this);
    this.createHelpWindow();
    this.createGoldWindow();
    this.createCommandWindow();
    this.createDummyWindow();
    this.createNumberWindow();
    this.createStatusWindow();
    this.createBuyWindow();
    this.createCategoryWindow();
    this.createSellWindow();
};

Scene_Shop.prototype.createGoldWindow = function() {
    this._goldWindow = new Window_Gold(0, this._helpWindow.height);
    this._goldWindow.x = Graphics.boxWidth - this._goldWindow.width;
    this.addWindow(this._goldWindow);
```

```
};

Scene_Shop.prototype.createCommandWindow = function() {
    this._commandWindow = new Window_ShopCommand(this._goldWindow.x, this._purchaseOnly);
    this._commandWindow.y = this._helpWindow.height;
    this._commandWindow.setHandler('buy',    this.commandBuy.bind(this));
    this._commandWindow.setHandler('sell',   this.commandSell.bind(this));
    this._commandWindow.setHandler('cancel', this.popScene.bind(this));
    this.addWindow(this._commandWindow);
};

Scene_Shop.prototype.createDummyWindow = function() {
    var wy = this._commandWindow.y + this._commandWindow.height;
    var wh = Graphics.boxHeight - wy;
    this._dummyWindow = new Window_Base(0, wy, Graphics.boxWidth, wh);
    this.addWindow(this._dummyWindow);
};

Scene_Shop.prototype.createNumberWindow = function() {
    var wy = this._dummyWindow.y;
    var wh = this._dummyWindow.height;
    this._numberWindow = new Window_ShopNumber(0, wy, wh);
    this._numberWindow.hide();
    this._numberWindow.setHandler('ok',     this.onNumberOk.bind(this));
    this._numberWindow.setHandler('cancel', this.onNumberCancel.bind(this));
    this.addWindow(this._numberWindow);
};

Scene_Shop.prototype.createStatusWindow = function() {
    var wx = this._numberWindow.width;
    var wy = this._dummyWindow.y;
    var ww = Graphics.boxWidth - wx;
    var wh = this._dummyWindow.height;
    this._statusWindow = new Window_ShopStatus(wx, wy, ww, wh);
    this._statusWindow.hide();
    this.addWindow(this._statusWindow);
};

Scene_Shop.prototype.createBuyWindow = function() {
    var wy = this._dummyWindow.y;
    var wh = this._dummyWindow.height;
    this._buyWindow = new Window_ShopBuy(0, wy, wh, this._goods);
    this._buyWindow.setHelpWindow(this._helpWindow);
    this._buyWindow.setStatusWindow(this._statusWindow);
    this._buyWindow.hide();
    this._buyWindow.setHandler('ok',     this.onBuyOk.bind(this));
    this._buyWindow.setHandler('cancel', this.onBuyCancel.bind(this));
    this.addWindow(this._buyWindow);
};

Scene_Shop.prototype.createCategoryWindow = function() {
    this._categoryWindow = new Window_ItemCategory();
    this._categoryWindow.setHelpWindow(this._helpWindow);
    this._categoryWindow.y = this._dummyWindow.y;
    this._categoryWindow.hide();
    this._categoryWindow.deactivate();
    this._categoryWindow.setHandler('ok',     this.onCategoryOk.bind(this));
    this._categoryWindow.setHandler('cancel', this.onCategoryCancel.bind(this));
    this.addWindow(this._categoryWindow);
};
```

```
Scene_Shop.prototype.createSellWindow = function() {
    var wy = this._categoryWindow.y + this._categoryWindow.height;
    var wh = Graphics.boxHeight - wy;
    this._sellWindow = new Window_ShopSell(0, wy, Graphics.boxWidth, wh);
    this._sellWindow.setHelpWindow(this._helpWindow);
    this._sellWindow.hide();
    this._sellWindow.setHandler('ok',     this.onSellOk.bind(this));
    this._sellWindow.setHandler('cancel', this.onSellCancel.bind(this));
    this._categoryWindow.setItemWindow(this._sellWindow);
    this.addWindow(this._sellWindow);
};

Scene_Shop.prototype.activateBuyWindow = function() {
    this._buyWindow.setMoney(this.money());
    this._buyWindow.show();
    this._buyWindow.activate();
    this._statusWindow.show();
};

Scene_Shop.prototype.activateSellWindow = function() {
    this._categoryWindow.show();
    this._sellWindow.refresh();
    this._sellWindow.show();
    this._sellWindow.activate();
    this._statusWindow.hide();
};

Scene_Shop.prototype.commandBuy = function() {
    this._dummyWindow.hide();
    this.activateBuyWindow();
};

Scene_Shop.prototype.commandSell = function() {
    this._dummyWindow.hide();
    this._categoryWindow.show();
    this._categoryWindow.activate();
    this._sellWindow.show();
    this._sellWindow.deselect();
    this._sellWindow.refresh();
};

Scene_Shop.prototype.onBuyOk = function() {
    this._item = this._buyWindow.item();
    this._buyWindow.hide();
    this._numberWindow.setup(this._item, this.maxBuy(), this.buyingPrice());
    this._numberWindow.setCurrencyUnit(this.currencyUnit());
    this._numberWindow.show();
    this._numberWindow.activate();
};

Scene_Shop.prototype.onBuyCancel = function() {
    this._commandWindow.activate();
    this._dummyWindow.show();
    this._buyWindow.hide();
    this._statusWindow.hide();
    this._statusWindow.setItem(null);
    this._helpWindow.clear();
};
```

```
Scene_Shop.prototype.onCategoryOk = function() {
    this.activateSellWindow();
    this._sellWindow.select(0);
};

Scene_Shop.prototype.onCategoryCancel = function() {
    this._commandWindow.activate();
    this._dummyWindow.show();
    this._categoryWindow.hide();
    this._sellWindow.hide();
};

Scene_Shop.prototype.onSellOk = function() {
    this._item = this._sellWindow.item();
    this._categoryWindow.hide();
    this._sellWindow.hide();
    this._numberWindow.setup(this._item, this.maxSell(), this.sellingPrice());
    this._numberWindow.setCurrencyUnit(this.currencyUnit());
    this._numberWindow.show();
    this._numberWindow.activate();
    this._statusWindow.setItem(this._item);
    this._statusWindow.show();
};

Scene_Shop.prototype.onSellCancel = function() {
    this._sellWindow.deselect();
    this._categoryWindow.activate();
    this._statusWindow.setItem(null);
    this._helpWindow.clear();
};

Scene_Shop.prototype.onNumberOk = function() {
    SoundManager.playShop();
    switch (this._commandWindow.currentSymbol()) {
    case 'buy':
        this.doBuy(this._numberWindow.number());
        break;
    case 'sell':
        this.doSell(this._numberWindow.number());
        break;
    }
    this.endNumberInput();
    this._goldWindow.refresh();
    this._statusWindow.refresh();
};

Scene_Shop.prototype.onNumberCancel = function() {
    SoundManager.playCancel();
    this.endNumberInput();
};

Scene_Shop.prototype.doBuy = function(number) {
    $gameParty.loseGold(number * this.buyingPrice());
    $gameParty.gainItem(this._item, number);
};

Scene_Shop.prototype.doSell = function(number) {
    $gameParty.gainGold(number * this.sellingPrice());
    $gameParty.loseItem(this._item, number);
};
```

```
Scene_Shop.prototype.endNumberInput = function() {
    this._numberWindow.hide();
    switch (this._commandWindow.currentSymbol()) {
    case 'buy':
        this.activateBuyWindow();
        break;
    case 'sell':
        this.activateSellWindow();
        break;
    }
};

Scene_Shop.prototype.maxBuy = function() {
    var max = $gameParty.maxItems(this._item) - $gameParty.numItems(this._item);
    var price = this.buyingPrice();
    if (price > 0) {
        return Math.min(max, Math.floor(this.money() / price));
    } else {
        return max;
    }
};

Scene_Shop.prototype.maxSell = function() {
    return $gameParty.numItems(this._item);
};

Scene_Shop.prototype.money = function() {
    return this._goldWindow.value();
};

Scene_Shop.prototype.currencyUnit = function() {
    return this._goldWindow.currencyUnit();
};

Scene_Shop.prototype.buyingPrice = function() {
    return this._buyWindow.price(this._item);
};

Scene_Shop.prototype.sellingPrice = function() {
    return Math.floor(this._item.price / 2);
};


function Scene_Name() {
    this.initialize.apply(this, arguments);
}

Scene_Name.prototype = Object.create(Scene_MenuBase.prototype);
Scene_Name.prototype.constructor = Scene_Name;

Scene_Name.prototype.initialize = function() {
    Scene_MenuBase.prototype.initialize.call(this);
};

Scene_Name.prototype.prepare = function(actorId, maxLength) {
    this._actorId = actorId;
    this._maxLength = maxLength;
};
```

```
Scene_Name.prototype.create = function() {
    Scene_MenuBase.prototype.create.call(this);
    this._actor = $gameActors.actor(this._actorId);
    this.createEditWindow();
    this.createInputWindow();
};

Scene_Name.prototype.start = function() {
    Scene_MenuBase.prototype.start.call(this);
    this._editWindow.refresh();
};

Scene_Name.prototype.createEditWindow = function() {
    this._editWindow = new Window_NameEdit(this._actor, this._maxLength);
    this.addWindow(this._editWindow);
};

Scene_Name.prototype.createInputWindow = function() {
    this._inputWindow = new Window_NameInput(this._editWindow);
    this._inputWindow.setHandler('ok', this.onInputOk.bind(this));
    this.addWindow(this._inputWindow);
};

Scene_Name.prototype.onInputOk = function() {
    this._actor.setName(this._editWindow.name());
    this.popScene();
};


function Scene_Debug() {
    this.initialize.apply(this, arguments);
}

Scene_Debug.prototype = Object.create(Scene_MenuBase.prototype);
Scene_Debug.prototype.constructor = Scene_Debug;

Scene_Debug.prototype.initialize = function() {
    Scene_MenuBase.prototype.initialize.call(this);
};

Scene_Debug.prototype.create = function() {
    Scene_MenuBase.prototype.create.call(this);
    this.createRangeWindow();
    this.createEditWindow();
    this.createDebugHelpWindow();
};

Scene_Debug.prototype.createRangeWindow = function() {
    this._rangeWindow = new Window_DebugRange(0, 0);
    this._rangeWindow.setHandler('ok',     this.onRangeOk.bind(this));
    this._rangeWindow.setHandler('cancel', this.popScene.bind(this));
    this.addWindow(this._rangeWindow);
};

Scene_Debug.prototype.createEditWindow = function() {
    var wx = this._rangeWindow.width;
    var ww = Graphics.boxWidth - wx;
    this._editWindow = new Window_DebugEdit(wx, 0, ww);
    this._editWindow.setHandler('cancel', this.onEditCancel.bind(this));
    this._rangeWindow.setEditWindow(this._editWindow);
```

```
    this.addWindow(this._editWindow);
};

Scene_Debug.prototype.createDebugHelpWindow = function() {
    var wx = this._editWindow.x;
    var wy = this._editWindow.height;
    var ww = this._editWindow.width;
    var wh = Graphics.boxHeight - wy;
    this._debugHelpWindow = new Window_Base(wx, wy, ww, wh);
    this.addWindow(this._debugHelpWindow);
};

Scene_Debug.prototype.onRangeOk = function() {
    this._editWindow.activate();
    this._editWindow.select(0);
    this.refreshHelpWindow();
};

Scene_Debug.prototype.onEditCancel = function() {
    this._rangeWindow.activate();
    this._editWindow.deselect();
    this.refreshHelpWindow();
};

Scene_Debug.prototype.refreshHelpWindow = function() {
    this._debugHelpWindow.contents.clear();
    if (this._editWindow.active) {
        this._debugHelpWindow.drawTextEx(this.helpText(), 4, 0);
    }
};

Scene_Debug.prototype.helpText = function() {
    if (this._rangeWindow.mode() === 'switch') {
        return 'Enter : ON / OFF';
    } else {
        return ('Left     :  -1\n' +
                'Right    :  +1\n' +
                'Pageup   : -10\n' +
                'Pagedown : +10');
    }
};

function Scene_Battle() {
    this.initialize.apply(this, arguments);
}

Scene_Battle.prototype = Object.create(Scene_Base.prototype);
Scene_Battle.prototype.constructor = Scene_Battle;

Scene_Battle.prototype.initialize = function() {
    Scene_Base.prototype.initialize.call(this);
};

Scene_Battle.prototype.create = function() {
    Scene_Base.prototype.create.call(this);
    this.createDisplayObjects();
};

Scene_Battle.prototype.start = function() {
    Scene_Base.prototype.start.call(this);
```

```
    this.startFadeIn(this.fadeSpeed(), false);
    BattleManager.playBattleBgm();
    BattleManager.startBattle();
};

Scene_Battle.prototype.update = function() {
    var active = this.isActive();
    $gameTimer.update(active);
    $gameScreen.update();
    this.updateStatusWindow();
    this.updateWindowPositions();
    if (active && !this.isBusy()) {
        this.updateBattleProcess();
    }
    Scene_Base.prototype.update.call(this);
};

Scene_Battle.prototype.updateBattleProcess = function() {
    if (!this.isAnyInputWindowActive() || BattleManager.isAborting() ||
            BattleManager.isBattleEnd()) {
        BattleManager.update();
        this.changeInputWindow();
    }
};

Scene_Battle.prototype.isAnyInputWindowActive = function() {
    return (this._partyCommandWindow.active ||
            this._actorCommandWindow.active ||
            this._skillWindow.active ||
            this._itemWindow.active ||
            this._actorWindow.active ||
            this._enemyWindow.active);
};

Scene_Battle.prototype.changeInputWindow = function() {
    if (BattleManager.isInputting()) {
        if (BattleManager.actor()) {
            this.startActorCommandSelection();
        } else {
            this.startPartyCommandSelection();
        }
    } else {
        this.endCommandSelection();
    }
};

Scene_Battle.prototype.stop = function() {
    Scene_Base.prototype.stop.call(this);
    if (this.needsSlowFadeOut()) {
        this.startFadeOut(this.slowFadeSpeed(), false);
    } else {
        this.startFadeOut(this.fadeSpeed(), false);
    }
    this._statusWindow.close();
    this._partyCommandWindow.close();
    this._actorCommandWindow.close();
};

Scene_Battle.prototype.terminate = function() {
    Scene_Base.prototype.terminate.call(this);
```

```
    $gameParty.onBattleEnd();
    $gameTroop.onBattleEnd();
    AudioManager.stopMe();

    ImageManager.clearRequest();
};

Scene_Battle.prototype.needsSlowFadeOut = function() {
    return (SceneManager.isNextScene(Scene_Title) ||
            SceneManager.isNextScene(Scene_Gameover));
};

Scene_Battle.prototype.updateStatusWindow = function() {
    if ($gameMessage.isBusy()) {
        this._statusWindow.close();
        this._partyCommandWindow.close();
        this._actorCommandWindow.close();
    } else if (this.isActive() && !this._messageWindow.isClosing()) {
        this._statusWindow.open();
    }
};

Scene_Battle.prototype.updateWindowPositions = function() {
    var statusX = 0;
    if (BattleManager.isInputting()) {
        statusX = this._partyCommandWindow.width;
    } else {
        statusX = this._partyCommandWindow.width / 2;
    }
    if (this._statusWindow.x < statusX) {
        this._statusWindow.x += 16;
        if (this._statusWindow.x > statusX) {
            this._statusWindow.x = statusX;
        }
    }
    if (this._statusWindow.x > statusX) {
        this._statusWindow.x -= 16;
        if (this._statusWindow.x < statusX) {
            this._statusWindow.x = statusX;
        }
    }
};

Scene_Battle.prototype.createDisplayObjects = function() {
    this.createSpriteset();
    this.createWindowLayer();
    this.createAllWindows();
    BattleManager.setLogWindow(this._logWindow);
    BattleManager.setStatusWindow(this._statusWindow);
    BattleManager.setSpriteset(this._spriteset);
    this._logWindow.setSpriteset(this._spriteset);
};

Scene_Battle.prototype.createSpriteset = function() {
    this._spriteset = new Spriteset_Battle();
    this.addChild(this._spriteset);
};

Scene_Battle.prototype.createAllWindows = function() {
    this.createLogWindow();
```

```
    this.createStatusWindow();
    this.createPartyCommandWindow();
    this.createActorCommandWindow();
    this.createHelpWindow();
    this.createSkillWindow();
    this.createItemWindow();
    this.createActorWindow();
    this.createEnemyWindow();
    this.createMessageWindow();
    this.createScrollTextWindow();
};

Scene_Battle.prototype.createLogWindow = function() {
    this._logWindow = new Window_BattleLog();
    this.addWindow(this._logWindow);
};

Scene_Battle.prototype.createStatusWindow = function() {
    this._statusWindow = new Window_BattleStatus();
    this.addWindow(this._statusWindow);
};

Scene_Battle.prototype.createPartyCommandWindow = function() {
    this._partyCommandWindow = new Window_PartyCommand();
    this._partyCommandWindow.setHandler('fight',  this.commandFight.bind(this));
    this._partyCommandWindow.setHandler('escape', this.commandEscape.bind(this));
    this._partyCommandWindow.deselect();
    this.addWindow(this._partyCommandWindow);
};

Scene_Battle.prototype.createActorCommandWindow = function() {
    this._actorCommandWindow = new Window_ActorCommand();
    this._actorCommandWindow.setHandler('attack', this.commandAttack.bind(this));
    this._actorCommandWindow.setHandler('skill',  this.commandSkill.bind(this));
    this._actorCommandWindow.setHandler('guard',  this.commandGuard.bind(this));
    this._actorCommandWindow.setHandler('item',   this.commandItem.bind(this));
    this._actorCommandWindow.setHandler('cancel', this.selectPreviousCommand.bind(this));
    this.addWindow(this._actorCommandWindow);
};

Scene_Battle.prototype.createHelpWindow = function() {
    this._helpWindow = new Window_Help();
    this._helpWindow.visible = false;
    this.addWindow(this._helpWindow);
};

Scene_Battle.prototype.createSkillWindow = function() {
    var wy = this._helpWindow.y + this._helpWindow.height;
    var wh = this._statusWindow.y - wy;
    this._skillWindow = new Window_BattleSkill(0, wy, Graphics.boxWidth, wh);
    this._skillWindow.setHelpWindow(this._helpWindow);
    this._skillWindow.setHandler('ok',     this.onSkillOk.bind(this));
    this._skillWindow.setHandler('cancel', this.onSkillCancel.bind(this));
    this.addWindow(this._skillWindow);
};

Scene_Battle.prototype.createItemWindow = function() {
    var wy = this._helpWindow.y + this._helpWindow.height;
    var wh = this._statusWindow.y - wy;
    this._itemWindow = new Window_BattleItem(0, wy, Graphics.boxWidth, wh);
```

```
    this._itemWindow.setHelpWindow(this._helpWindow);
    this._itemWindow.setHandler('ok',     this.onItemOk.bind(this));
    this._itemWindow.setHandler('cancel', this.onItemCancel.bind(this));
    this.addWindow(this._itemWindow);
};

Scene_Battle.prototype.createActorWindow = function() {
    this._actorWindow = new Window_BattleActor(0, this._statusWindow.y);
    this._actorWindow.setHandler('ok',     this.onActorOk.bind(this));
    this._actorWindow.setHandler('cancel', this.onActorCancel.bind(this));
    this.addWindow(this._actorWindow);
};

Scene_Battle.prototype.createEnemyWindow = function() {
    this._enemyWindow = new Window_BattleEnemy(0, this._statusWindow.y);
    this._enemyWindow.x = Graphics.boxWidth - this._enemyWindow.width;
    this._enemyWindow.setHandler('ok',     this.onEnemyOk.bind(this));
    this._enemyWindow.setHandler('cancel', this.onEnemyCancel.bind(this));
    this.addWindow(this._enemyWindow);
};

Scene_Battle.prototype.createMessageWindow = function() {
    this._messageWindow = new Window_Message();
    this.addWindow(this._messageWindow);
    this._messageWindow.subWindows().forEach(function(window) {
        this.addWindow(window);
    }, this);
};

Scene_Battle.prototype.createScrollTextWindow = function() {
    this._scrollTextWindow = new Window_ScrollText();
    this.addWindow(this._scrollTextWindow);
};

Scene_Battle.prototype.refreshStatus = function() {
    this._statusWindow.refresh();
};

Scene_Battle.prototype.startPartyCommandSelection = function() {
    this.refreshStatus();
    this._statusWindow.deselect();
    this._statusWindow.open();
    this._actorCommandWindow.close();
    this._partyCommandWindow.setup();
};

Scene_Battle.prototype.commandFight = function() {
    this.selectNextCommand();
};

Scene_Battle.prototype.commandEscape = function() {
    BattleManager.processEscape();
    this.changeInputWindow();
};

Scene_Battle.prototype.startActorCommandSelection = function() {
    this._statusWindow.select(BattleManager.actor().index());
    this._partyCommandWindow.close();
    this._actorCommandWindow.setup(BattleManager.actor());
};
```

```
Scene_Battle.prototype.commandAttack = function() {
    BattleManager.inputtingAction().setAttack();
    this.selectEnemySelection();
};

Scene_Battle.prototype.commandSkill = function() {
    this._skillWindow.setActor(BattleManager.actor());
    this._skillWindow.setStypeId(this._actorCommandWindow.currentExt());
    this._skillWindow.refresh();
    this._skillWindow.show();
    this._skillWindow.activate();
};

Scene_Battle.prototype.commandGuard = function() {
    BattleManager.inputtingAction().setGuard();
    this.selectNextCommand();
};

Scene_Battle.prototype.commandItem = function() {
    this._itemWindow.refresh();
    this._itemWindow.show();
    this._itemWindow.activate();
};

Scene_Battle.prototype.selectNextCommand = function() {
    BattleManager.selectNextCommand();
    this.changeInputWindow();
};

Scene_Battle.prototype.selectPreviousCommand = function() {
    BattleManager.selectPreviousCommand();
    this.changeInputWindow();
};

Scene_Battle.prototype.selectActorSelection = function() {
    this._actorWindow.refresh();
    this._actorWindow.show();
    this._actorWindow.activate();
};

Scene_Battle.prototype.onActorOk = function() {
    var action = BattleManager.inputtingAction();
    action.setTarget(this._actorWindow.index());
    this._actorWindow.hide();
    this._skillWindow.hide();
    this._itemWindow.hide();
    this.selectNextCommand();
};

Scene_Battle.prototype.onActorCancel = function() {
    this._actorWindow.hide();
    switch (this._actorCommandWindow.currentSymbol()) {
    case 'skill':
        this._skillWindow.show();
        this._skillWindow.activate();
        break;
    case 'item':
        this._itemWindow.show();
        this._itemWindow.activate();
```

```
        break;
    }
};

Scene_Battle.prototype.selectEnemySelection = function() {
    this._enemyWindow.refresh();
    this._enemyWindow.show();
    this._enemyWindow.select(0);
    this._enemyWindow.activate();
};

Scene_Battle.prototype.onEnemyOk = function() {
    var action = BattleManager.inputtingAction();
    action.setTarget(this._enemyWindow.enemyIndex());
    this._enemyWindow.hide();
    this._skillWindow.hide();
    this._itemWindow.hide();
    this.selectNextCommand();
};

Scene_Battle.prototype.onEnemyCancel = function() {
    this._enemyWindow.hide();
    switch (this._actorCommandWindow.currentSymbol()) {
    case 'attack':
        this._actorCommandWindow.activate();
        break;
    case 'skill':
        this._skillWindow.show();
        this._skillWindow.activate();
        break;
    case 'item':
        this._itemWindow.show();
        this._itemWindow.activate();
        break;
    }
};

Scene_Battle.prototype.onSkillOk = function() {
    var skill = this._skillWindow.item();
    var action = BattleManager.inputtingAction();
    action.setSkill(skill.id);
    BattleManager.actor().setLastBattleSkill(skill);
    this.onSelectAction();
};

Scene_Battle.prototype.onSkillCancel = function() {
    this._skillWindow.hide();
    this._actorCommandWindow.activate();
};

Scene_Battle.prototype.onItemOk = function() {
    var item = this._itemWindow.item();
    var action = BattleManager.inputtingAction();
    action.setItem(item.id);
    $gameParty.setLastItem(item);
    this.onSelectAction();
};

Scene_Battle.prototype.onItemCancel = function() {
    this._itemWindow.hide();
```

```
    this._actorCommandWindow.activate();
};

Scene_Battle.prototype.onSelectAction = function() {
    var action = BattleManager.inputtingAction();
    this._skillWindow.hide();
    this._itemWindow.hide();
    if (!action.needsSelection()) {
        this.selectNextCommand();
    } else if (action.isForOpponent()) {
        this.selectEnemySelection();
    } else {
        this.selectActorSelection();
    }
};

Scene_Battle.prototype.endCommandSelection = function() {
    this._partyCommandWindow.close();
    this._actorCommandWindow.close();
    this._statusWindow.deselect();
};


function Scene_Gameover() {
    this.initialize.apply(this, arguments);
}

Scene_Gameover.prototype = Object.create(Scene_Base.prototype);
Scene_Gameover.prototype.constructor = Scene_Gameover;

Scene_Gameover.prototype.initialize = function() {
    Scene_Base.prototype.initialize.call(this);
};

Scene_Gameover.prototype.create = function() {
    Scene_Base.prototype.create.call(this);
    this.playGameoverMusic();
    this.createBackground();
};

Scene_Gameover.prototype.start = function() {
    Scene_Base.prototype.start.call(this);
    this.startFadeIn(this.slowFadeSpeed(), false);
};

Scene_Gameover.prototype.update = function() {
    if (this.isActive() && !this.isBusy() && this.isTriggered()) {
        this.gotoTitle();
    }
    Scene_Base.prototype.update.call(this);
};

Scene_Gameover.prototype.stop = function() {
    Scene_Base.prototype.stop.call(this);
    this.fadeOutAll();
};

Scene_Gameover.prototype.terminate = function() {
    Scene_Base.prototype.terminate.call(this);
    AudioManager.stopAll();
```

```
};

Scene_Gameover.prototype.playGameoverMusic = function() {
    AudioManager.stopBgm();
    AudioManager.stopBgs();
    AudioManager.playMe($dataSystem.gameoverMe);
};

Scene_Gameover.prototype.createBackground = function() {
    this._backSprite = new Sprite();
    this._backSprite.bitmap = ImageManager.loadSystem('GameOver');
    this.addChild(this._backSprite);
};

Scene_Gameover.prototype.isTriggered = function() {
    return Input.isTriggered('ok') || TouchInput.isTriggered();
};

Scene_Gameover.prototype.gotoTitle = function() {
    SceneManager.goto(Scene_Title);
};
```