Generality of Online Games

Generalitet i Online Spil

Adam Hayeem

S022591@student.dtu.dk

Eksamensprojekt

28/2 2006

Vejleder: Niels Jørgen Christensen

Preface

The gaming industry is one of the biggest industries in entertainment, having surpassed that of the movie industry as an example. And it is getting bigger. To be working on a project that lies within the industry is something that I have always wanted to do.

Unfortunately, the gaming industry is still very much in its adolescent stages, without much concrete theory. The result is that much of the analysis of this project is based on other similar games of design, and that much of the theory is based on loose descriptions here and there in books. Articles concerning the exact elements of, for example, a role-playing game are hard to come by and may differ in description from one another. Luckily, the road is taking a new turn and more and more theory is being produced.

I regret having chosen so large a subject to deal with. There are many elements within this design that could have been analyzed much further and would have added to a better result. For example, an in depth analysis of today's aggro systems could have been a project in itself and would have extended the solution to this design's system, creating a much better basis for strategic combat. The greatest challenge of this project is therefore the immensity of it.

As a side note, decimal points have been used instead of the Danish comma when dealing with numbers for international standardization.

Abstract 3
About the Project 4
- Target Group 5

Problem Definition 6
Theory 8
- Role-Playing Games 8
- Balance and RPGs 9
- Massively Multiplayer Online Games 9
- MMORPG Settings 9
- Generic RPG Systems 10

Analysis 11
- Core Mechanics 12
  - The Performance Bar 14
  - Automatic Success and Failure 14
- Character Creation 15
  - Attributes 15
  - Skills 18
  - Skill Based Systems and Class Based Systems 19
- Advancement 21
  - Experience 21
- Combat 22
  - The Combat Round 22
  - Hitting and Missing 22
  - Damage 25
  - Hit Points 27
  - Critical Hits and Fumbles 28
  - Aggro 28
  - Combat Advancement 30

Hypothesis 31
- The Core Mechanic 31
- Character Attributes 31
  - The Mental Attributes 31
  - The Physical Attributes 32
  - The Spiritual Attributes 33
- Combat 34
  - Hitting the Opponent 34
  - Blocking, Parrying and Dodging 38
  - Damage and Critical Hits 38
  - Weapon Speed 38
  - Range 38
  - Facing 38
  - Damage Reduction and Attack Types 39
  - Aggro 39
  - Combat Skills 40
- Analyzing the Outcome of Combat 40

Discussion 41

The Core Mechanic 41
Level Based System 41
Surprise and Initiative 42
The Attributes 42
Hitting and Missing 43
Weapon and Defense Proficiencies 44
Overall Attack and Overall Defense 45
Weapon Speed and Damage 45
Damage Reduction 46
Aggro 47
Advancement 48
The Combat Analyzer 49
Test Scenario ***50***
The Warrior 50
The Rogue 51
Combat 52
Conclusion ***54***
References ***55***
Appendix Section ***56***

Abstract

Generic rule sets are universally designed to contain rules for any role-playing setting. The most successful rule sets in the pen and paper-role playing game community are the generic ones. Despite this fact, the computer industry for role-playing games has yet to implement a generic solution specifically designed for the computer. Instead, many computer role-playing games have borrowed generic rule sets from the pen & paper market, or have created "specific" rule sets for their games lacking the power of the generic rule sets.

The Information Technology University of Denmark currently lacks a generic rule set for a combat system for their massively multiplayer online role-playing game (MMORPG) engine. To provide them with this system, the generic rule sets of the most successful systems in the pen and paper market have been analyzed, together with some of the specific rule sets found in the MMORPG market. The design of the combat system resulted in an intuitive core mechanic, and several algorithms that administer combative elements such as hitting and parrying a target. Because the combat system is often closely tied to the rest of the rule system, rules influencing combative values have also been analyzed and designed.

The implementation of the project was conducted in the python programming language and resulted in a system for the design and balancing of characters in terms of combative conflict. In comparison to other combat systems, the design of the system is more complex equipping future game developers with a much finer degree of balancing elements but unfortunately also expands the time needed for new players to learn the system. Solutions to the latter have been discussed.

About the Project

Generic rule sets have become the most successful rule sets within the role-playing pen and paper community. They provide players with a basic framework of rules that can be used for any type of setting, be it a historically correct setting such as ancient Greece or a futuristic fictional setting such as “Starwars”. The popularity of role-playing games has substantially increased over the years and recently begun to dominate the computer game market with role-playing titles like the “Diablo” series, the “Baldur’s Gate” series, “Everquest”, “Lineage”, and “World of Warcraft”.

As the number of computer role-playing games increases, the choice of developing a generic rule set becomes more and more important for companies that continue to develop games within the role-playing genre. A generic rule set saves both time and money since the development of successive role-playing titles need not start with a completely new rule set. As an added bonus, the company will also ease the learning curve of the game for the player since the player may have experience with rule set from another game.

There are two popular approaches in the pen and paper market for creating a generic rule set. The first involves a system that encompasses all rules for typical settings. Note that a rule system that encompasses all possible settings cannot exist, since it would imply that our imagination is limited.

The second approach is to create a very basic rule set governing only common role-playing elements that are the same regardless of setting. The basic rule set is also a framework for adding new rules for role-playing settings. The two options are illustrated below.

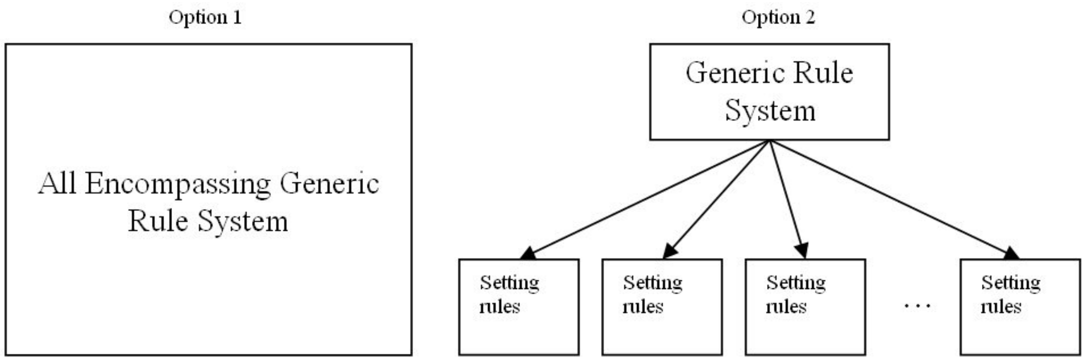


Figure 1: Two widely used design solutions in the pen and paper role-playing game market.

The pen and paper generic rule sets have proven to be a good solution to implement for computer role-playing games, but many companies choose to create their own rule sets regardless. This might be because they want a non generic rule set, with rules specifically designed for a game in mind, or because they wish to design a rule set specifically for the computer.[1]

The Information Technology University of Denmark (ITU) is working on a project named ODDPAW,[2] short for Open source Design and Development of Persistent gAme Worlds.

[1] Computers have their share of strengths and weaknesses, and it is these strengths and weaknesses that the design of the rule set must take into account, instead of those of a human which is the case with pen and paper systems.
[2] See the appendix for more information concerning ODDPAW

ODDPAW is a framework for persistent world games (more often called massively multiplayer online games or MMO[3]s) designed to test new concepts of play. A generic rule system is important for OODPAW enabling its designers to create any role-playing setting with a minimum of effort in terms of the rule set. ODDPAW is a framework for any type of MMO, but the facts are that currently the most successful MMOs are the role-playing games which hold 90% of the market share. [Woodcock, 2005] Therefore, it is safe to create a rule system exclusively for the role-playing market.

ITU could choose to base their rule set on some of the existing pen and paper rule sets. Unfortunately, the most popular and by far the most successful system, the d20 system, forces ITU to abide by a licensing agreement. In order to make use of d20's trademarked terms and a distinctive logo to help consumers identify the products, ITU must abide by a license known as the d20 System License (STL). Some of the problems with the license are that Infrogames Inc. has all the rights to create d20 "interactive games" as described by the STL. Should Infrogames give permission to create a d20 engine, the STL is still revocable and controlled by Wizards of the Coast (WotC), owner of the d20 system. WotC have the ability to alter the d20 license at will and gives a short, 30 day "cure period" to rectify any issues with the license before termination. Upon breach of the terms of the d20 license, all inventory and marketing material regarding the license must be destroyed.

The ODDPAW project has not found an open source core to base itself upon, but much implies that the Underware project[4] by Mekensleep.org will be that core. Underware is currently under development and is planned to be a set of free software libraries and tools used to produce 3D online persistent universes. Because the Underware core lacks a combat system, the ODDPAW project currently has an available satellite project that deals with this need. The result is this thesis.

Target Group

The report is written for pre-graduate students with a technical background in computer science but without any special knowledge of the role-playing community.

[3] Notice that MMO does not contain an abbreviation for the word Game although it stands for Massively Multiplayer Online **Game**.
[4] See the appendix for more information concerning Mekensleep and the Underware project.

Problem Definition

The problem definition can be summed up in five steps. A more thorough description of the project definition follows.

1. Develop a general rule set for a combat system that can be used as a framework for the development of any Massively Multiplayer Online Role-Playing Game (MMORPG) genre[5] within the ODDPAW project at ITU.
2. Analyze the framework by comparing it to existing rule sets such as the d20 system reference document.[6]
3. Develop a "Number Simulator"[7] program for prototyping and balance purposes in terms of the combat system. The program must be developed using the Python[8] programming language, and must produce a log of its tests.
4. Develop an analysis program that analyzes the number simulator log.
5. Create a test scenario to show how the system works. The test scenario will result in a graphical rendering of combat in order to visualize the generality of the combat and rule systems.

ECTS Points : 30

A rule set contains a baseline of game values, and it is often appropriate to start the development phase with the rules governing the combat system[9]. The combat system in role-playing games is usually highly interrelated with the rest of the rule set, and therefore it only makes sense to develop the combat system in unison with that part of the rule set. Once the combat system and interrelated rule set are in place, more rules can easily be applied to the system concerning other aspects of the game.

The combat system will result in a set of software libraries created in the Python programming language that does *not* rely on any of the technologies present in Underware. The combat system is simply an addition that lies on top of the Underware core (see Figure 2) but is based on the needs of the OODPAW project.

[5] A genre is the same as a role-playing game setting.
[6] The d20 System is a game engine created by Wizards of the Coast for their hobby gaming role-playing games. The system reference document (SRD) is based on the d20 System, and the SRD is an abstract version of the rules and nonrule content of the various d20 system role-playing games from Wizards. [Wizards, 1995]
[7] See section 1.3 in [Alexander, 2003].
[8] Python is an interpreted, interactive, object-oriented programming language used in many MMOs today.
[9] Many computer role-playing games are primarily based on their combat engine.

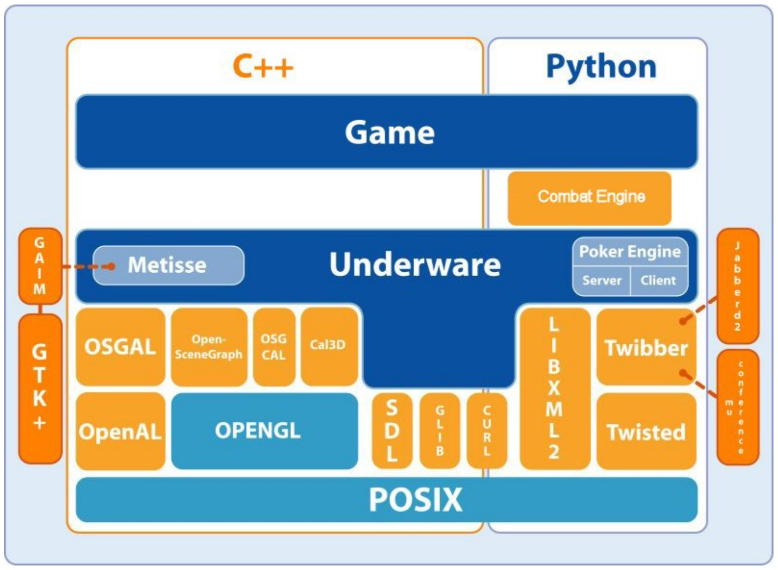


Figure 2: The figure illustrates the Underware design diagram with the addition of the MMORPG combat engine located directly under the actual Game. The image illustrates which library components that Underware consists of and places them according to language. The figure is taken from www.mekensleep .org and has been modified to add the combat engine.

A comparison of the MMORPG combat library will take place in the discussion section of the thesis and answer questions as to why design choices have been made. The discussion will take other successful role-playing games into account.

The number simulator is a tool for the developer to test the design of characters in terms of their balance during combat. Its purpose is to act as a balancing system for any given genre. The objective of the number simulator is *not* the development of equipment and items, but rather to test for balance between equipment, items, and races during combat. The results of these tests will be saved in a log so that they can be analyzed by the program.

Since the primary purpose of the number simulator is to balance a given game, a test scenario will be needed. The test scenario will be used to show how the number simulator works and will result in a graphical rendering of, for example, two warriors fighting to visualize how the combat system and rules system work.

Theory

Role-Playing Games

Role-playing games (RPGs) are a form of interactive, cooperative and collaborative storytelling. Participants play either as characters in an imaginary world (also called the players of the RPG) taking both roles of audience and character or play as the gamemaster[10] (GM) who is the main storyteller and referee. The players are the main "actors" in the story while the GM plays the parts of all the supporting roles and is also responsible for advancing a storyline or plot and directing the outcome of player decisions. [Wikcon, 2006]

In RPGs players and GM cooperate with one another instead of competing against each other. There is no purpose of coming out the "winner" since there is no way in which one can beat, or win the game. The players and GM are considered to be on the same team while the GM also plays the part of the opposing team. The GM is in charge of presenting the players with challenges and choices, but also in charge of indirectly helping the players to make the correct choices and defeating the challenges. Although the GM is considered the main storyteller, the story is told by both players and GM; the reason being that the GM has no control over actions of the players' characters. [Wikcon, 2006]

Even though RPGs are considered stories, they have rules as well. These rules help determine the success or failure of character endeavors and therefore normally involve assigning certain abilities to characters. Abilities may represent the skills of a character such as driving, computer or cooking skills, or represent the physical or mental attributes of the character such as strength and intelligence. Furthermore, almost all RPGs introduce an element of chance through the use of dice. Modern day RPGs also require information on the background, personality and resources of the character. This information is usually stored on a character sheet often taking the form of numerical values. [Wikcon, 2006]

There are many different variations of playing RPGs, but in terms of this paper, the traditional method of playing a RPG, the pen and paper (P&P) games, and the computer role-playing games (CRPGs) will be of interest. In P&P games the rule system often uses dice to determine the success and failure of events. Several people, often around five people, are involved in play, and combat is a significant aspect of such games, although it is far from a requirement. Miniatures are often used during combat to represent strategies and position which in turn may affect probabilities and chances. CRPGs have much in common with the P&P game but are usually much more combat oriented because they lack the ability to produce free a cooperative storytelling environment as the P&P game. They can be divided into persistent and non persistent worlds, and each of these further divided into text based or graphical represented worlds. One major difference between CRPGs and P&P RPGs is that the rules are obscured from the player in CRPGs. [Wikcon, 2006]

Typical elements of RPGs include classes (fighters, wizards, paladins…), character statistics, experience, magic, and combat. Character development is accomplished through rise in statistics and abilities, acquired items, and through story. [Hallford, 2001]

[10] A.k.a. the dungeon master.

The defining characteristic in what makes a role-playing game a role-playing game is the development of the role of the player's character or in other words the player's identification with the character. [Hallford, 2001]

Balance and RPGs

Balancing a RPG is one of the most important issues in good RPG game design. Balance should ensure that one character does not unfairly outperform another, and that the game is not too hard or too easy to play. RPGs often use a paper-scissor-rock like system to ensure a degree of balance, and in terms of the combat engine the usual systems that need to be balanced are [Carpenter, 2003]

- Player character races
- Player character classes
- Player vs. environment conflict
- Player vs. player conflict
- Player skills

Massively Multiplayer Online Games

There is still no accurate definition of what a MMO is. However, there are some guidelines that are used to define what sets MMOs apart from other games. MMOs are different because of an often added persistency aspect within its virtual world. Persistency means that the game continues regardless of whether you or anyone else is playing it. The game state rarely resets meaning that what you earned or overcame yesterday is still actual today. These games are multiplayer games by design, and therefore most of them lack single player aspects such as being able to "win" the game or save and reload the game. MMOs are different from "classic" multiplayer games such as "Counter Strike" or "Total Annihilation" because they are capable of hosting a large number of players (usually in the thousands) in a single game world where all players can interact. Another requirement for a game to become a MMO is the need for a large scale virtual world. The world usually consists of large areas that are interconnected within the game in such a way that a player can traverse vast distances without having to switch servers manually. [Wikcon, 2006]

Apart from the differences mentioned above, there are a number of subtle differences. MMOs usually charge the player a monthly fee to play. MMOs offer support for clans and guilds in the game. Examples of where the boundaries are not clear or obvious include the game "Guild Wars" that has been called a MMO even though most of its gameplay is set in private areas for groups of players. "Diablo II" and "Neverwinter Nights" have also been called MMOs even though these game worlds usually only support less than one hundred players. [Wikcon, 2006]

MMORPG Settings

There is no perfect distinction between the genres of games. Nevertheless, genres are distinguished from one another by what characteristics predominate and help us differentiate between games. In role-playing games these genres are called settings. The settings are often divided into the following list of taxonomy (or a combination of these):

- Fantasy (Dungeons and Dragons, Tolkien, Vampire)
- Modern (Detective, Mafia, Military)
- Future (Starwars, Startrek, Punk)
- Horror (Call of Chulhu)
- Super-hero (X-men, Spiderman)
- Historical (Rome)

Generic RPG Systems

Recall that role-playing systems are rule systems that are used to determine the success or failure of character endeavors. A “normal” role-playing system will only provide these rules with a specific setting in mind while a generic system will provide basic rules for any role-playing setting. [Wikcon, 2006]

There are several advantages and disadvantages concerning generic systems. As noted before, generic systems save money for both the developer and player who wish to engage in development or play of different settings. Secondly, since the generic system will cover a large set of basic features shared by all settings, players and developers will not have to relearn these rules when switching to another setting. Furthermore, during a switch from an old to a new setting, characters of the old setting may often be transferred into the new system with a minimal of changes. [Wikcon, 2006]

Disadvantages include a rule system that may often be more complex than that of its counterpart since the rules must govern features that sometimes might not be used in certain settings. The generic rules might not cover enough rules to provide meaningful gameplay in certain settings. This can be resolved with an addition of setting specific rules, but results in a larger number of changes needed if a character must be transferred from one system to another. [Wikcon, 2006]

The basic generic rules covered by some of the most popular generic systems include the rules governing:

- The attributes of the character.
- Movement and exploration, including carrying capacity and environment.
- Character advancement.
- General combat.
- Some conditions such as being prone, panicked or paralyzed.

Rules that are often setting specific and not necessarily a part of the generic rule set include the rules governing:

- Equipment.
- Skills and classes.
- Professions.
- Races.
- Spells and magic.
- Natural and special abilities.
- Monsters.
- Setting specific combat.
- Some conditions such as sicknesses.

Analysis

Before an analysis is conducted on how combat works in role-playing games, possible bindings with the rest of the rule system must be fully understood. The process is best conducted via a character generation analysis[11] and a core mechanic analysis to get a structured order of bindings. The design of the combat engine will first be conducted after the analysis.

Analyzing popular game systems should give an idea as to how the core mechanic and the combat system can be structured. The games primarily influencing the analysis are the three systems presented below. Other popular systems that have been reviewed and have had an influence are presented in the appendix section.

The d20 System
Currently, the most successful Pen and Paper (P&P) role-playing rule system is the d20 system created by Wizards of the Coast.[12] The system is named after its core mechanic, the twenty sided die, and was published in 2000. Much of the system is released via the System Reference Document (SRD[13]) and can be used freely in accordance with the Open Gaming License[14] also published by Wizards of the Coast. The strength of the d20 system and one of the prime reasons for its popularity.[15] lies in its inherent generic design.

World of Warcraft
Blizzard Entertainment® Inc.,[16] a division of Vivendi Universal Games, is a premier developer and publisher of entertainment software renowned for creating many of the industry's most critically acclaimed games. Blizzard's track record includes nine "#1-selling games" and multiple "Game of the Year" awards.

If not the largest and most successful MMORPG game of today, then it is close to being it, is World of Warcraft with over 5 million paying customers[17]. The game is developed by Blizzard Entertainment.

GURPS
GURPS stands for Generic Universal Role-Playing System and is created by Steve Jackson Games. It is one of the first[18] role-playing systems to have been universally designed for any genre within RPG and has won a "Best Role-Playing Rules" award.

GURPS was one of the first RPGs to introduce a point based system in which one "buys" characteristics and skills instead of having them assigned.

[11] Elements within the character generation process, that have nothing to do with the combat system, have been left out of the analysis.
[12] http://www.wizards.com
[13] The SRD can be found at http://www.wizards.com/default.asp?x=d20/article/srd35
[14] http://www.opengamingfoundation.org/ogl.html
[15] Other reasons include Tactical Studio Rules (a.k.a. TSR; the original creators of the dungeons and dragons game system now owned by WotC as of 1997) already having a large fan base, and a good understanding of economics resulting in the release of the open gaming license.
[16] www.blizzard.com
[17] http://www.blizzard.com/press/051219.shtml
[18] Other examples include the Chaosium role-playing system, best known for the highly successful *"Call of Cthulhu"* role-playing game, which was also developed to be a "generic" set of role-playing rules.

Core Mechanics

It is important to understand that the complexity of the rules concerning the character and its interaction with the world may be simplified. The simplification is often called the core mechanic of the system and many pen and paper RPGs are based on a single core mechanic that is used throughout the system[19]. The core mechanic is primarily a tool to determine the success and failure of events.

There are two traditional forms of core mechanics. [Hallford, 2001] The first is known as the percentile skill system where the avatar's skills are represented via percentile values. These values reflect the basic chance for success. A roll is made and compared to the percentile value. If the roll exceeds the percentage, then the roll is considered a failure.

The second system is called the threshold skill system. This system also sets values defining the basic skills of the character, but no element of chance is used. Instead the value is compared to the difficulty class (DC) of the event that the avatar wishes to undertake. If the character possesses a higher skill level than the DC, the event is a success. Both systems contain flaws and because of this many of today's successful systems are based on a mixture of the strengths of both systems. [Hallford, 2001]

For example the core mechanic of the d20 system is "roll a d20[20], add the relevant modifiers and compare the result to a target number". The roll represents the element of chance as in the percentile skill system, whereas the modifier and comparison to a target number can be interpreted as representing the threshold system. The target number is called the difficulty class in the d20 system and is usually set by the GM. If the roll hits or exceeds the DC, then your actions have been successful, otherwise you will fail. A simple example of its use; two characters attack one another.

$$\textbf{Success} \text{ if d20 + modifier1 + modifier2 + ... } \geq \text{ Difficulty Class}$$
$$\textbf{Failure} \text{ if d20 + modifier1 + modifier2 + ... } < \text{ Difficulty Class}$$

It is character one's turn and in order to hit character two, a d20 is rolled. An attack bonus (a modifier) and a strength bonus (a modifier) is added to the roll and compared to character two's armor class (the DC). If the roll exceeds the DC, character two is hit, otherwise character one has missed. This mechanic is used in almost all cases in the d20 system. The only clear exception is rolling for damage.[21]

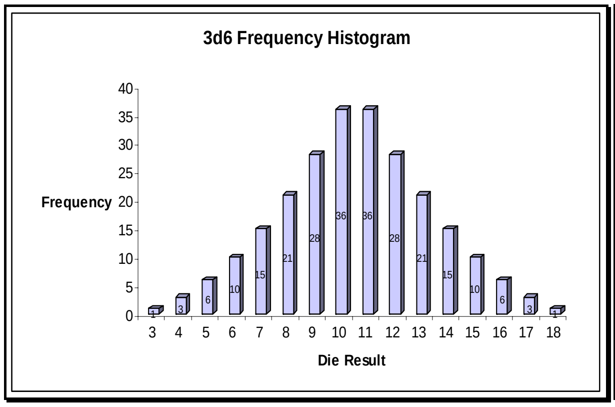
An example of a different core mechanic is the GURPS system which uses the six-sided die (d6). Even damage is calculated as a result of the six-sided die. Statistic and skill[22] checks are performed by rolling three six sided dice with the result compared to the statistic's rating or skill's level.[23] Here the mechanic has been reversed; Instead of comparing the roll to a target number set by the GM, the objective is to roll as low as possible preferably under one's own

[19] GURPS, Tri Stat, d20, and Hero are all based on single core mechanics.
[20] d20 means the twenty sided die. For example, 2d20 represents two rolls of the twenty sided die. The results of the two rolls are added to give a final result. Another example is 2d4+3 where a four sided die is used instead and a modifier of three is added after the 2 rolls have been summed (yielding a final result from 5 to 11).
[21] Damage often varies depending on the type of weapon used, and therefore the d20 system uses a larger variety of dice for this purpose.
[22] Recall that most RPG systems appoint statistics and skills to define a character. A check simply determines if the character is able to succeed in an action, such as lifting a heavy object using the character's strength statistic.
[23] Traditionally a statistic is a numeric value, while a skill is a description of an action. Skills are dealt into levels to describe the efficiency level of the skill. See the "Skills" section of the analysis for more on this.

target number in order to succeed. In this system, the GM only has control over some variable modifiers. The DC is always set according to the character's statistics or skills.

Success if 3d6 $\leq$ Statistic or Skill + GM modifier
Failure if 3d6 > Statistic or Skill + GM modifier

There are many other examples of core mechanics, but most are similar in one way or another to the above two. The first object of notice is that both systems ensure that there can only be so and so many variations of a result when using a d20 or d6. This is fine in many situations, but there are a lot of cases where a larger variety of results is desirable. Typical ways of resolving this problem is by using a larger die such as the percentile die (d100).



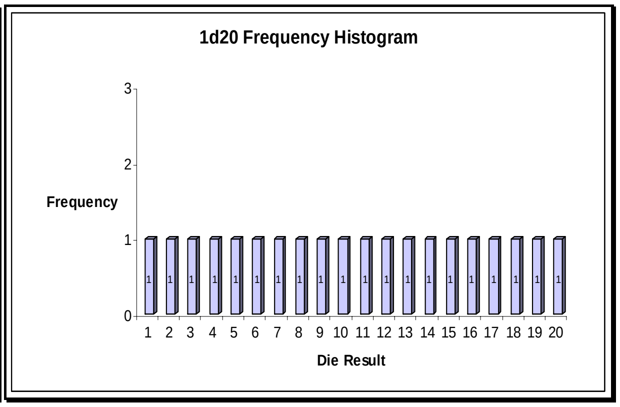


Figure 3: Frequency histograms of the 3d6 and d20 core systems with intervals set according to die results.

The GURPS 3d6 histogram shows that the greatest frequency of rolls will result in rolls of 10 or 11. The d20 system provides an equal chance for any roll result. The histograms give a good understanding of how luck plays a big role in RPGs. At early levels in the d20 system, one might have an average skill modifier ranging from +0 to +8. The roll will on the average represent 83.3% of the result. At higher levels, this will drop depending on the amount of points set into skills. For example, at level 15, a character might have as much as a +19 modifier in a chosen skill in terms of the rules and the die roll will represent 51.3% of the result.

The standard variations of some different core systems are shown in the following table.

Core system	Mean	Standard variation
d20	10.5	5.92
2d10	11	4.08
3d6	10.5	2.96

Figure 4: The mean and standard variation of some sample core mechanic systems.

The 3d6 standard variation is about 3 whereas the d20 standard variation is about 6, meaning that the 3d6 core mechanic system is about half as variable as the d20 core mechanic.

The 3d6 histogram could be interpreted as a symmetric unimodal[24] frequency diagram[25]. If the modal bar were to be moved farther to the left, the histogram would be a right skewed histogram. If the median was still located in the modal bar, the right skewed histogram would ensure that a character either never performs under initial skill level or almost never, depending on the

[24] A mode is a value or item occurring most frequently in a series of observations.
[25] Let the set {10,11} represent the die results of 10 and 11 instead.

placement of the modal bar. This is important for developers wishing to create truly heroic games (as is often the case with super hero settings) where the character rarely performs badly. The opposite would occur with a left skewed histogram.

The Performance Bar

The core systems described above are success/fail systems. The introduction of a performance bar, by some systems, adds degrees of success and failure to the system. The performance bar is a ladder that characters can ascend or descend depending on their roll. It is primarily used to determine the performance of an ability or skill. To use the bar, the relevant skill is placed on the bar dependent on the current performance level of the skill (for example, a character has a fair jumping skill). A roll takes place, and depending on the result of the roll, the character either ascends to a greater performance level, or descends to a lesser one. The mean roll will usually result in the current performance level of your skill on the performance bar, while a higher roll will increase your performance level and a lower roll decrease it. With a standard variation of 2.96 the GURPS system would ensure that one’s level of performance often would be within close range of one’s initial performance skill.

Olympic
Excellent
Good
Fair
Ordinary
Poor
Terrible

Figure 5: An example of a performance bar.[26]

Automatic Success and Failure

The automatic success and failure rolls exist to reward characters for an outstanding roll or to punish characters for a poor roll. It brings an added level of excitement to gameplay, because of the small chance that something seemingly “impossible” might happen. In terms of combat, it gives an amount of hope in defeating a vastly superior enemy.

Both of the above mentioned systems have an automatic success and automatic failure roll. In GURPS a dice roll of 18 or 17 is always a failure, and a roll of 3 or 4 is always a success. In d20 a roll of 20 is always a success while 1 is always a failure.[27] The result of the d20 system is a 5% chance for an automatic success roll and a 5% chance for an automatic failure, while the GURPS system results in a 1.7% (4/240) that each situation will occur.

[26] This performance bar is a simple example of one. In fact, one could actually exceed the amount of steps on this example bar if one rolls exceedingly high. To counter this, we might say that it takes 3 points of a roll over 11 to take one step up the bar, and vice versa, and that the highest and lowest skill level one may achieve, is already represented by the bar.

[27] Note that some systems include optional rules for automatic success or failure. For example, d20 states that instead of a roll of 20 being an automatic success, one can merely add an extra 10 as a modifier due to the outstanding roll. Likewise a roll of 1 should add a negative 10 modifier.

Character Creation

Character creation is one of the defining elements for RPGs. [Hallford, 2001] The character creation process assigns initial values usually closely tied to the combat engine. The assignment and advancement of these values influence the balance of the combat engine. In computer games, there are three traditional approaches to describe the creation of a character: Character Generation, Class Selection, and the Founding Approach. [Hallford, 2001]

- **Character Generation** involves the player determining the statistic scores of the character. The system gives the player a significant amount of control of the type of character he will be playing. Cons include a greater level of complexity and a need for the player to possess a good amount of understanding of the rule system so that the significance of the choices is understood.
- **Class Selection** allows the player to choose the overall type of character that the player wishes to play without having to set the statistics. Cons include the cons that are described under class based systems in the section "Skill vs. Class Based Systems".
- **The Founding Approach** hands a character out to the player at the beginning of the game. The player is then in charge of evolving the character into the character of choice during the game. Unfortunately, this means that every player starts out with the same character.

Attributes

The character creation process involves the assignment (either by the player or the developer) of statistics describing the psyche and physique of the character. These statistics are called attributes.[28] An example is the GURPS concept which only includes 4 attributes governing strength, dexterity, intellect and health. These four attributes represent the four most common attributes found in P&P games. Attributes are usually defined through numeric values where a high value represents a higher degree of excellence in the attribute. A number of derived attributes such as reaction, speed, willpower, perception, hit points[29] and fatigue points are then calculated via the four defining attributes. In GURPS, a score of 10 points represents an average score, while 18 is considered super human, and 1 completely lacking of the attribute. Most systems resemble GURPS in one form or another. The d20 system contains no derived values, but instead uses six attributes. Its attributes start at a score of 1 and provide modifiers to the core mechanic. For example, a score of 1 provides a modifier of -5. An average human score lies between 10 and 13, where a 10 provides a modifier of 0, 12 a modifier of +1 and so on. An attribute of less than 1 means that the character either does not possess the attribute or that the character has died from loss thereof. The modifiers for the d20 system are found via the following formula.

$$\frac{Score-10}{2} \text{ (round down) = modifier}$$

[28] Although many would argue that attributes are essential to defining a character, the MMORPG "Guild Wars" has proven otherwise. There is no mention of any attributes within the system, and a character is represented only through his skills.

[29] Hit points are not self explanatory, but are the common statistic used to define the health of a character.

Typical Combat Uses of the Attributes

- **Strength** influences the amount of damage caused and often one's chance to hit with a weapon. Strength may also influence the amount of damage absorbed from a blow to a shield.
- **Intellect** may govern the amount of combat skills available to a character, as well as how quickly one learns the given skills. It also may influence the chance to resist mind numbing effects and influence the damage of certain intellectual abilities and skills.
- **Dexterity** represents the reflexes of the character which is often used to get away from harmful effects. Some weapons, such as a dagger, require dexterous hands instead of sheer brute strength to use effectively and in these cases the dexterity attribute is used to calculate the hit chance of the weapon. Dexterity may also dictate general accuracy.
- **Health** influences the amount of life points a character possesses, how fast the character may grow fatigued during combat and sometimes the chance to resist mal effects to the body.

Level of Influence

A common factor in almost all RPGs is the attainment of more attribute points as one's character advances in the game to represent the character growing stronger or more intelligent. [Hallford, 2001] Because attributes influence the rule system in different ways it is important to balance the level of influence of an attribute together with its increments.

Using the d20 system as an example, a sword that does 1-6 points of damage upon a hit has its damage additionally influenced by the strength modifier of a character. The modifier also acts as a bonus to the character's chance to hit a target.

Hit	if d20 + strength modifier $\geq$ DC
Miss	if d20 + strength modifier < DC
Damage	1-6 + strength modifier

An increment in strength will have to be balanced with the defense of the opponent (in this case the DC) as well as with the health of the opponent (one subtracts damage from health). Taking the example a little further, a character might attack three times every five seconds gaining the effectiveness of the attribute threefold compared to a character attacking only once every five seconds.

The number of increments during the life span[30] of a character differs from game to game but it is interesting to note that CRPGs that are not based on pen and paper generic systems have a finer degree of attribute effectiveness. Most games define an amount of points in an attribute that should be perceived as the average human potential. With this information, an average point worth for an attribute in terms of the human potential can be calculated.

System Name	System Type	Average Point Worth in Terms of Average Human Potential
d20	P&P	8.6%
GURPS	P&P	10%
Tri Stat dx	P&P	25%
Chaosium Call of Cthulhu	P&P	9.5%
d4-d4	P&P	25%

[30] Where "life span" means from the point of creation to the time where the character cannot advance anymore.

Dark Age of Camelot	CRPG	Approx 3%
World of Warcraft	CRPG	Approx 5%

Figure 6: The table provides a point value of an attribute's worth in terms of the average human potential. The games "World of Warcraft" and "Dark Age of Camelot" do not note an average human potential score, and therefore the percentage value is based on approximate average starting game scores, since most games start players off as the average human.

For example, each attribute point in the d20 system is worth 8.6% of the average human potential (stated as being around 10-13 in the d20 system). The effectiveness of 1 point in an attribute therefore corresponds to 8.6 percent of the average human potential.

Assigning Points

There are a large number of ways to generate a character's ability scores. The most popular[31] are known as random generation, planned generation and standard score packages.

The random generation system is often used when players are (or should be) satisfied with a random result of their ability scores. The system is designed to create a character for the player, and then have the player decide how to role-play the character based on the scores given. Unfortunately the system introduces an amount of imbalance. While the majority of players will roll average scores, some will roll exceedingly well and others exceptionally bad scores. Games based on this system include the award winning "Baldur's Gate" and "Baldur's Gate 2" series.

More often than not, a player already has in mind the type of character that he wishes to role-play. The planned generation system is designed for this purpose. It involves granting all players an equal number of points to spend on ability scores. The cost usually grows as the attribute score gets higher, ensuring that players wishing to have exceedingly high scores in one area will have to sacrifice a lot in other areas. This system is used in some games such as "NeverWinter Nights" and "Dark Age of Camelot".

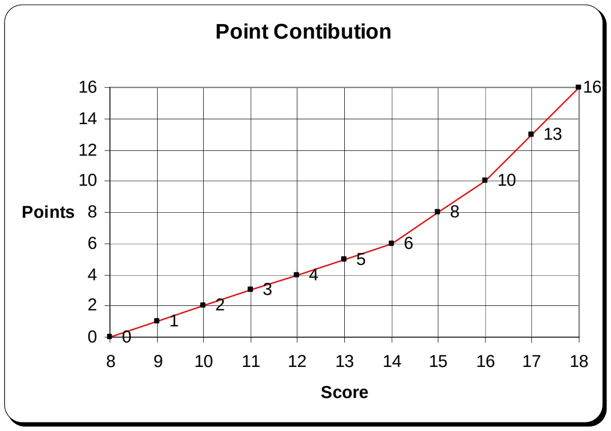


Figure 7: The planned generation system of the d20 system is visualized above with point contribution growing as the score is increased.

Finally there is the standard score package. A standard package of scores is available ensuring that everyone has the exact same scores to place as the player sees fit. In many circumstances the

[31] In terms of my own experience.

scores are already placed according to the class one wishes to play. This increases balance considerably since the developer will know exactly what attribute scores the characters possess. Unfortunately everyone is alike, only class and appearance set them apart. “World of Warcraft”, and “Diablo” use the standard score package where the scores have also been fixed according to class.

Skills

Skills are the synonym used for the abilities that a character possesses. Skills may represent any ability that is not represented through an attribute. Examples of skills are swimming, hearing, bluffing, diplomacy, hiding and so forth, but also extend into combative abilities such as dodging, and casting a spell. Because skills may represent many different types of character abilities, skills are often divided into separate systems for balance purposes. For example, a combat based game will usually provide players with a separate combat skill system to ensure that players choose combative abilities.

Upon creation, characters have no skills. Skills are either “bought” through a point system (much like the buying of attribute points) or simply handed to the character according to class[32] chosen, or a combination of the approaches. Many skills have varying degrees of efficiency so that characters differ in ability. For example, your character might swim better than mine, even though we both posses the swimming skill. To incorporate this, skills use a point system just like the attribute system. The more points spent in a skill, the higher the level of skill and the more efficient you will become in that skill. Skills which are considered hard to learn (such as rocket science) simply cost more points to learn.

Skill Availability

The availability of skills must be balanced to ensure that the game does not become too difficult or too easy. There are a number of ways to deal with the availability of skills besides the point buying concept mentioned above.

- Skills may be restricted according to class when using a class based system.
- Skills can be tied to the level system, limiting a skill until the advancement of a level has been attained.
- Skills can be tied to a usage system. Higher levels of proficiency are available after suitable use of the skill.
- Skill trees occur when skills are prerequisites for the availability of other skills, or where level is a prerequisite for the availability of a skill.
- The availability of a skill may be tied to the attributes of the character as may the strength of the skill.
- Skills may become available through exploration of the game world.

Skill Use

Many skills can be used whenever the player wishes, but usually there are skills that must be balanced in terms of their use. There are many approaches on how to deal with the use of skills. The list below contains a few of the popular ones.

[32] A class is a category of members having certain attributes or traits in common [Lexico, 1995]. In RPGs classes are archetype characters such as a fighter, wizard or priest.

- Cool down control: The skills is ready after a cool down time that begins counting down as soon as the skill has been used.
- Casting time: It takes a given time frame to use the skill.
- Reagent control: You will need something in order to use the skill, and this item may be consumed upon activation of the skill.
- Situation control: The skill can be used in certain situations such as after having used another skill.
- Mana control: See section on "Mana"

Mana

Mana refers to a supernatural force said to exist within all things. It is often used in many fantasy settings sometimes with a different name such as "force", "ki" or "power". The primary use of mana is to act as a well for magic skills. Once a character's mana well has been drained, the character cannot cast anymore spells until enough of the well has been replenished. The well may be represented as a number of mana points or as an amount of spells (and sometimes, as in the d20 system, the actual spells).

Mana consists of an often large well that is drained when casting magic. Spells remove a certain amount of mana points from the well depending on the type of spell. Spells are therefore highly analyzed in terms of their mana efficiency. The well becomes deeper as the player attains levels, granting more mana. New spells become available upon advancement and usually cost more mana as well. The mana well may be slow or quick to replenishing itself. Another approach seen in some games is to hold the mana well constant in capacity and simply change the amount of mana used by spells. A third and final approach used by the game "Guild Wars" introduces a thermometer like system where the mana well is completely drained from the start but as the player begins to battle, it slowly starts to replenish. The well can be drained by using skills that require the built up mana stored within it.

Note that the refilling of a well need not happen constantly, some systems for example require that the character get a full night's rest before the well is replenished. This system is usually called a "fire and forget" system to reflect that the skill has been forgotten until complete bed-rest has been achieved.

Skill Based Systems and Class Based Systems

A skill based system provides a player with the choice of which skills his character will posses. This gives the player a considerable amount of freedom in shaping his character. The skill based system is therefore considered a very flexible system as opposed to a class based system. Class based systems divide characters into archetypes that are defined through skill sets. For example, choosing to play a fighter as one's class will provide the player with skills that the developers found appropriate when defining what a fighter is. There are a number of pros and cons for each system. The list below reflects the pros and cons of a skill based system.

- Skill systems give players more choices and freedom over the character development process. Characters have a larger degree of individuality and specialization.

- Advancement is often not tied to a leveling system33 but is a part of the skill system itself. The more one uses a given skill, the better one becomes at it. This process is often "character development".
- Skills that the player finds of little use, or becomes bored with may often be replaced by other skills.
- Skills systems where a maximum number of skills have not been set results in a "the more you play, the better your character will become" system.
- There exists a tendency of a large majority of players choosing the same skill sets, thereby almost creating a class based system.34
- The power curve in a skill based system is difficult to steer. While de/empowering a skill in a skill based system may affect the unbalanced characters, it will also affect the balanced ones.

The advantages and disadvantages of the class based system are mentioned below.

- Classes usually come with a package of skills that define the character through the class chosen. The player cannot choose freely among the skills available in the game.
- A class is often easier to balance since these skills are often unique to the class. Empowering a class skill has no effect on other classes.
- The class system has historically been bound to a class and level system [Hallford, 2001] meaning that balance can be improved through having skills and abilities become available at appropriate levels. The class system is often called a character advancement scheme.
- A class based system is appealing to developers who create challenges for specific roles and wish cooperative play among their players35. In order to overcome a challenge, a mixture of roles might be required, whereas in a skill based system, the roles are foggy and one character might end up being a master of all trades.
- Weaknesses include the leveling system. A lower level character of the same class will have little to contribute to others since the higher level character will in all aspects (with the exception of gear, although not likely. See section 1.9 in [Alexander, 2005]) be a better choice. This is due to the fact that the differentiation between characters of a class is usually small in class based systems.
- Once a class is chosen, the player is stuck with the skill set provided by that class for the rest of the game.

There are many different and interesting solutions to some of the drawbacks mentioned above for both class and skill systems. For example some games incorporate both methodologies to certain degrees. A class might give access to the weapons that are available to the character but it is the skill based system that determines one's proficiency with the weapons.

[33] See the section "Advancement" in the analysis for more on leveling.

[34] Although "Diablo 2" does not classify as being a skill based system it does contain a skill tree that highly defines what your character is capable of. The following link is an example of how players define classes through the attainment of certain skill sets. http://www.battle.net/forums/thread.aspx?fn=d2-skills&t=125048&p=1&tmp=1#new125048

[35] Cooperative play among players in MMORPGs is one of the best ways to create a community of players and keep players playing the game [Alexander, 2005].

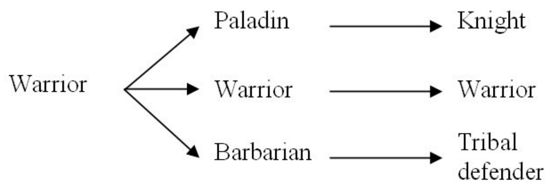


Figure 8: The warrior has two prestige class choices upon advancement, or he can continue on the path as warrior. Notice that classes can be forced to take another class as seen upon the advancement of the paladin or barbarian.

A multi class system has also been introduced as a solution to some of the class based problems. During advancement, a character can change class and thereby gain the skills from another class while losing the advancement gains he would otherwise have received by advancing in his own class. The primary problem with this methodology is that it adds to the balance difficulty issue. The imbalance is often dealt with by introducing prestige classes instead. Prestige classes are not available during the beginning of game play, and often have a level and class requirement before one can choose it. Prestige classes often resemble the base class, but provide a different approach. This ensures that certain combinations of skill sets that are perceived as highly unbalancing are never available to characters.

Advancement

The easiest way to represent advancement in an RPG is to represent characters and their skills through values and statistics and increment these upon advancement. The increments might be predetermined for balance purposes, or a pool of increment points might be available for the player to set in attributes and skills (much like buying attributes and skills) to give the player the influence on advancement.

Experience

The advancement of a character usually occurs through the accumulation of enough experience points (xp or exp) resulting in a predetermined improvement of skills and attributes. This process may be accomplished through the accumulation of more skill and attribute points, new combat and spell abilities etc. Xp is usually awarded through the defeat of opponents or obstacles, finishing quests, and on some occasions through successful role-playing. The amount of xp gained is highly dependent on the challenge. As the difficulty increases, the reward for completion increases. To counter this, the amount of experience needed to gain new abilities typically increases as well, as the character's level increases. This encourages players to accept tasks that commensurate with their improved abilities in order to advance.

"Power leveling" is the biggest problem with the xp system that game designers face. Power leveling is known as the process of sustained fast leveling through strategic playing with the sole purpose of gaining xp as quickly as possible. This is often seen as cheating or manipulation of the game system for unintended results. In multiplayer games this usually refers to a high level character helping a low level character to defeat monsters that are far too powerful for the low level character to defeat otherwise. Examples of how to combat power leveling include rewarding a player based on their contribution to the completion of the task, rewarding a player dependent on the player's level, and setting a limit for the maximum amount of xp a player may receive from a given encounter.

Advancement systems may also discard an experience point concept. Instead, skill used over time will slowly increase the effectiveness of the skill. Some games classify skills into skill classes, and it is the skill class which increases over time as a function of its use instead of just one skill. Once a skill class has increased, all skills within that class increase in effectiveness.

Some systems require the character to burn xp in order to gain advancement in skills. This approach is a combination of the experience concept and the skill use concept.

Combat

An analysis is now conducted on the combative features in role-playing games.

The Combat Round

Since combat is fairly complex, it is often broken down into combat rounds representing from one second of combat in some systems to ten seconds or even one minute in others. The round is a cyclical event and keeps occurring until combat is disengaged. To represent who goes first in a round, many P&P systems use an initiative value that may be random, weapon based, derived from an attribute such as speed or dexterity, or a combination of these. Many MMOPRGs use the reaction speed of the player as the reaction speed of the character. Therefore initiative is represented by the players speed at reacting to or initiating combat. The same goes for the concept of surprise.

Attacks are often broken down into the average number of attacks per round. This number is sometimes dependent upon a character's combat prowess, attack speed, the weapon's attack speed and the type of attack. Usually it is the weapon's attack speed that represents the attack speed of the character. For example, say a round is set to represent five seconds. Then a dagger might have an attack speed of 8 (attacking eight times per round), while a two-handed sword might have an attack speed of 2. There are three traditional forms of attack; Grappling, melee and ranged. Melee and ranged weapons have **reach** representing the range at which they can be used. Note that natural weapons, such as claws and fists, are considered melee weapons unless they can be used with range. To initiate a grapple the target must usually[36] be in natural weapon reach of the initiator.

Hitting and Missing

The most basic part of combat is the calculation of hitting a target or missing it. There are a wide variety of methods to determine whether your character is hitting the opponent or not. Before the methods are analyzed, it is important to understand what types of factors are included in the methodology. Common hitting and missing factors are summarized below. Most of the factors act as either bonuses or penalties to a random roll.

- **Level of proficiency with the weapon** sometimes represented via class, level and weapon skill
- Difference in **size** of target and attacker
- **Attributes** (often strength and dexterity)
- The **environment** (cover, higher ground, weather…)
- Target and attacker **situation** (position and facing, crouching, prone)

[36] Weapons and spells may sometimes initiate grapples.

- **Attack style** or **combat ability**
- Target's **defensive style** (trying to dodge, or parry)
- **Magical property** or **quality** or **keenness** of attacker's weapon or armor of target
- **Weapon type**, some weapons are easier to hit with than others
- The greater the **range**, the more difficult it will be to hit
- Taking time to **aim**
- **Target armor** (shield included)

The Equation

Most games use their core mechanic in order to determine a hit or miss outcome. The attacker will usually have an attack factor based upon his level, class, skill, weapon proficiency, or a combination of these while the defender will have a defense proficiency or a DC determined in somewhat the same way. There are three main approaches to how the equation could be structured.[37]

The first method involves the opponent containing a DC, and the attacker conducting the random roll to beat the DC. Here the DC is set via factors such as armor, dodge chance, parry chance, speed, facing and so on. A random value is rolled and bonuses such as situation, aim, range and so forth are added. The results are compared with one another, and if the attack roll is higher than or equal to the DC, then the attacker has scored a hit (An example of this method has already been shown in the analysis section under "Core Mechanic").

The second case is almost the exact same process as above except that the DC is set from attacker values instead. For example, the DC may be set according to how difficult the weapon being used is, or how good the attacker is at fighting in melee combat. Once the result has been rolled, the defender is entitled to a defensive roll if the attacker scored a hit. The defender must beat a defensive DC, set via some of the defender's values, and may attain bonuses or penalties to the roll. If the defensive roll is a success, then the attack has missed. Since the defensive and offensive capabilities rely solely upon the defender and attacker respectively, a defender will defend equally well against any attacker, as will the attack roll of an attacker against any defender. In other words if your character is really good at defending, then it does not matter how good the attacker is at attacking since he cannot influence your defensive capabilities.

The third case sets the DC according to both attacker's and defender's combative prowess. The DC often takes into account, the defender's chance to dodge, parry or block the attack and the defender's armor. The attacker rolls a random value and adds any bonuses or penalties to the result and compares it to the DC.

This third case can be interpreted as a modification of the first in terms of its results. It merely sets factors into a system that determines a DC and then adds the bonuses **afterward,** to determine a new DC. To illustrate this, let us use the d20 system as an example of system 1. A d20 die is rolled and used as our random factor. The attacker has bonuses and penalties totaling +9 to the random roll and the defender has a DC set at 24. Therefore the attacker will hit on a 15 (24 – 9) or more when rolling the die. The chance to hit percentage (30%) can be found via the following formula.

[37] From study of a multitude of pen and paper RPG systems, almost all of the systems fall into one of the three categories being described.

$$\frac{21-(DC-BP)}{20}\cdot 100\% = CTH$$

Where
DC = Difficulty Class
BP = Bonuses and Penalties
CTH = % Chance to Hit

In the d20 system if the roll plus bonuses and penalties is equal to the DC, the roll is considered a success. When subtracting (DC – BP) from 20, that specific roll will not be taken into account,[38] therefore 20 is increased by 1. Thereafter, the numerator is divided by 20 to get a % chance to hit.

Let the third system represent a percentage system built mirroring the d20 system. The algorithm needed can be found using the algorithm above.

$$\frac{21-(DC-BP)}{20}\cdot 100\% = CTH$$
$$\Updownarrow$$
$$\left(\frac{21}{20}-\frac{DC}{20}+\frac{BP}{20}\right)\cdot 100\% = CTH$$
$$\Updownarrow$$
$$\frac{21-DC}{20}\cdot 100\% = CTH - \frac{BP}{20}\cdot 100\% = CTH'$$

In the case above, the value 21 can be interpreted as the Attack Rating of the attacker while the DC interpreted as the Defense Rating of the defender. Recall that system 3 sets a DC according to a factor from both attacker and defender (In this case the attack rating and defense rating). The DC will be a percentage chance to hit where bonuses and penalties are applied after the system has found the chance to hit. With these small adjustments the formula is now

$$\frac{ATTR-DEFR}{20}\cdot 100\% = CTH'$$

and

$$CTH = CTH' + \frac{BP}{20}\cdot 100\%$$

Where
ATTR = Attack Rating
DEFR = Defense Rating
CTH = Chance to Hit
BP = Bonuses and Penalties

So, if we use the same example values as we did in system A, the attacker now has an attack of 21 and the defender has a defense of 24. System C gives us the result of a **negative** 15%. Luckily our attacker has a bonus of +9 which is added after the calculation. +9 is the same as 45% and the attacker now has a –15% + 45% = 30% chance to hit.

[38] (15, 16, 17, 18, 19, 20) should all hit, but 20-15=5 and there are six numbers in the set.

As stated before, the chance to hit may contain a level based factor when dealing with a level based advancement system. This level modifier is used to encourage players to deal with tasks that are suitable for their level[39].

Damage

Once a hit has occurred, damage is inflicted upon the target. Damage may be dependent on the attributes of the character (such as the attacker's strength), but is mostly dependent on the weapon being used. Otherwise, only magic plays a role in damage. Damage is usually a whole number. An example of damage is a short sword doing 10 to 12 damage.

Damage Reduction

Physical damage received during combat is often affected by armor. Armor helps reduce damage, and this concept is known as damage reduction. Damage reduction is expressed as a percentage score gained through use of armor, or magic that acts as armor. Armor is therefore valued in terms of its reduction capability through an armor rating value. The better the rating is, the better the reduction. Damage reduction is equivalent to decreased incoming damage. There is usually no randomness included in the algorithm for damage reduction.

For an example of damage reduction, let us view the "World of Warcraft" system. The algorithm is presumably

$$\frac{AR}{AR + TL \cdot 85 + 400} = DR$$

Where:
AR = Armor Rating
DR = Damage Reduction
TL = Target Level

The target's level is taken into account to ensure that higher level targets have an easier job of doing damage. It also ensures that players will always want to upgrade their equipment. Often there is a limit set for damage reduction so that a character cannot become impervious to damage. The graph shown below is based on an input of 10000 damage per second (dps) and a target level of 60. It helps visualize the above equation.

[39] See the "Advancement" section for more on why this is important.

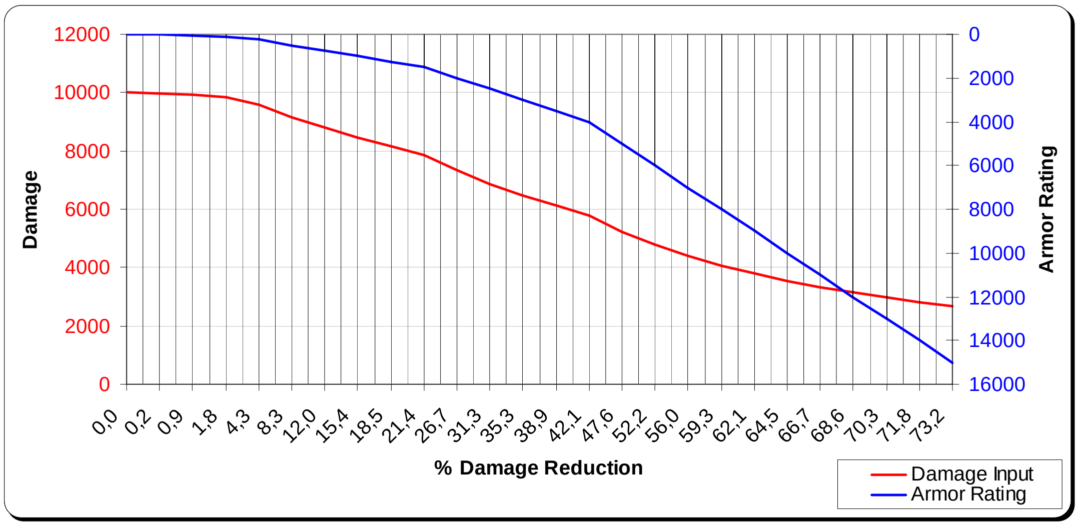


Figure 9: The graph illustrates how damage reduction works in the game "World of Warcraft".

An armor rating of 2000 for example would result in a DR of about 26% (the blue line). The same armor rating would decrease the damage input from 10000 to around 7500 (follow the vertical black lines back to the red series, which depicts the damage done).

A point worth making is that an increase in damage reduction does not mean the same increase in damage received. Consult the table below.

From armor value to armor value	Extra armor reduction %	Reduction % in dps	Comparison of armor reduction % and reduction in damage %
1000-2000	11.28	13.33	1.18
2000-3000	8.63	11.76	1.36
3000-4000	6.81	10.53	1.55
4000-5000	5.51	9.52	1.73
5000-6000	4.55	8.70	1.91
6000-7000	3.83	8.00	2.09
7000-8000	3.26	7.41	2.27
8000-9000	2.81	6.90	2.45
9000-10000	2.45	6.45	2.64
10000-11000	2.15	6.06	2.82
11000-12000	1.90	5.71	3.00
12000-13000	1.70	5.41	3.18
13000-14000	1.52	5.13	3.36
14000-15000	1.38	4.88	3.55

Figure 10: The table is based upon a target level of 60 in the game "World of Warcraft"

For example an increase in armor value from 5000 to 6000 will result in 4.55% extra damage reduction (52.17% – 47.62% found via the algorithm above). But the actual damage being received will lower 8.7% which is around 1.91 times more in comparison.

Damage Resistance

Armor also has the capability to completely block some damage, especially if one is equipped with a shield. A good example is a kitchen knife used to attack a fully plated armor. The knife will simply not get through in most places. This concept is called damage resistance or the hardness value of armor. Since a block of total damage can be interpreted as a miss, some algorithms take damage resistance into account as a variable when calculating the chance to hit. Others simply add a value upon the armor or shield that is subtracted from any damage caused.

Saves is another type of damage resistance. They are rolls against a character's characteristics or rolls against a subsystem of scores (usually called saving throws). They represent a resistance to damage caused indirectly such as from an explosion or though magic.

To give an example of saves, the d20 system might have the best general saving throw system there is. Here saving throws have been dealt into three classes: fortitude, reflex and will. The system reference document describes the throws as:

- Fortitude: This save measure your ability to stand up to physical punishment or attacks against your vitality and health. Apply your Constitution modifier[40] to your Fortitude saving throws.
- Reflex: These saves test your ability to dodge area attacks. Apply your Dexterity modifier to your Reflex saving throws.
- Will: These saves reflect your resistance to mental influence as well as many magical effects. Apply your Wisdom modifier to your Will saving throws.

The formula which uses the saves is again based on the core mechanic of the system.

Success if	$d20 + \text{resistance score} \geq DC$
Failure if	$d20 + \text{resistance score} < DC$

In cases where a saving throw is more logical to take against an attribute, the d20 system provides rules such as a strength save, or intellect save. Saves are tied to skills so that a save is taken only if the skill allows a save.

Spell Resistance

Spell resistance is simply an extra type of damage reduction or damage resistance that works against spells. With spells that are all or nothing such as "fear" or "entangle" spells, spell resistance either blocks the spell or does not.

Hit Points

Each time a target is hit, and damage is caused, an amount of damage is deducted from the target's health. In RPGs health is usually called hit points and hit points are often represented as a value. Any damage obtained reduces the value, while healing obtained increases it, although not beyond its capacity. Higher level characters have a larger amount of hit points to represent their greater battle experience. In a sense, these characters are to be thought of as containing the

[40] Constitution, dexterity and wisdom are ability scores in the d20 system

exact same amount of life as a low level character, and that they are better at dodging and moving out of the way when damage is inflicted (A potentially lethal hit on a low level character would result in a scratch on a high level character).

The main problem with this methodology is that most people do not accept it as being realistic.[41] The argument is that any type of combat should be potentially dangerous. The solution has been to introduce a wound system behind the hit point system. Hit points now represent a buffer of non lethal damage, and wounds have taken on the role of representing the character's actual life. A character usually has a low amount of wounds compared to hit points and wounds often do not increase. As an example, a character might possess 30 wound points and 4000 hit points. Lethal damage reduces wounds while non lethal damage reduces hit points. Most damage is considered non lethal, but after having attained a certain amount of non lethal damage, the damage becomes lethal. Therefore, once a character's hit points have been reduced to 0, wounds are reduced instead. Critical hits from weapons might reduce a character's wounds instead of hit points and certain weapons or skills may pierce through one's hit points and reduce wounds as well. This ensures that certain weapons will always kill a character, no matter the amount of hit points.

Finally, some rule systems contain a dying buffer, so that after having lost one's hit points or wounds, the character is in a dying state, usually with only a few seconds to live. This gives an extra chance for the character's comrades to react.

Critical Hits and Fumbles

The rules for critical hits and fumble rolls are mentioned in the Automatic Success and Failure section in the core mechanic analysis. An automatic success in combat will often result in a critical hit which can result in more damage done or damage to ones wounds instead of hit points. An automatic failure in combat might result in a mishap such as the loss of the next strike, or a chance to fall prone.

Aggro

Aggro is English slang for aggressive or violent behavior. In MMORPGs aggro is used as a term to express the anger level of a "mob" on a certain character. Mob is short for "mobile object" and is the popular term given used to describe the non player opponents found in games. Aggro is a form of artificial intelligence. It adds a level of strategy to play, enforcing the roles of characters.

With the addition of aggro, mobs now switch targets depending on how angry they are at the target. Anger may come from different sources but the best ways to make a mob angry is either to taunt it, damage it, or heal someone currently engaged with it. Because aggro only affects the switching of targets, aggro is only interesting in team based games.

Aggro is usually a list with values describing the current amount of anger with each aggressor the mob is in combat with. The list is used so that the aggressors with the highest values are the ones that the mob will be attacking. The values on the list are obtained through the amount of damage caused, healing cast and threatening abilities used.

The easiest way to further explain aggro is with an example of its use. Imagine three characters working together in combat versus a strong mob.

[41] This is of course not a problem in systems that are not meant to be realistic.

	Mob	Tank	Healer	Glass Canon
Health	30000	4500	3000	1500
Normal damage per sec	700	300	200	200
Spell damage per sec	0	0	150	400
Damage reduction vs. normal damage	50%	75%	20%	10%
Damage reduction vs. spell damage	20%	75%	20%	10%
Extra abilities	None	Can taunt every 5 seconds, instantly causing 200 threat.	Can heal 300 damage per second instead of dealing damage.	Can add 200 spell damage to an attack every 5 seconds

Figure 11: An example of values defining three characters and a mob. Characters are illustrated in blue text while the mob is in red.

The mob's values are visualized in red text whereas the characters' are visualized in blue. The first character is the "tank", containing a large number of hit points and a good amount of damage reduction. The second character is the "healer" containing a low amount of armor and an average number of hit points. The last character is the "glass canon" (GC), with almost no armor and no hit points but a high amount of damage output. The mob is set with values to ensure that it is a formidable adversary.

Round	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Tank aggro	350	500	650	800	950	1300	1450
Healer aggro	100	400	700	1000	1100	1400	1700
GC aggro	480	800	1120	1440	1440	1440	1440
				Health			
Tank health	4500	4500	4500	4500	4500	4500	4500
Healer health	3000	3000	3000	3000	3000/2440	2740/2180	2480/1920
GC health	1500/870	1170/540	840/210	510/0	0	0	0

Figure 12: The table is an example of how an aggro list is generated and incremented each round in combat. The aggro list affects the targeting of the mob as can be seen via the loss of health on certain characters (red text). Blue text illustrates a rise in health.

The table above is an example of one of the possible outcomes for combat between the mob and the characters. Rounds are set to 1 second intervals with the characters attacking first and the mob last. Aggro towards the mob is caused through damage and healing where the following rules apply.

- 1 point of damage inflicted = 1 point of aggro towards inflictor
- 1 point of healing cast = 1 point of aggro towards caster

Without any tactics set, the players simply decide to attack the mob and are therefore first in initiative. Both the tank and GC choose to use their extra abilities creating the aggro list as seen above for round 1 (The tanks threat = 200 threat damage from his ability + (300 · 50%) from damage done taking into account the damage reduction of the mob). The Mob is also allowed an attack in round one. Consulting the aggro list, he realizes that he is most angry with the GC. The GC's health is reduced from 1500 to 870.

In round 2, the tank attacks, the healer realizes that he will need to heal, and the GC continues his attack confident that the healer will take care of him. The GC's health is therefore incremented to a value of 1170. Unfortunately for the GC, he still retains the highest spot on the mob's aggro list, and the mob chooses the GC as his target, reducing health from 1170 to 540. This process continues until the beginning of round 5 where the GC has died in round 4.

In round 5 the healer chooses to continue his attack, but is surprised to see that the mob now chooses him as the target. This will force him to heal in round 6.

In round 6 the tank has his taunt ability ready and uses it, but unfortunately it, together with his attack, cannot produce enough aggro to gain the anger of the mob. As long as the healer keeps healing himself, the tank will never gain aggro, and the battle will be lost.

Combat Advancement

Advancement in combat is usually highly influenced by the advancement of skills, attributes and equipment. The typical combat influences are depicted in the table below.

Equipment*	**Skills***	**Attributes**
Damage	Heighten combat abilities	Hit points
Damage reduction	Gain more combat abilities	Damage
Resistances		Chance to hit

Figure 13: Typical influences in combat advancement from three major role-playing aspects. *Equipment and skills typically can influence the rule system freely and therefore might affect any part of the combat system. Only their typical effects have been listed.

The chance to hit and the resistance equations are almost always affected by an advancement of the character as well. The increment in a variable is naturally dependent on the formula but if we take a look at one of the formulas, for example the one that was based on both attacker and defender, namely:

$$\frac{ATTR - DEFR}{20} \cdot 100\% = CTH'$$

and

$$CTH = CTH' + \frac{BP}{20} \cdot 100\%$$

Where
ATTR = Attack Rating
DEFR = Defense Rating
CTH = Chance to Hit
BP = Bonuses and Penalties

then the attack rating and the defense rating of a character would be the values to increment.

Hypothesis

The hypothesis introduces a possible solution to a generic based combat system. A discussion of the choices made in the hypothesis follows in the "Discussion" section of the report.

The Core Mechanic

The core mechanic is based on a percentage system ranging from 0.000% to 100.000% (three significant decimals). Challenges are set according to different formulas but always result in a percentage value where the objective is to roll lower than the value (endeavor is a success).

Unless otherwise noted all combative values are positive decimal numbers and with a maximum decimal place in the thousandth's position. In the case where a number should exceed this position, the number is rounded to the thousandth position.

Character Attributes

Eighteen attributes define the character. The attributes are arranged in three classes; mental, physical, and spiritual attributes. Attributes are always positive whole numbers or zero. A value that is reduced to a negative value is automatically set to zero. Attributes that affect resistance ratings have their affect multiplied by the resistance rating, while all other attributes benefit the combat system through an addition of their affects.

Mental	Physical	Spiritual
Intellect	Might	Energy
Wit	Agility	Luck
Knowledge	Fatigue	Insight
Reason	Vitality	Willpower
Concentration	Hit Points	Precision
Sanity	Appearance	Character

The Mental Attributes

Intellect: Intellect governs the character's level of understanding which in turn affects the damage output of intellectual skills.

1 point in intellect = +0.1 damage per second.

Wit: Wit governs the keenness and quickness of perception. It influences the amount of time needed to learn something new such as a new skill.

1 point in wit = +0.025% of increase in learning speed
1 point in wit = +0.025% reflex resistance

Knowledge: Knowledge reflects the experience and educational level of the character. Knowledgeable characters may know information concerning quests, areas, lore, magical items and so on. An example of its use might include the whereabouts of a magical item in a certain area. Without a certain amount of points in this skill, a character would not be able to gain knowledge of the item, and therefore not be able to find it.

1 point in knowledge = 1 knowledge
Knowledge is a value that is compared to a DC

Reason: Reason represents the character's ability to reach the correct decision. A character with a high score in reason should be able to "see" a path to the correct decision better than a character with a lower score could. An example of its use; A high reason attribute shows the exact amount of hit points of a mob enabling the player to make a better decision of whether he should engage the mob or not.

1 point in reason = 1 reason
Reason is a value that is compared to a DC

Concentration: Concentration is an attribute tied to skills that require concentration to use. When using a skill that requires your character to concentrate, there is a chance that the skill may be interrupted by a certain amount of time. The chance may be reduced via the concentration attribute.

1 point in concentration = 0.05% chance of decreased interruption chance

Sanity: Sanity represents the character's ability to remain cool and sane of mind. Sanity resembles hit points since it is a value that may be decreased and increased. A sanity value of zero represents insanity.

1 point in sanity = 1 sanity

The Physical Attributes

Might: Might is the value that represents how strong a character is. It affects the amount of physical damage done, and the amount of damage that may be absorbed by a shield.

1 point in might = +0.1 damage per second
1 point in might = +0.05 absorbed damage

Agility: Agility represents the nimble body movements of the character. Agility affects the evasive ability of the character, the chance to block with a shield and the reflexive resistance of a character.

1 point in agility = +0.035% in increased shield block chance
1 point in agility = +0.035% in increased dodge chance
1 point in agility = +0.025% reflex resistance

Fatigue: Fatigue represents how quickly a character becomes tired. Skills may be tiring to use and therefore reduce fatigue by a certain amount of points upon their use. Once fatigue reaches 0, your character becomes exhausted decreasing your chance to hit, and your damage output highly, and increasing the opponent's chance to hit.

1 point in fatigue = 10 fatigue
Exhausted = 25% penalty to hit, 50% damage penalty, 25% bonus to hit for opponents

Vitality: Vitality is the value which represents the life points and health of your character. If vitality reaches zero or less, the character is considered dead. The chance to resist mal effects of the body is also influenced by vitality. Furthermore, a loss of ones health is extremely tiring.

1 point in vitality = 1 vitality
For each loss of 20% of one's vitality = a loss of 20% fatigue occurs
1 point in vitality = +0.05% hardiness resistance

Hit Points: Hit points represent a buffer of scratches and smaller wounds as well as your character's battle prowess. Once the buffer reaches 0, vitality is subtracted instead.

1 point in hit points = 10 hit points

Appearance: The overall look of your character is represented by this value. A better score in appearance may result in better rewards and quests. Areas that are closed to others may be open to you.

1 point in appearance = 1 appearance
Appearance is a value that is compared to a DC

The Spiritual Attributes

Energy: This attribute simply represents the mana value of a character. It is an energy source that all characters may be in touch with.

1 point in energy = 10 mana

Luck: Luck presents a bonus of an increased chance to find magical equipment.

1 point in luck = 0.05% chance of a 5% bonus to finding magical equipment

Insight: Your spirit affects the physical and magical regeneration rate of your body. Magic is closely tied to one's spirit. The proclivity that a magic effect will procreate on an item[42] is directly influenced by this attribute as well.

1 point in insight = +0.01% proc.
1 point in insight = +0.5 recovery rate in mana and hit points per game tick

Willpower: All characters contain a measure of will and certain situations, especially during combat, might call upon a test of one's will.

1 point in willpower = +0.05% resolve resistance

Precision: Precision affects the chance to hit with certain weapons and also affects the critical probability of both spells and weapons.

[42] In RPGs this is known as the chance for the item to "proc". For example, if an item has a 5% chance of casting a fireball when striking an enemy, then this chance is known as its "proc" chance.

1 point in precision = +0.02% critical hit chance

Character: The overall personality of your character is represented by this value. A better score in character may result in better rewards and quests. The hostility of mobs may be influence by this value. Areas that are closed to others may be open to you.

1 point in appearance = 1 appearance
Appearance is a value that is compared to a DC

Combat

Characters are always in one of the two states; "*in combat*" or "*out of combat*". To initiate combat a character must *threaten* the opponent. Threatening may be as simple as left clicking the mouse button or pressing the space bar while targeting, to having to take tests of will in order to see if your character will engage in combat. The defender is first considered in combat once a mal effect has occurred, unless the defender chooses to threaten the attacker as well. Certain skills may also initiate combat.

Changing an "*in combat*" state to an "*out of combat*" state requires that

- one side has lost the combat
- both sides agree to disengage combat
- both sides become out of range of one another for 10 seconds, where out of range is set by the developer
- an effect occurs that puts the character out of combat (e.g. stealth, teleportation)

There are certain actions available when in combat. All of these actions are expressed through skills, except the movement of the character which always is available (unless the movement of a character has somehow been impaired). The most common action is the "attack target" action, which will attempt to hit the target with any weapon currently equipped. If there is no weapon equipped, the character will attack using natural weapons, such as his fists. The "attack target" skill is the only skill that is guaranteed to be available to all characters by the combat system.

When *in combat* one's recovery rate in hit points from the spiritual insight attribute drops to 0%. The recovery rate of ones spiritual energy drops to 5% as soon as the spiritual energy buffer has been used whether in or out of combat. It will rise to its original rate if the spiritual energy buffer is untapped for 5 seconds.

Hitting the Opponent

To resolve when a hit or miss has occurred, characters possess two levels of proficiency with each weapon type in the game (e.g. swords, guns, bows). The first level of proficiency reflects the level of attack with the weapon type while the second proficiency reflects the level of defense with the weapon type. The level of proficiency may range from zero representing a character that cannot use the weapon class, to the character's level times 10 plus a bonus value from skills, magic and so on.

The character's weapon attack proficiency and weapon defense proficiency are the most important factors when determining the chance to hit. They are represented as whole numbers and have a range of 10 points per level of the character. The base chance to hit is set to 90%, and a bonus or penalty depending on level of proficiency of 0.1% is given for each point of difference in attack and defense. Note that the chance to hit may also be influenced by combat skills and equipment representing a miscellaneous bonus or penalty.

$$CtH = (90\%+(AWP - DDP)\cdot 0.1\%)+Misc$$

Where:
AWP = Attacker's Weapon Proficiency
DDP = Defender's Weapon Defense Proficiency
CtH = % Chance to hit
Misc = Miscellaneous Bonus or Penalty

To discourage power leveling, a slight difference in level will severely lower the chance to hit. The below formula is only used if a **mob is the target** and is **two or more** levels above the character. The formula lowers the chance to hit by 1.5% for each difference in proficiency level beyond 20 (representing the two levels).

$$CtH = 89\%+((AWP - DDP)+10)\cdot 1.5\% +Misc$$

Although the penalty does not apply when attacking other players, a superior level character will have a bonus to defense while fighting lower level characters. To encourage player vs. player combat and still take steps to provide a higher level character with greater combat expertise, the penalty in level difference has been increased to a difference of more than 5, and a penalty of 1% per point of proficiency.

$$CtH = 85\%+((AWP - DDP)+50)\cdot 1\%+Misc$$

The formulas are proficiency dependent, making it important to keep one's weapon proficiencies up to date with one's level. To add an automatic success and failure roll, the minimum hit chance is set to 2% (a roll of 98 or more) while the minimum miss chance is set to 1% (a roll of 1 or less).

The formulas above can be interpreted as being outcomes in a fair discrete *Bournoulli trial*[43]. Let $P_n(H)$ denote the probability that a hit event will occur in $_n$ number of trials. And let $P_n(M) = 1-P_n(H)$ denote the probability of the event where a miss is the outcome. If the outcome of a trial results in a hit then $P_n(H)$ could furthermore represent the following traditional set of outcomes {basic hit, critical, dodge, parry, block} so that

$$P_n(H) = P_n(bh)+P_n(c)+P_n(d)+P_n(p)+P_n(b)$$

Where:
bh = "basic hit" and c = "critical hit" and d = "dodge"
and p = "parry" and b = "block"

To illustrate this, the following table contains the Bournoulli hit or miss trial. Should a hit occur, then the outcome is still not decided and may result in any outcome within $P_n(H)$.

Hit	Miss

[43] A Bournoulli trial is an occurrence in which exactly one of two possible outcomes can occur. [Petruccelli, 1999]

Hit	Critical	Dodge	Parry	Block	Miss

Although dodge and parry outcomes exist in the $P_n(H)$ set, they result in the attacker "missing" the target. A block is not considered a miss because it usually results in the defender being damaged. See the "Blocking, Parrying and Dodging" section.

$P_n(H)$ is a set of exclusive events represented via percentages that sum to 100%. All combatants possess a percentage chance to block, parry, and dodge an attack as well as a critical hit chance. These chances influence the basic hit chance calculated as follows:

$$P_n(bh) = 100\% - (P_n(c)+P_n(d)+P_n(p)+P_n(b))$$

Note that the critical chance is the **attacker's critical hit chance** while the dodge, parry, and block chances are the defender's chances. The opponent's block, parry and dodge sum is limited to 50% to ensure that there always exists a chance to hit or to critical hit. The excess is reduced from dodge, parry and block equally. The rules governing dodge, block, parry, critical hit and hit are:

1. Dodge, parry and block take precedence over hit and crit
2. Dodge, parry and block sum may not exceed 50%, excess chance is reduced from all equally
3. Critical hits take precedence over hit.

To illustrate the model, imagine the defending character with a 10% block chance, 10% parry chance, and a 20% dodge chance. The attacker has a 10% critical hit chance so the model would result in the figure 15 weighted diagram. To illustrate the three laws in use, figure 14 uses a defender with 15% block chance, 10% parry chance, 30% dodge chance, and an attacker with 58% critical hit chance. Notice that the basic hit chance in figure 15 has been reduced to 0% because of rule three.

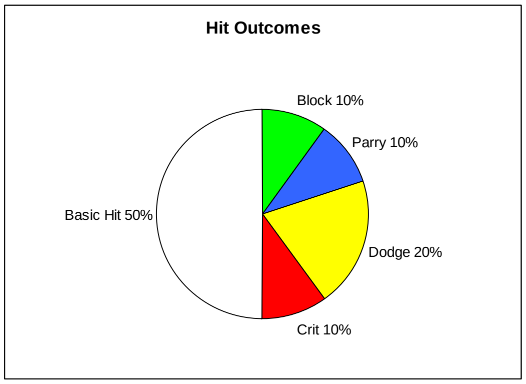


Figure 15: $P_n(bh) = 100\% - (P_n(c)+P_n(d)+P_n(p)+P_n(b))$

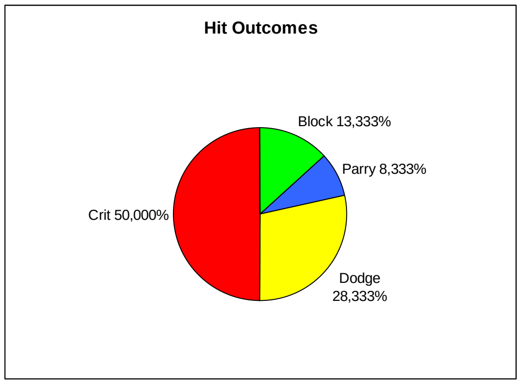


Figure 14: The critical hit chance has taken precedence over the basic hit chance but parry block and dodge have the highest precedence.

The base chances to critical hit, dodge, parry or block an attack are set to 4% for each outcome. Dodging and performing a critical hit are abilities that are available to all characters while parrying and blocking are skills that characters can learn.

Weapon Proficiency

Weapon proficiencies have already been mentioned in the "Hitting the Opponent" section. Weapons are grouped into interlocking weapon categories where weapons usually share some forms of familiarity. The categories are up to the developer to create. These categories pertain to what training is needed to become proficient in a weapon's use. For example a proficiency in the set of *swords {short sword, long sword, two handed sword, rapier, broadsword, bastard sword, lionheart sword, crusader sword}* will result in proficiency in all items within the set.

For each weapon class that exists in the game, a character has a set {attack, defense} of weapon proficiencies for that class. Initially a character has almost no knowledge of how to use a weapon class resulting in a weapon proficiency level of 1 in both attack and defense. If the proficiency level is set to 0, the character cannot use the weapon class. As the character uses a weapon, there is a chance that he will gain a level of proficiency with the weapon, resulting in more efficient combat as seen by the hit/miss formulas presented in the "Hitting the Opponent" section of the hypothesis. The chance to gain proficiency in a weapon category is influenced by one's *wit* attribute. A miscellaneous modifier exists as well in case equipment, race and so forth influence the gain in proficiency.

$$\text{Chance to gain a point each strike} = 1\% + wit + (level \cdot 10 + misc - CPL)$$

Where CPL =Current Proficiency Level

The maximum attainable proficiency level is set to *10· character level + miscellaneous modifiers* such as a racial modifier. The formula ensures that a high level character will easily learn new proficiency levels in sets that are new to him, but as the proficiency level reaches the maximum attainable level, the rate of gain will fall. Note that the attack and defense proficiency are not related and both use the formula. Therefore, one might raise one of the proficiencies while not raising the other.

Overall Attack and Overall Defense

The overall attack and overall defense of a target are two characteristics that determine the overall attack and overall defensive capabilities of the character. They are not related to the weapon proficiencies mentioned above although they use the same formula in terms of gaining points. The values are whole numbers and have a range of 10 points for each level of the character.

The overall attack and overall defense of a character affects the critical hit, dodge, parry and block chances in the following manner. For each level of difference in skill between a character's overall attack versus the opponent's overall defense, a 0.015% bonus is awarded to the higher skilled character and a 0.015% penalty to the lower skilled. The value affects the dodge, parry, block and critical hit chances in the following manner.

- In the case of the character having a higher level of overall attack vs. the opponents overall defense, the attacker receives the value as a bonus to his critical hit chance and lowers the defender's parry, block and dodge chances by the same value as a penalty.
- In the case of the character having a higher level of overall defense vs. the opponents overall attack, the attacker receives the value as a bonus to his parry, dodge and block chances while lowering the opponent's critical hit chance as a penalty by the same value.

All out defense and *all out attack* are two options that are available to all characters. These options increase either one's overall attack or overall defense depending on the option chosen by 10% of one's current point values in the item while lowering the other by 20% of its points. For example a character with 103 points in overall attack and 112 points in overall defense choosing an all out attack would gain 10.3 points in attack while losing 22.4 points in defense.

Blocking, Parrying and Dodging

Blocking requires an equipped shield. If an attack is blocked, the reduction in damage will be the *block value of the shield* and the added absorbed block damage of the *might value* of the shield user. One cannot block attacks that are performed from behind.

Parrying requires that a weapon be equipped, and one cannot parry an attack that is performed from behind. Dodging can always be accomplished.

Damage and Critical Hits

Successful attacks deal damage, and damage is based upon the attributes of the user, the type of weapon, and an eventual miscellaneous bonus or bonuses. The minimum amount of damage that can be caused regardless of damage reduction is 1, unless a block with a shield occurs, in which case the minimum is set to 0. Damage is always a whole number. Damage may be fixed or vary depending on the preference of the developer. Variable damage is set via the least and highest amounts of damage the weapon is capable of inflicting.

When you make an attack and get a *critical hit*, more damage will be dealt depending on the weapon or skill being used. The critical hit is a *multiplier* found on the weapon or skill. The multiplier affects the total damage caused to the target (including the *might* or *intellect* contribution of the attacker for example).

Weapon Speed

All attacks have a speed attribute stating how often the attack will occur. The weapon speed is expressed in seconds or some fraction thereof to a maximum of three decimal places. A dagger might have an attack speed of 0.6 seconds, while a heavy maul might have an attack speed of 3 seconds.

Range

All weapons possess a range, varying from melee weapons, any range within 5 feet, to ranged weapons, ranges of more than 5 feet. The opponent is either in two states, *in reach* of the character, or *out of reach.*

Ranged weapons are thrown weapons or projectile weapons that may not be effective in melee. Therefore most ranged weapons have a minimum range at which they can be used as well as a maximum range. Weapons may also possess range increments resulting in a decreased chance to hit when attacking over the increment. The chance is expressed via the weapon being used and through a percentage.

Facing

The facing of the character is important during combat. A character can only attack a target that he is facing unless otherwise noted (not all skills require that the character face the target). If attacked from behind, the character cannot block or parry the attack. Targets within a 180° arc are accessible to attack as seen below. If any part of the target is within the green area, the target may be attacked.

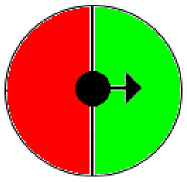


Figure 16: Facing the way the arrow is pointing, anything within the green area is accessible to attack.

Damage Reduction and Attack Types

All attacks have a certain attack type corresponding to one of the elements in the following table:

Mundane	Magic
Slashing	Fire
Piercing	Water
Bludgeoning	Air
Explosive	Earth
Resolve	Light
Hardiness	Dark
Reflex	

Figure 17: Attack types and resistance types found in the combat engine.

All attack types correspond to a resistance, meaning that the table also lists the resistances available in the combat system. Resisting an attack type can result in a damage reduction of the attack type or a Bournoulli trial result. Therefore, resistance and damage reduction use the same formula which is:

$$\frac{RR}{RR + TL \cdot 100} \cdot 100\% = DR$$

Where:
RR = Resistance Rating
DR = Damage Reduction (or Bournoulli chance)
TL = Target Level

Resistances are applied after damage has been done if used as damage reduction, or before damage is done if used as a Bournoulli trial.

Aggro

All mobs have an aggro list containing a numeric value for the threat of each player. This list is the mobs "hate" list. The rules governing the value on the hate list are as follows:

1. 1 point of damage = 1 point of threat.
2. 1 point of healing, fatigue or mana = 0.5 points of threat (incensement beyond the targets capacity (over healing) does not count).

3. Abilities may cause threat.
4. The target that the mob chooses to attack is the one with the largest amount of threat on the hate list unless:
    a. The mob is already attacking a target in which case one must exceed the target's threat by 15% to gain aggro.
    b. The target is out of the mob's range in which case the target must exceed 35% of the current targets threat.
    c. If the target cannot be reached (because of a skill, terrain, or an error as examples), the mob will choose the next target on the list.
5. The hate list is erased if the mob goes from "in combat" to "out of combat".

Combat Skills

Combat skills are what can make combat interesting. Combat skills can represent everything from spells to disarming your target. They can manipulate with any of the factors mentioned in the "Hypothesis" section including attributes. Unfortunately, combat skills are dependent upon setting (you cannot cast a fireball in a present day setting for example).

Analyzing the Outcome of Combat

To create an efficient analysis of combat, combatants will have to fight each other a number of times so that

$$P_n(W) = N_n(W)/_n$$

$$\lim_{n\to\infty} P_n(W) \to P(W)$$

the outcome of a combat resulting in a win will reach the probability of a win for that combat as the number of fights $n \to \infty$ approach infinite. For this purpose the number of fights conducted when analyzing combat is set to 100.

An outcome is analyzed in terms of

- the average number of seconds it took for one combatant to defeat the other
- the average damage per second that was inflicted
- the average number of misses and hits

The analyzer provides the functionality of incrementing a predetermined attribute one step every 100 outcomes, so that the affect of the attribute on combat can be viewed. If this functionality is chosen, combat is prolonged to 400 outcomes, enabling the analyzer to increment the attribute 4 times. The amount to increment is chosen by the user.

Discussion

The reasons for the choices made in the "Hypothesis" section are discussed below. Arguments take into account the design of some of the systems found in the analysis.

The Core Mechanic

Computers have no problems working with large numbers. A d20 or 3d6 core mechanic is fine for a table top game where smaller numbers are easier for humans to process, but for a computer, a core mechanic with a larger range of values is almost no different than a mechanic with a small range of values. There are developers though, who wish their systems to be transparent[44] to the player, and this creates a need for the system to be intuitively easy to understand for the player. [Alexander, 2005] Providing a percentage based system ensures that people can easily understand the system, and because the percentage system is based on three decimal places, it provides the developer with a fine degree of control.

The choice of a one die core system might seem disturbing since a multi die core system creates a modal bar. The modal bar ensures that characters will rely more on their skills rather than on a lucky roll. Consider this common scenario; the developer creates a game that is only combat based and tens of thousands of players will be using the combat system at the same time. Defeating an opponent in most games takes more than just one hit. [Alexander, 2005] Let $N_n(H)$ denote the number of trials that resulted in a hit and $P_n(H)$ denote the probability that a hit event will occur in n number of trials. Assuming that the random generator is fair, then

$$P_n(H) = N_n(H)/n$$

$$\lim_{n\to\infty} P_n(H) \to P(H)$$

So even if it may seem that luck plays a larger part in a one die core system, as the number of rolls approaches infinite $n \to \infty$ the relative frequency of the roll will approach the probability of the roll.

Combat systems are traditionally all or nothing systems, you either hit or you miss. An implementation of a performance bar could result in variable damage depending on how well the performance of the attack contra was performed. But weapons already have a variable damage output that represents this. Instead, a level of performance has been introduced through the weapon and defense proficiency system.

Finally, a multi die core system that, for example, requires three die rolls will be roughly three times slower than a one die core system when determining the success of an event.

Level Based System

The combat system is a level based system so that characters gain in combative skill upon an increase in level. Because the system is tied to an advancement methodology, both skill based and class based systems can be used with the combat system since both methodologies rely on

[44] Little to no information hiding

one of the basic elements of role-playing games; the advancement of a character. A skill based system is usually not tied to a leveling system, but this is not a problem since the leveling system can be transparent where mobs and players contain a hidden value summing their status in terms of their power level.

Surprise and Initiative

Notice that there are no attributes or formulas that deal with surprise and initiative. Since a computer can simulate much of what is being described by a game master in a P&P game, a player's reaction time in MMORPGs represents their character's reaction time in the game. If surprise is desired, it may be represented via a skill that, for example, only can be used while in stealth. The skill might temporarily lower the defense of the target for an amount of time, stun the target, or slow the target.

The Attributes

Mental	Physical	Spiritual
Intellect	Might	Energy
Wit	Agility	Luck
Knowledge	Fatigue	Insight
Reason	Vitality	Willpower
Concentration	Hit Points	Precision
Sanity	Appearance	Character

Eighteen attributes is about three times more attributes than most other systems. Twelve of these attributes[45] affect combat directly. The large number of attributes in the combat system gives players more choices and allows the players to create individuals that are potentially vastly different than the next player's individual. Because the combat system is completely concealed from the user[46], players will never have to reference or lookup a value, as is the case with unimplemented P&P systems. Therefore, having eighteen attributes is not much different than having six attributes. The developer has a greater degree of control with a larger number of attributes and therefore a better chance at balancing any imbalances. This is the same reason for not introducing derived values. Changing an attribute will often result in changes in all derived values.

The chosen design unfortunately results in a greater degree of complexity for the new player. Again, the time needed to learn the meaning of eighteen attributes is three times more time consuming than that needed to learn a basis of six. This can easily be dealt with by ensuring that players begin the game with standard score packages that are fixed.[47] Limiting the amount of choices during initial play will reduce the complexity of the system, and because the system presents players with a greater number of choices further along during play, experienced players should not be too discouraged. As an added bonus, the standard score package is extremely easy to balance. The developer should easily be able to create challenges that are ensured to be

[45] (Intellect, Wit, Concentration, Might, Agility, Fatigue, Vitality, Hit Points, Energy, Insight, Will Power and Precision)
[46] Everything is done automatically by the system. For example, the user need not resist a spell, the combat system will take care of it for him.
[47] See the section "Assigning Points" in the analysis.

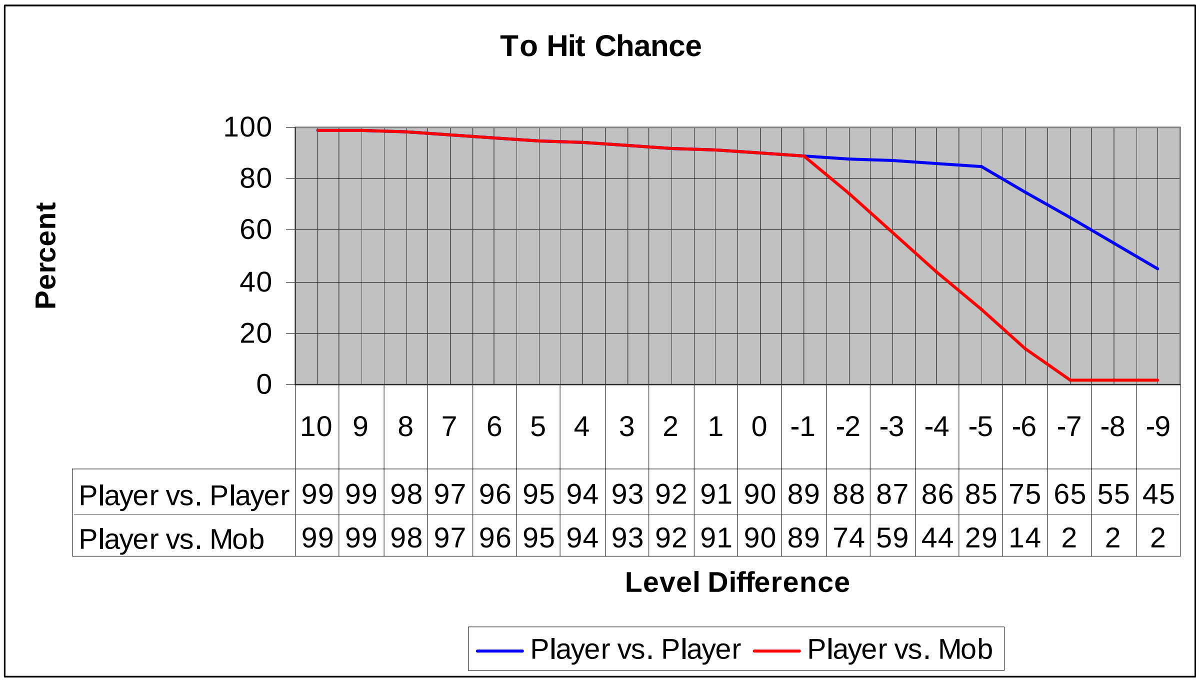
balanced with the values of the attributes during initial play. This is extremely important since it highly affects the first 30 minutes[48] of gameplay.

The non combat attributes are included only to give a few role-playing attributes and to balance the physical, spiritual and mental classes so that each class contains the same number of attributes. Providing the developer with a balanced number of attributes for each class, gives the developer the added choice of presenting players with increment bonuses to attribute classes, instead of just bonuses to the attributes. For example an increment of +2 points to the mental class would equal a bonus of +2 points to each attribute within the mental class (an overall bonus of +12 points).

Because the non-combat attributes are not balanced with the combat engine, the developer is encouraged to make sure that these attributes either influence combat in some fashion or another (the appearance of your character might influence which allies are available in combat for example), or ensure that the game is more than just combat oriented (where these attributes will come into play).

The placement of points in scores is completely up to the developer. Because of the small degree of influence, a scheme might be to set the average human potential at around 50 points per attribute. Small variations can occur to differentiate between races during initial game-play. A stratum for the number of points awarded to players upon the advancement of the characters must then be analyzed. See the section on "Advancement" later in the discussion for more on this.

Hitting and Missing



	10	9	8	7	6	5	4	3	2	1	0	-1	-2	-3	-4	-5	-6	-7	-8	-9
Player vs. Player	99	99	98	97	96	95	94	93	92	91	90	89	88	87	86	85	75	65	55	45
Player vs. Mob	99	99	98	97	96	95	94	93	92	91	90	89	74	59	44	29	14	2	2	2

Figure 18: The figure illustrates how the basic hit chance is influenced by difference in level, and is determined via the target's type.

[48] The "three thirties" are the first 30 minute, 30 hours and 30 days of a gaming session for new players. See the appendix section for more.

The figure above visualizes how the “to hit” system works. To encourage player vs. player combat, the penalty for attacking a higher level opponent has been lowered giving the lower level player a significant chance to win as can be seen by following the blue line. A character attacking a player 5 levels higher than himself and then a player 6 levels higher will feel a drop from an 85% chance to hit, to a 75% chance to hit. The three formulas used are shown below:

Equal or lower level:	$CtH = (90\%+(AWP - DDP)\cdot 0.1\%)+Misc$
Higher level mobs:	$CtH = 89\%+((AWP - DDP)+10)\cdot 1.5\%+Misc$
Higher level characters:	$CtH = 85\%+((AWP - DDP)+50)\cdot 1\%+Misc$

The formulas are level dependent to ensure that higher level characters still have a bonus against characters of lower level even if their weapon and defense proficiencies have not reached the maximum attainable level. This is especially useful for spell wielding characters or non melee based characters whose weapon and defense proficiencies will often be lower.

The base chance to hit is set to 90% when attacking a character of equal level. This chance may seem extremely high but in comparison to other games, it really is not (“World of Warcraft” has a base chance of 96% as an example). When taking into account that characters and mobs may dodge and parry, the chance to hit is severely lowered, since these hits will result in no damage being inflicted (essentially the same as a miss). Blocking affects the issue as well to a certain degree. The unmodified chance to hit an opponent of equal level that can dodge, parry and block is 82%, with 4% of the hits reducing damage inflicted via the shield.

To ensure that a side can win an encounter a limit to the maximum dodge, parry and block chances have been set. Because parry, dodge and block override critical hit and hit chances, players will want to acquire items, or points in attributes that increment these values. But, because the three chances have a limit in terms of their affect on the formula, the critical hit chance will also be of interest. Additionally, the theory on role-playing games in terms of the advancement of a character would suggest that players will want to enhance their critical hit chance as much as possible so as to advance faster in the game.

Role-playing games reward characters with faster advancement for harder challenges (this is balanced with the need for more experience). Therefore, to accelerate the advancement of a character, the player will often attack mobs of higher levels (a harder challenge, and therefore yielding more experience). Remember that an advancement of a character also means the acquisition of items, and that higher level mobs usually carry higher level equipment. The result of combat with such a mob is a lower chance to hit, in which case a base chance of 90% is more than justified.

Weapon and Defense Proficiencies

An attack and defense proficiency exists for all classes of weapons that the developer chooses to create. The reason for this is because defending with one type of weapon is often completely different than defending with another and the same rule applies for attacking with the weapons.

The attack and defense proficiency system is intuitive for players to understand. “Your level of attack with your weapon is set against my level of defense with my weapon.” The system

ensures that both attacker and defender influence the outcome of the hitting system which is a much more realistic account of combat than some of the systems presented in the analysis.
The formula dealing with the rate of gain in proficiency level ensures that characters differ in terms of the amount of time they use in battle, or the amount of points that they have spent in the *wit* attribute. The level of proficiency is broken into 10 points per level giving characters 10 stages of efficiency in terms of the hitting algorithm per level. The d20 and GURPS systems and do not contain any levels of efficiency, but merely grant characters an advancement in proficiency. "World of Warcraft" contains a proficiency system similar to the one described here only courser.

Wit influences the algorithm so that smarter characters learn quickly. It gives the player an added choice to place attribute points in, and because hitting the opponent is important during combat, the decision is a significant one. Furthermore, if the game introduces 10 sets of weapons, the character will have to keep 20 proficiency scores up to date with his level making the wit attribute an important attribute.

Overall Attack and Overall Defense

Overall defense and overall attack are inspired by the "World of Warcraft" system. Although the inner working of their system is unknown, certain information has been provided. A bonus or penalty of 0.04 is applied to characters through weapon proficiency difference instead in much the same fashion as presented in this system. But because the proficiencies already affect the chance to hit in this system, the result of a combined affect would be an extremely high level of influence on combat for proficiency values.

Setting the value to 0.015 creates a maximum of 0.3% percent difference per level between characters (0.015 * 10 (maximum point difference that can be achieved per level) * 2 (the affect of the difference goes both ways, e.g. I raise my critical hit and reduce yours)). The result is a 3% difference between characters of 10 levels multiplied by the number of skills it affects (critical hit, dodge, parry, and block). The overall advantage one could maximally achieve is 12% per ten levels of difference. This advantage may seem small (some games only consist of 20 levels) but considering the fact that a higher level character already has a better chance to hit and most probably a larger amount of health and resistances, the value seems fair.

Weapon Speed and Damage

Having two factors to define weapons makes it easy to balance weapons and at the same time have a large variety of weapons. The weapon speed and damage are the classical ways for MMOs to define their weapons.

Damage must be analyzed by the designer of the system in terms of the damage type vs. reduction, the chance to hit, and the number of hit points of the target. For example, let us create a combat scenario between two equal characters without any chances to critical hit. The characters are of equal level and have the following values:

- Dodge = 4%
- Parry = 4%
- Block = 0%

- Equal level of weapon proficiency resulting in 90% base chance to hit.
- Equal level of overall attack and overall defense
- Attack type slashing
- 30% resistance to slashing damage

If we grant the characters 20 hit points and 20 vitality, they will have 220 health points that will need to be reduced (provided that none of the characters can reduce vitality before hit points) for one combatant to be the victor. If we wish for combat to last around 10 seconds, the damage of the characters should equal:

$$damage = \frac{22}{0.70 * 0.82} = 38.32$$

We want an incoming 22 (220 health / 10 seconds) points inflicted per second but have to take into account the miss chance and the resistance values.

To set the damage correctly, the damage must be analyzed in terms of the character's attributes, such as might, and the characters equipment, such as a sword. With this information, our character could have a might value of 70 (7 damage per second) and therefore need to be equipped with a two handed sword doing 72.3-84.3 damage and a swing speed of 2.5 seconds (31.32 damage per second).

Damage Reduction

Attacks considered magical have been dealt into the classical elements from Greek history namely fire, water, air and earth. Light and dark attack types have been included as well to give a larger variety of resistances and because they also are frequently present in fantasy literature.

The mundane resistances include the three basic attack types for weapons; slashing, piercing and bludgeoning. Explosions are a common attack type in many games, and although an explosion can be categorized under a mixture of the basic resistances, the addition of the resistance type seemed appropriate. A player will usually not doubt whether an explosion just occurred, but may be in doubt of whether the explosion consisted of bludgeoning damage (e.g. if at a considerable distance from an explosion, one might get hit by sharp shrapnel but not from the impact).

Finally, for attack types that fit none of the resistance categories, the three overall categories, resolve, hardiness, and reflex exist. These resistances correspond to the d20 resistances, albeit with a much lesser impact because of the nine other resistance categories.

To calculate the desired resistance rating for characters the following formula can be used:

$$RR = \frac{DR \cdot lvl \cdot 100}{100 - DR}$$

Where:
RR = Resistance Rating
Lvl = Level
DR = Damage Reduction in %

To illustrate how damage reduction in the combat system works, take a look at figure 18.

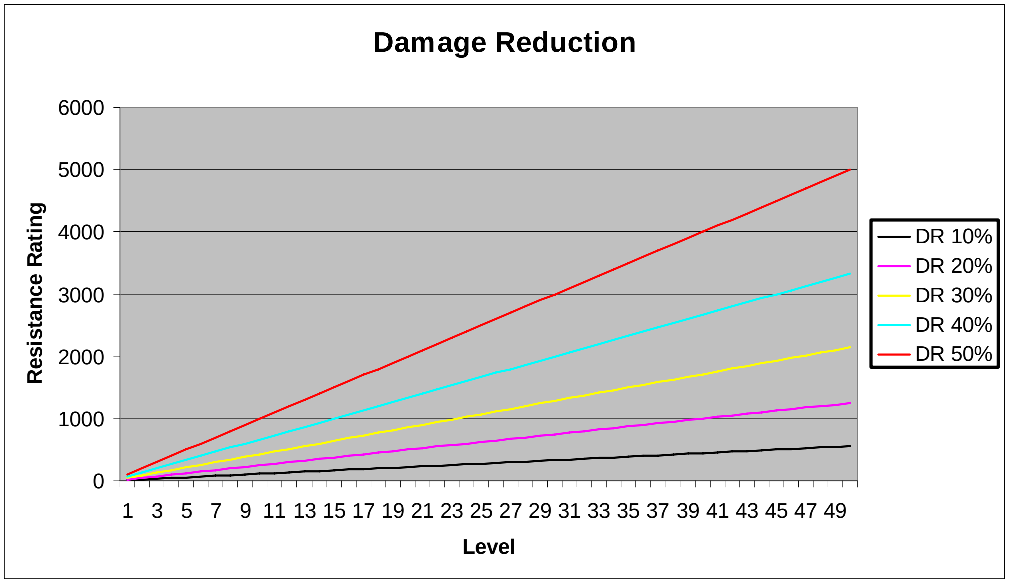


Figure 19: An illustration of the damage reduction formula in use.

Developers wishing for a damage reduction of around 30% for their ranged characters, for example, need to ensure that the total armor value of a character stays close to the yellow line each level. The combat system sets no limits to the maximum damage reduction that can be achieved, although there is a limit for the minimum damage that can be done which is 1. It is up to the developer to ensure that the resistance rating stays within the range that the developer wishes for each level. The reason for this design choice is to allow the developer to give near total immunity to characters or mobs against certain attack types.

Aggro

The aggro system contains a number of simple rules that should ensure a level of strategic play from players. Ranged characters are allowed to inflict more threat upon the mob than melee characters, and this design is important to ensure that the enemy is not caught in between characters unable to decide who to attack. This is called "juggling" a mob, and it involves two or more ranged characters attacking from two different directions. If the threat is somewhat equal, then the mob will change direction with each hit from the ranged weapon, and with a little luck never reach its destination to fight back.

With the current design, should a ranged character gain aggro, a melee character will have to inflict a minimum of almost 50% extra threat in terms of his current threat score to gain back the aggro. This can be seen via the following example.

1. A melee character currently has 100 threat on the mob
2. The ranged character will have to inflict 135 threat to gain aggro
3. The melee character must surpass the 135 threat with 10% to gain aggro back equaling 148.5 threat which is 48,5% more aggro than he currently has

Taking into account that melee characters are traditionally the best armored characters (why else would they get into melee range), and ranged characters are traditionally the worst armored, managing aggro is of vital importance.

Two ranged characters trying to juggle the mob will have to increase their threat with at least 82,25% of their last threat value, and as the threat grows on their target, this becomes more and more difficult to do. As an example, take two ranged characters attacking a mob. Both ranged characters agree beforehand that once one gains aggro, he stops attacking until his partner gains aggro. They are capable of doing 90-110 threat with each shot of their gun.

1. Both characters attack and character one does 100 threat damage while character two does 90
2. Character two needs 182.25 threat damage to take aggro, luckily he is capable of doing so since he can produce 90-110 threat. His attack does 100 and he now gains aggro with 190 threat
3. Character one now needs 346.275 threat to gain aggro which unfortunately will take 3 shots. Let us assume he does so and hits the minimum of 370 threat (which is fortunate for character 2 who will not have to work harder to gain aggro)
4. Character two now needs a staggering 674.325 threat to keep "juggling" the mob. This will take him a minimum of 5 shots.

The aggro system is a form of artificial intelligence for mobs. The system is very simple compared to many other aggro systems ("World of Warcraft" contains algorithms for when the mob should flee as an example) but still results in a good level of strategic play. Because artificial intelligence is beyond the scope of this project, a more efficient aggro system has not been created. Desirables could include a fleeing system and an ambush system depending on the number of players the mob will engage.

Advancement

The advancement of a character will influence the combat system substantially. The attribute influences have been kept small for exactly this reason. For example, a character with 100 points in all attributes will have the following bonuses from his attributes.

- +10 damage per second
- +2,5% learning speed
- 5% extra resistances
- 5% decreased interruption chance
- +5% damage absorbed when blocking
- +3,5% extra block and dodge chance
- +5% chance of a bonus to find valuable items
- +2% chance to critical hit

The damage per second can easily be balanced with resistance ratings and the damage of items. The same applies to the bonuses given to resistance ratings. Critical hit chances are also easy to balance with the multipliers found on the weapons. On the other hand the block, dodge, learning speed and decreased interruption chances are difficult to balance. The attributes affecting them must be analyzed thoroughly before designing an advancement policy. For example, say that the learning speed must never surpass 25%, then the amount of advancements in the attribute must never exceed

1000 – bonuses to the attribute through items. If for example, the game includes 100 levels, then upon gaining a level, the character might gain 1-5 points in all attributes. A maximum of 500 points can be achieved through level advancement, and the rest of the points must be obtained though items found in the game.

The Combat Analyzer

Because the combat engine is fairly complex, it is difficult for the combat engine itself to balance two characters in a way that will not disrupt the design of a character. For example, say that we pin a warrior and a magician against one another, and the warrior has no problem winning the outcome. A possible fix for the magician would be to enhance the number of hit points, and thereby hopefully increase the duration of combat in the magician's favor. Unfortunately magicians are usually designed to have a small number of hit points and this result is useless.

The designer usually already has an idea of which attributes he might want to change in order to balance a given character. In order to give the developer relevant feedback concerning such a change, the analyzer provides functionality to increment one value during tests.

Test Scenario

The default values of the implemented program are the result of a test scenario created to show how the combat system works.

Setting

The most popular genre in role-playing games is that of the fantasy genre so the test will include two characters engaging one another from this setting.

The Warrior

The warrior is a character with a high amount of resistance but a low amount of damage. He kills his enemies with slow strikes because of the weight of his armor, and bares a shield for extra protection. Should his enemies pierce his armor, the warrior remains confident thanks to his large amount of health.

Mental	Physical	Spiritual
Intellect 20	Might 35	Energy 20
Wit 26	Agility 20	Luck 27
Knowledge 26	Fatigue 30	Insight 30
Reason 27	Vitality 30	Willpower 30
Concentration 22	Hit Points 26	Precision 20
Sanity 20	Appearance 26	Character 22

The warrior is a level one character, possessing a high amount of resistance due to his armor against most forms of mundane damage. Unfortunately, warriors aren't the type of class that gets out of the way of things, which results in poor reflexes. Also, warriors are considered weak of mind, and this can be seen via his resolve resistance rating. Magic is foreign to the warrior and he has not been able to find any gear that will help him resist the effects of it resulting in the following resistance ratings.

Mundane	Magic
Slashing 100	Fire 10
Piercing 89	Water 10
Bludgeoning 105	Air 10
Explosive 67	Earth 10
Resolve 30	Light 10
Hardiness 98	Dark 10
Reflex 20	

The warrior is a capable adversary in terms of his defense. Thanks to his shield he is capable of absorbing blocking an attack and absorbing damage. The unmodified values of the warrior are given below.

- Dodge 4%
- Parry 4%
- Block 4% (Absorbs 2 damage)
- Critical hit 4%
- Weapon attack proficiency skill 10 with category "one handed swords"

- Weapon defense proficiency skill 9 with category “one handed swords”
- Overall Attack 9
- Overall Defense 10

Finally, the warrior is equipped with a rusty long sword given to him from his uncle. The sword is capable of inflicting *30-40 damage* and has an attack speed of 1.5 seconds and a critical hit multiplier of 2.5 (damage is multiplied by 2.5). The damage type of the sword is *slashing*.

The Rogue

The rogue is the exact opposite of the warrior, capable of doing large amounts of damage by attacking precisely and quickly. He wears a small amount of armor enabling him to conduct his agile strikes and is a master of dodging and parrying attacks.

Mental	Physical	Spiritual
Intellect 20	Might 20	Energy 15
Wit 30	Agility 40	Luck 21
Knowledge 26	Fatigue 20	Insight 30
Reason 27	Vitality 23	Willpower 26
Concentration 22	Hit Points 18	Precision 35
Sanity 20	Appearance 24	Character 28

The rogue is somewhat has an average resistance rating to most forms of damage. He wears leather armor, ensuring that he can continue to fight effectively while still protecting him in the most vital areas. The rogue fears and despises magic in all of its forms and has been looking for gear specifically granting him greater resistances towards magic.

Mundane	Magic
Slashing 30	Fire 30
Piercing 40	Water 40
Bludgeoning 25	Air 20
Explosive 33	Earth 45
Resolve 30	Light 30
Hardiness 35	Dark 46
Reflex 75	

The rogue is well prepared for combat but enjoys taking out his foes quickly and precisely. Should an adversary prove difficult, the rogue has learned an effective way of dodging almost any attack as seen in the chance to dodge. Because the rogue is used to quick encounters, he has not learned all there is to know about fighting. Therefore his defensive and offensive proficiencies are set to be quite low. The unmodified values of the rogue are shown below.

- Dodge 13%
- Parry 4%
- Block 0%
- Critical hit 25%
- Weapon attack proficiency skill 5 with category “daggers and knives”
- Weapon defense proficiency skill 4 with category “daggers and knives”

- Overall Attack 8
- Overall Defense 7

The rogue is armed with the preferred weapon of almost any rogue; the dagger. The dagger is capable of inflicting 28-32 damage with a critical modifier of 2 (damage is multiplied by 2) and attacks every second. The damage type is piercing.

Combat

An analysis of the encounter is conducted. The resistance formula used acts as a damage reduction for the purpose of this combat, and the level of both characters is 1. The modified combat values of both participants have been calculated and are shown below.

	Warrior	**Rogue**
Health (HP + Vitality)	290	203
Dodge	4+0.7(agility) +0.03(OA vs. OD) = 4.73	13+1.4%(dodge) -0.03(OA vs. OD) =14.37
Parry	4+0.03(OA vs. OD)=4.03	4-0.03 (OA vs. OD)=3.97
Block	4+0.7(agility)+0.03(OA vs. OD)=4.73	0
Block Absorption	2+1.75(might)=3.75	0
Critical Hit	4+0.4(precision) +0.03(OA vs. OD)=4.43	25+0.7(precision)-0.03(OA vs. OD)=25.67
Weapon Attack Proficiency	10	5
Weapon Defense Proficiency	9	4
Overall Attack (OA)	9	8
Overall Defense (OD)	10	7
Relevant % Resistance Rating	47.089%	23.076%
Damage Per Second	23.333 +3.5 (might) =26.	30+2 (might) = 32
Chance to Hit vs. opponent	96%	86%

To determine the average outcome the following formulas are used (Note that the formula does not take into account the rules regarding dodge, parry, block and critical hit. Fortunately none of the combatants have any values where the rules must be applied):

$$dps' = dps \cdot CtH$$
$$dps'' = dps' \cdot (100\% - (P(d) + P(p) + P(c) + P(b))$$
$$+ (dps' \cdot P(c) \cdot M)$$
$$+ dps' - absorb \cdot P(b)$$
$$dps''' = dps'' \cdot (1 - RR)$$

Where:
dps = your damage per second, P(d) = opponent dodge chance,
P(p) = opponent parry chance, P(b) =opponent block chance,
P(c) = your critical hit chance, M = critical multiplier
RR = Resistance Rating

First we multiply the damage by the chance to hit, and then find the value of damage that results in normal damage. This result is added to the damage that results in critical damage which again is added to blocked damage. The final result is then reduced due to resistance rating.

The equation results in the rogue inflicting 12.814 damage per second against the warrior, and the warrior 16.944 against the rogue. The warrior should win the encounter if the core mechanic rolls fairly.

Conclusion

A generic combat system for the massively multiplayer online role-playing genre has been designed and implemented. The design of the system is based on successful generic role-playing games such as the d20 system and the GURPS concept. The combat system consists of

- a core mechanic that is intuitive, possessing a fine degree of precision
- a hitting and missing algorithm with several different outcomes
- attributes that are tied to the enhancement of combat
- a resistance system
- an aggro system
- a weapon proficiency system

In comparison to other combat systems, the combat system is not innovative, consisting only of the classical elements of RPGs. On the other hand, certain aspects of the design are unique. The degree of precision and the amount of attributes are, to my knowledge, never before seen and therefore the combat system will be able to outperform many other systems in terms of its balance. The number of choices that the combat system presents is also superior to many other systems. These choices, from a player's perspective, will range from significant to critical[49] in terms of the choices the player will face during the course of a game.

The balancing of characters is of prime importance in MMORPGs. Nobody wants to play a character that they feel is inferior compared to other characters. With a fine level of balance, the developer can adjust certain aspects of the character without affecting other aspects that might result in a state change from inferior to superior.

In terms of the learning curve for players, the combat system is quite difficult. But if one follows the stratums mentioned in "The Three Thirties of MMP Game Design" found in the appendix section, then the combat system can gradually be introduced to the player.

The implementation of the combat system resulted in a combat class closely tied to a character class. A user interface has been provided in order to set values and test for balance between created characters. The default values of the combat system are the values found in the test scenario. An implementation of the combat analyzer and graphical representation of combat was not accomplished.

[49] See the appendix section "Decisions"

References

[Wizards, 1995] 1995-2005 Wizards of the Coast, Inc., a subsidiary of Hasbro, Inc. (Online) Available from 2006 Feb 7 http://www.wizards.com/default.asp?x=d20/welcome (March 3, 2005)

[Alexander, 2003] Alexander, Thor (2003). Massively Multiplayer Game Development, Charles River Media

[Alexander, 2005] Alexander, Thor (2005) Massively Multiplayer Game Development 2, Charles River Media

[Hallford, 2001] Hallford, Neal (2001). Swords and Circuitry, a designer's guide to computer role-playing games, Prima Tech

[Petruccelli, 1999] Petruccelli, Joseph D (1999). Applied Statistics for Engineers and Scientists

[Woodcock, 2005] Woodcock, Bruce Sterling. "An Analysis of MMOG Subscription Growth" MMOGCHART.COM 12.0. 29 November 2004. 1 January 2005. (Online) Available from 2006 Feb 7 http://www.mmogchart.com

[Dachary, 2004] Dachary, Loic "Underware will be a generic set of libraries and tools designed to produce 3D online games" 14 November 2004 (Online) Available from 2006 Feb 7 http://www.mekensleep.org/

[Wikcon, 2006] Wikipedia contributors. "Role-playing game" Wikipedia, The Free Encyclopedia; 2006 Feb 7, (Online) Available from: http://en.wikipedia.org/w/index.php?title=Role-playing_game&oldid=38608658.

[Carpenter, 2003] Carpenter, Adam (2003) Applying Risk Analysis To Play-Balance RPGs, Gamasutra 11 June, 2003 (Online) Avaliable from 2006 Feb 7 http://www.gamasutra.com/features/20030611/carpenter_03.shtml

[Lexico,1995] Dictionary.com is a multi-source dictionary search service produced by Lexico Publishing Group, LLC, a leading provider of language reference products and services on the Internet. 2006 Feb 2 (Online) http://dictionary.reference.com/help/about.html

[Riley, 2004] Riley, Sean (2004) Game Programming with Python, Charles River Media

[Woo, 1999] Woo, Mason; Neider, Jackie; Davis, Tom; Shriener, Dave (1999) OpenGL Programming Guide Third Edditio, Addison Wesley

[Martelli, 2005] Martelli, Alex; Martelli, Anna; Ascher, David (2005) Python Cookbook, O'Reilly

[Lutz, 2003] Lutz, Mark; Ascher, David (2003) Learning Python, O'reilly

[Fullerton, 2004](Online) Fullerton, Tracy; Swain, Christopher; Hoffman, Steven (2004) "Improving Player Choices". Gamasutra 2004. available from 2007 Jan 20 at http://www.gamasutra.com/features/20040310/fullerton_01.shtml

Appendix Section

Appendix

Appendix *1*

The Combat Program **2**

Attributes of Interest 2

Methods of Interest 2

Interface 3

Defining a Player **5**

Why we play 6

Emergence 7

Decisions **8**

Other Systems Contributing to the Analysis **8**

Roll-playing vs. role-playing **9**

The Three Thirties of MMP Game Design **10**

The First Thirty Minutes 10

The First Thirty Hours 10

The First Thirty Days 11

ODDPAW: Open-source Design and Development of Persistent gAme Worlds **12**

I. Introduction 12

II. Project Goals 13

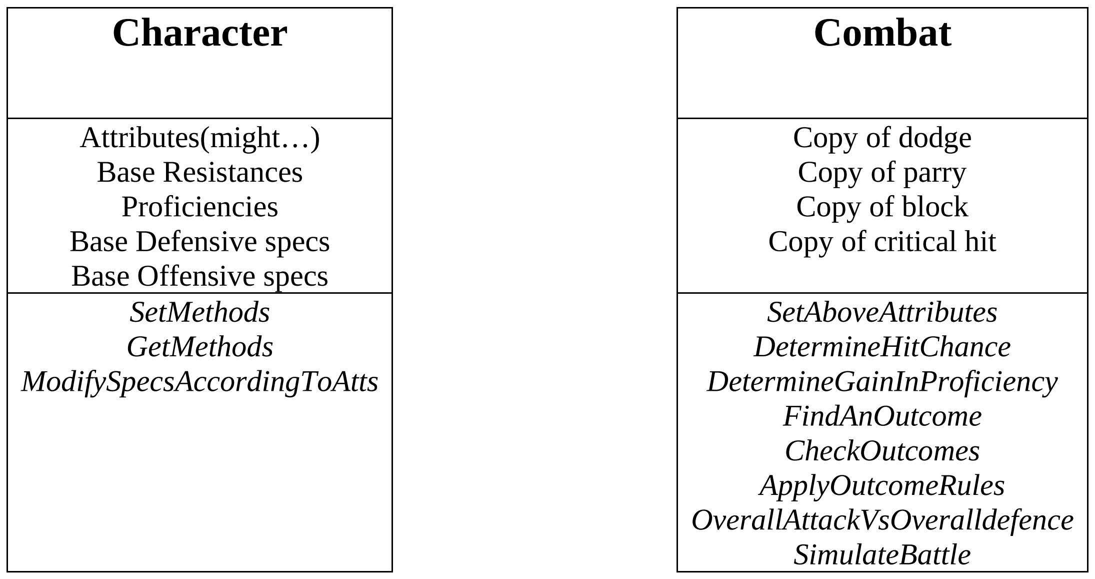
UNDERWARE **13**

Program Code **14**

The Combat Program

To run the code you will need python 2.4, pygame, pyui, PIL, PyOpenGL and opengl installed on your system.

A combat class and a character class work together to form the combat engine. The character class contains all information concerning the character needed by the combat engine, while the combat class ensures that two characters can engage one another in combat.



The two classes above are the two central classes comprising the combat engine. An Interface class and an Application class exist as well. The Interface Class is used to manipulate the values of the Character Class and start a combat via the Combat class. The application class is used to start the program and set up a window for the Interface class.

Apart from the "*ModifySpecsAccordingToAtts*" method in the Character class, the methods in the combat class are of the main interest. A description of each method follow, but first a small note about the Combat class's attributes.

Attributes of Interest

A copy of the dodge, parry, block of the opponent and critical hit chances of the attacker is made because the values are modified before the determination of a hit chance. This ensures that the base values of a character remain the same. The modifications can be found in the description of the methods below.

Methods of Interest

ModifySpecsAccordingToAtts

The method contains all information regarding the influential values of all attributes. Once attributes have been set, base values are updated to reflect the influence from attributes. To ensure that all attributes have been set, the call to this method is made directly before combat.

DetermineHitChance
The method determines the chance to hit by comparing the level and type of the two combatants, and then choosing the correct formula for hit chance determination. If the hit chance exceeds the minimum and maximum chances to hit, then the hit chance is corrected to reflect the automatic success and failure of events on a lucky roll.

DetermineGainInProficiency
Every attack, no matter the outcome, has a chance to raise the proficiency and overall attack and defense levels of the attacker. The chance of gaining an increment is determined. If the character has not reached his learning capacity in respect to his values, a random roll for each of the four potential increments is conducted and compared to the chance. If the result is less than the chance, an increment of 1 is set.

ApplyOutcomeRules
Before an outcome can be determined, this method is called to ensure that all values lie within the rules for outcome. If dodge+parry+block exceed the 50% limit, all are reduced with the same value so that the sum = the limit. In the case that only one or two outcomes comprise in the sum of over 50% (if one of the outcomes is 0% for example), these outcomes are the only ones reduced.

CheckOutcomes
The method ensures that none of the outcomes have been reduced to under 0. This can be the case because outcome rules and Overall Attack and Defense affect the outcome percentages. If an outcome is found to be negative, the outcome is set to 0

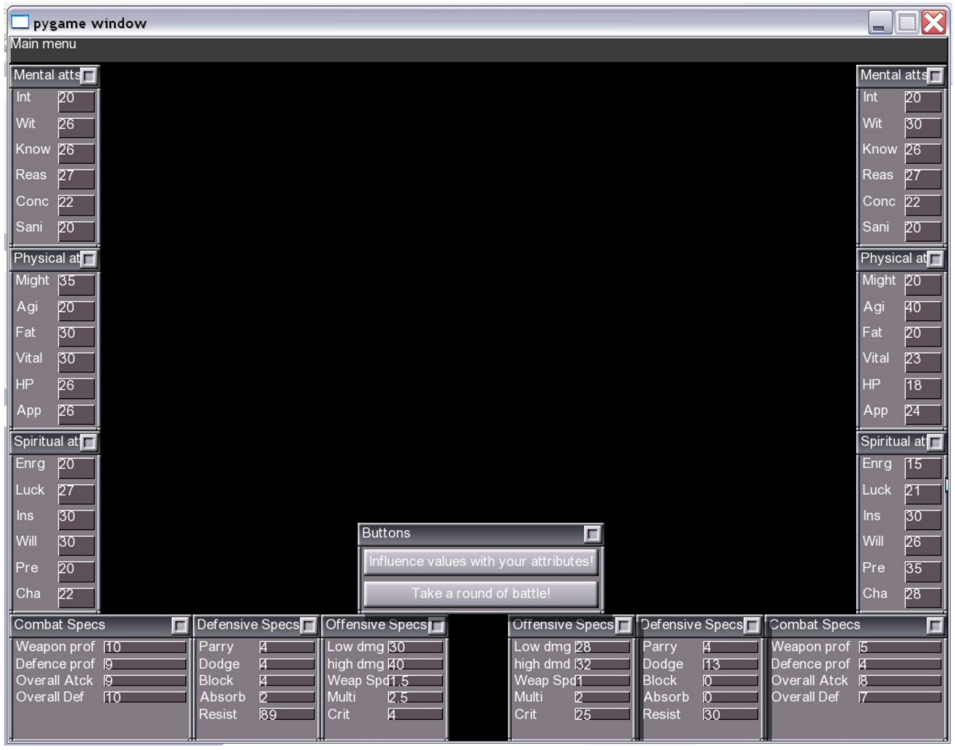
FindAnOutcome
To find the outcome of a hit, a random roll is conducted and first compared to one of the outcomes. If the roll is not within the outcome percentage it is compared to the value of the same outcome plus the value of the next outcome. Once all outcomes have been checked and the roll is still undetermined, the roll falls in the basic hit outcome category. If any of the outcomes result in damage, damage is determined with respect to the opponents resistance rating and damage absorption (in the case of a block), and finally deducted from the opponent’s health.

OverallAttackVsOveralldefence
The rules that apply if the attacker has a lower or higher value than the defender in overall attack and overall defense are applied via this method.

SimulateBattle
SimulateBattle simple ensures that all relevant methods for a battle are called in the correct sequence. SimulateBattle is the only connection the class has with the Interface class, via a button called Battle!

Interface

The resulting interface contains edit boxes for all character values as can be seen below. The battle button creates one round of combat between characters that lasts one second. Characters with attack speeds that differ from one second will have their damage normalized to 1 second. The outcomes of each attack can be seen in the console. Keep an eye out for the victor.’

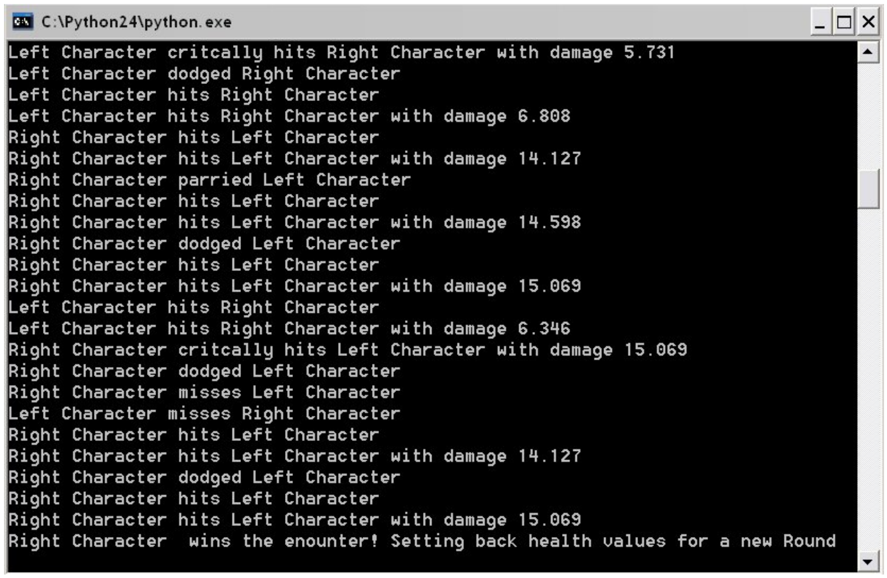


Mental atts		Mental atts	
Int	20	Int	20
Wit	26	Wit	30
Know	26	Know	26
Reas	27	Reas	27
Conc	22	Conc	22
Sani	20	Sani	20

Physical at		Physical at	
Might	35	Might	20
Agi	20	Agi	40
Fat	30	Fat	20
Vital	30	Vital	23
HP	26	HP	18
App	26	App	24

Spiritual at		Spiritual at	
Enrg	20	Enrg	15
Luck	27	Luck	21
Ins	30	Ins	30
Will	30	Will	26
Pre	20	Pre	35
Cha	22	Cha	28

Combat Specs		Defensive Specs		Offensive Specs		Offensive Specs		Defensive Specs		Combat Specs	
Weapon prof	10	Parry	4	Low dmg	30	Low dmg	28	Parry	4	Weapon prof	5
Defence prof	9	Dodge	4	high dmg	40	high dmd	32	Dodge	13	Defence prof	4
Overall Atck	9	Block	4	Weap Spd	1.5	Weap Spd	1	Block	0	Overall Atck	8
Overall Def	10	Absorb	2	Multi	2.5	Multi	2	Absorb	0	Overall Def	7
		Resist	89	Crit	4	Crit	25	Resist	30		


C:\Python24\python.exe

```
Left Character critcally hits Right Character with damage 5.731
Left Character dodged Right Character
Left Character hits Right Character
Left Character hits Right Character with damage 6.808
Right Character hits Left Character
Right Character hits Left Character with damage 14.127
Right Character parried Left Character
Right Character hits Left Character
Right Character hits Left Character with damage 14.598
Right Character dodged Left Character
Right Character hits Left Character
Right Character hits Left Character with damage 15.069
Left Character hits Right Character
Left Character hits Right Character with damage 6.346
Right Character critcally hits Left Character with damage 15.069
Right Character dodged Left Character
Right Character misses Left Character
Left Character misses Right Character
Right Character hits Left Character
Right Character hits Left Character with damage 14.127
Right Character dodged Left Character
Right Character hits Left Character
Right Character hits Left Character with damage 15.069
Right Character  wins the enounter! Setting back health values for a new Round
```

Defining a Player

The information presented here was found relevant to this project and is a summary from [Alexander, 2005] A general rule set and combat system should accommodate all player types. A player of a game is defined as a participant therein. People who deal with the administrative aspects of playing are not considered players in terms of this paper.

Playing styles are divided into two dimensions and the inter-relationship of these dimensions identifies the player types: action vs. inaction, world oriented vs. player-oriented. The four types are as follows:

Achievers: Enjoy having game-related goals that they wish to achieve. They play to “win”.
Explorers: Interact and experiment with the world. Delight in discovery.
Socializers: Interact with other players, enjoy role-playing through the communicative facilities provided via the game interface.
Killers: Enjoy engaging with other players usually to dominate them through politics, bullying, killing, and so forth.

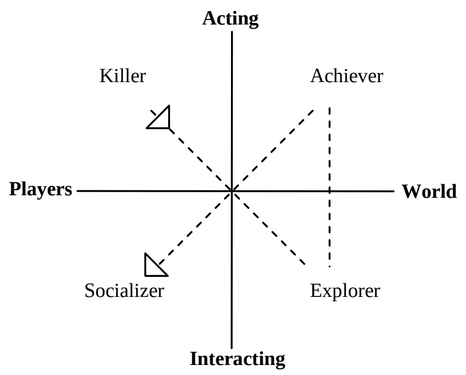


Figure 1: A graph depicting the dimensional placement of player types as well as the development sequence.

The development sequence also known as the “main sequence” is an observed tendency that many players step through when they begin to play. Players start by killing one another, and once the fighting gets tiring they start to explore the world. Having attained sufficient knowledge they begin to want to win the game and finally, once the game has been won, they settle down and socialize.

Subtypes have since been added to the above model adding a third dimension, a “z” axis containing “explicit” and “implicit” categories on its positive and negative axis. Implicit describes what is done automatically without the intervention of the conscious mind, while explicit action is “planned for action” to achieve a desired goal or effect.

	Explicit	Implicit
Achiever	Planner	Opportunist
Explorer	Scientist	Hacker
Socializer	Networker	Friend
Killer	Politician	Griefer

Figure 2: Explicit and Implicit dimensions added dividing player types into eight new categories.

A description of the new subtypes follows:

Opportunist: See a chance, and take it! Look around for things to do.
Planner: Set a goal and achieve it. Perform actions as part of a larger scheme.
Hacker: Discover new phenomena and experiment to reveal meaning.
Scientist: Experiment to form theories and explain phenomena. Work methodically.
Friend: Interact with people they know well, and have a deep understanding of. Accept little foibles.
Networker: Find people to interact with, learn what they know, and assess whom to hangout with.
Griefer: Kill, kill, kill to get a big bad reputation. Very "in your face".
Politician: Act with foresight and forethought, manipulating people subtly.

Again there is a main sequence but there are three other common sequences.

Main sequence:	Griefer -> Scientist -> Planner -> Friend
Minor sequence:	Opportunist -> Networker -> Planner -> Friend
Main socializer sequence:	Griefer -> Networker -> Politician -> Friend
Main explorer Sequence:	Opportunist -> Scientist -> Planner -> Hacker

The sequence of development is also of interest in terms of a rule set. As an example, it is apparent that the first step is either Griefer or Opportunist. If players new to game begin the game by waging war (Griefer), the rules for war should be simple so that the player is not overcome with difficulties in understanding the game in an early stage. The rules might become more complex as one gains levels.

Why we play

It is important to know why people play games so that the rules system may accommodate this (if possible). Studies show that the aforementioned player types are some of the reasons we wish to play. Players that are socializers wish to meet new friends via the game, and killers like to prove to other players that they are above everyone else in terms of who is best, and so on. The game simply gives the players a chance to express their desires through the medium, without there having to be any real life consequences[1].

There are other reasons that people choose to play, and one such reason is the notion of immersion. Immersion is the concept of a player's sense of feeling as if **in** a virtual world. Immersion is also described as one of the many forms of the concept of **presence**.

[Alexander, 2005] describes four levels of immersion:

1. Un-immersed
2. Avatar
3. Character
4. Persona

Being un-immersed requires that the player regard the character merely as an ***object*** and nothing more (like these sentences for example). An avatar level of immersion classifies players that identify with the character as being their ***representative*** in the virtual world. Should the player ***extend his personality*** on to the character then the level of immersion is considered a character

[1] Sometimes the desires lead to real life consequences such as gaining real life friends through playing games.

level of immersion. Finally, if the player considers the character as being ***himself in the virtual world***, then the immersion level is that of persona.

As mentioned before, the game is simply a medium for players to express their desires, but it is also a medium that develops affirmation of identity. Imagine an actor who must play the role of a character in a film. Playing the role will allow the actor to learn from the character and gain better insight into his own situation. But in MMORPG's the actor and the character are considered to meet halfway, since there is no actual role to play. Instead, the player often identifies with the character as an image of himself (see levels of immersion), or rather, the image that he would like to see when viewing a self reflection. Therefore, the player and character both learn and may change accordingly from any feedback that they may receive from other players since any feedback will be reflected on to one's own image of self.

Immersion may be discouraged by the underlying game rules as may player identity (killer, explorer …). A simple example may be socializers not being able to socialize. Also, the rules system should be designed in accordance with the desired game-play of the game, or vice versa, so as not to create imbalance within the game, where the game might not be received as a whole, discouraging immersion.

Emergence

Emergence (not to be confused with immersion) is described as being the process of complex pattern formation from simpler rules, or rather from a computer game theory perspective, emergence is the game's rules structure that when combined, lead to a large number of game variation that cannot be explained[2] from the simple rules structure. An example may be the strategies developed by players in strategy games.

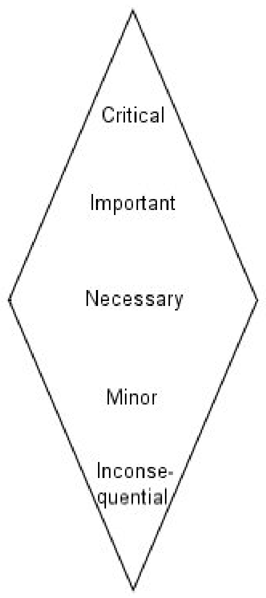
While RPGs may be described as a game composed primarily of progressional gameplay (where the completion of the game is dependent upon the performance of a predefined sequence of actions), they may also be composed of emersion. The concept of emersion is important in explaining that the underlying rule system may not be able to control or foresee the complex pattern formations, which are the result of it. In other words, game-play is not completely steered by the rule system.

[2] This has not been proven.

Decisions

If a game is a series of interesting decisions[3] then it would be of concern to see what decisions a player will come across when dealing with the combat system. Decisions have been classified into the following set {inconsequential, minor, necessary, important, critical} and have been placed into the following model. The thickness of the diamond illustrates the number of decision types that the player should be facing in the lifespan of a game. [Fullerton, 2004]

- Inconsequential decisions have no impact on the outcome of the game and should therefore be avoided as they are not worth making.
- Minor decisions have little impact or direct results on the game but do constitute to interesting decision making.
- Decisions considered necessary, are the decisions that players should be making most of the time. They have a good impact upon the game and the player's character and the impact should easily be seen by the player.
- Important decisions are crucial to the success of the character or have a large impact upon the character.
- Critical decisions are life threatening and although these decisions may be the most interesting to make, they should not be the ones that the player makes most often.



The illustration to the right has been modified to resemble a diamond instead of a triangle as originally presented in [Fullerton, 2004]. This is because the point being stated here is different than that of the article.

Other Systems Contributing to the Analysis

Chaosium

Chaosium is best know for their Call of Cthulhu series of games and novels. The company was founded in 1975 and has produced several horror related games and novels. Chaosium is also one of the first companies to introduce generic rule sets.

Tri Stat dx

The Guardians of Order, Inc. are the owners of the Tri Stat dx generic role playing system. The company mainly affiliates itself with anime.

Lineage

Lineage is one of the most successful MMORPGs to date. It is produced by NCSoft and has an extremely large bearing of players. The mechanics of the system are unknown.

[3] The popular definition of a game by Sid Meier, creator of SimCity.

Guild Wars
Guild wars is an MMORPG with a special approach to the Genre. It is a mission based game where the objective is to form teams for every mission. The game lacks a large persistent world state, but still qualifies as being an MMO.

City of Heroes
City of Heroes is an MMORPG positioned in a super hero setting. It is considered a very innovative MMORPG and has done well within the genre.

Roll-playing vs. role-playing

Some players enjoy the challenge of role-playing their characters. They often do not care about the actual statistics and skills of the character, but rather the challenge of role-playing these skills and statistics. Other players enjoy the challenge of min-maxing their characters for a certain purpose (often combat). These players are often known as the roll-players by the role-players because they will often find themselves rolling the dice more often than role-playing their characters[4]. For example, roll-players will tweak their characters for the sole purpose of becoming an ultimate killing machine caring not if their characters immensely lack in other areas such as reading/writing skills. The two player types often despise one another. [Wikcon, 2006]

Surprisingly, the blame for how each player type chooses to play does not always fall on the player. Instead, the blame lies with the governing rule set of the game in question. Role-players accuse rule sets of not providing more role-playing rules such as more emphasis on look and personality. They argue that there are too many rules, and that the rules are too precise instead of guide lining leading to better role-playing instead of rolling dice for almost everything. Also the term “role-playing game” is often used for games that feature character building but in which players cannot make any meaningful decisions or act “in character”. Players are not actually role-playing anything in the “role-playing game”, but are instead making tactical decisions. [Wikcon, 2006]

[4] Some roll-players will argue that they are role playing their characters quite well by min-maxing their characters. That is the exact role that they have given the character.

The Three Thirties of MMP Game Design

The information presented here is a summary "The Three Thirties of MMP Game Design" from [Alexander, 2005]. One of the most important challenges a developer must face is keeping his players interested in the game to keep their accounts active. A development stratum for designers addresses this issue and is called the three thirties of MMP (same as MMO) design.

The First Thirty Minutes

The first thirty minutes is the first stratum defined. The stratum is considered the most important since it influences whether the player sticks with the game, or finds the game uninteresting. During the first thirty minutes of game-play, the game will be introducing its controls, system, fiction, and functionality to the player. All of these must be kept simple and customizable and yet still grab the interest of the player. The key to accomplishing this is to study the genre that the game will be based in and use genre characteristic controls and functionality to keep the learning curve of the player low. As the game progresses, the player will begin to feel comfortable with the system, and the developer will be given the chance to unfold more of the complexities of the system. Another important aspect is to keep the paths of achievement short, so that the player can accomplish initial goals quickly. This will give the player a feeling of accomplishment with and advancement of the character.

The first thirty minutes is also the time in which one presents the single most important aspect of the game to the player: The character. The character creation process must be kept simple and provide opportunities for the player to establish a connection with the character. The three major decisions that the player will be facing upon character creation are:

- The character's path of advancement
- The character's appearance
- The character's identification in terms of name, title, etc.

It is important that the player be given choices of advancement at this stage and that the player has all the information he needs to establish these choices. Uninformed choices might discourage players from starting to play or might discourage players later in the game with thoughts of starting over, or simply quitting. Again, the number of choices must be limiting so that the player can start the game as quickly as possible but still have a feeling of having created a character in which he has a connection to.

The First Thirty Hours

The first thirty hours of game-play will involve the creation of tangible goals of advancement and object acquisition. It is the time in which the developer may introduce long term aspects of play, such as long term questing and long term advancement methods. It is also the time in which to give the player a feeling of affecting the world. For example, the player might be in charge of stopping a group of invaders upon a village. If the player is capable of stopping the invasion, then the village will remain safe, otherwise, the village will be diminished.

It is important to continue frequently awarding the player, and keep the pace of achievements somewhat short. This will continue to enhance the enthusiasm of the player and further tie the knot between player and character.

The first thirty hours is also the time in which the developer will want to introduce more of the world through exploration. To accomplish this, the act of exploration must be entertaining, and provide rewards for the player. This can be done by for example hiding non player characters, which are waiting to be found or providing resources for the player in desolate areas.

Finally, the developer will want to introduce the player to other players in the game. This helps a player further define his character by finding a role within a group. Players that group with other players are far more likely to be long-term players compared to those who do not. Also, by creating a large social environment in the game, the departure of a player will have less impact upon other players. For example, a player departing a group of four players will have a large impact on the group, but a player departing a group of thirty players will have a relatively small impact.

The First Thirty Days

Once a player has played through the first thirty hours, it is time to involve the player with the stratum for the first thirty days. The rate of achievement has slowed down and it is time to give the player the opportunity to create individual projects. These projects might include a housing system or an item achievement system where the look of the character will change. An example is the achievement of unique items giving the characters a unique look.

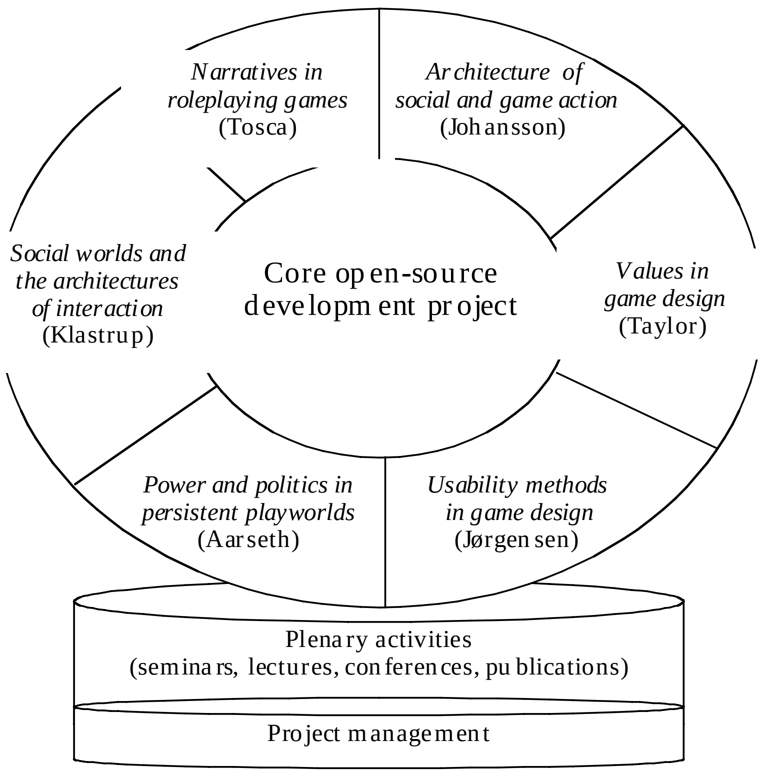
Group projects are another challenge for players at this stage of the game. They might include prestige acquisition for a group or guild, or competitions against other groups. Prestige might track which groups have the largest amount of money, are best at doing a form of activity, or are most effective at killing. Competitions between groups might be in the form of regional challenges or trophy acquisitions. Group projects strengthen all members’ ties within the group and are one of the most important issues in keeping one’s player base.

ODDPAW:
Open-source Design and Development of Persistent gAme Worlds

Proposal coordinator: Espen Aarseth aarseth@itu.dk, Center for Computer Games Research, Dept of Digital Aesthetics and Communication, The IT University of Copenhagen, Game.itu.dk

I. Introduction

The project's aim is to design a framework for persistent-world games where new concepts and new forms of play can be developed and explored, using open-source technology. Organizationally, we are centering the project on the development of the core technologies; surrounding that core with several satellite subprojects, each headed by an experienced games researcher. The satellites are interrelated, cooperate thematically, and depend on the core for technical development and support:


Narratives in roleplaying games (Tosca)
Architecture of social and game action (Johansson)
Social worlds and the architectures of interaction (Klastrup)
Core open-source development project
Values in game design (Taylor)
Power and politics in persistent playworlds (Aarseth)
Usability methods in game design (Jørgensen)
Plenary activities (seminars, lectures, conferences, publications)
Project management

The core development part of the project will identify and adopt a leading open-source game platform, and use this as a foundation on which to build new technologies and to support the satellite projects. The subprojects will each cover a relevant area in persistent-world game design, and inform and support each other in the process, through weekly seminars.

Large-scale, online, persistent world games such as Lineage and EverQuest are tomorrow's innovative social interfaces, but so far a few technological giants have dominated this important evolution. These massive-multiplayer online games (MMOG) draw in huge numbers of players who invest thousands of hours living, playing, and sometimes even working, in these virtual worlds. For some participants, these games have very real social and economic outcomes; outcomes that are partly determined by the immersive worlds, which have designers, programmers, and owners. This brings up the question: Who will control and influence the most important entertainment channels of tomorrow? As MMOGs become a new form of mass media, it is vital for the democratic future of these technologies that they are co-developed, influenced, and understood by research institutions and made available to artists, independent game developers and the general public. Through concept development, experimental social interface design and application of usability methods, the chosen open-source game platform will be further developed, implemented and used in persistent games research and for teaching the principles of game design and development.

The results of this research will be of use to many audiences, including the general community interested in computer games, the game industry, independent game developers and researchers. Through examining both design processes and alternative design possibilities for persistent online worlds, as a market *and* a cultural field which is steadily growing as more and more people choose to spend time playing and socializing online, we will address critical concerns about game design and its broader social, political, and economic implications.

II. Project Goals

- To explore, develop and implement experimental game designs, including alternative models for power distribution and democracy, both in the process of game development and in the games as such
- To help develop an open-access, open-source game platform that can be used for a variety of applications and by a variety of user groups, including educators, non-profit organizations, independent game developers, artists, minority language communities, and the public at large.
- To provide researchers with a low-cost, flexible platform for persistent-world experiments of any kind.
- To create a forum for independent persistent-world game developers and other interested parties.
- To create a body of knowledge about persistent-world games and game technologies, including the understanding of game development processes, useful for lawmakers and relevant public and commercial agencies.

These goals will be achieved through coordinated effort amongst the researchers as they carry out various subprojects.

More information on the ODDPAW system can be found via the Information Technology University of Denmark.

UNDERWARE

Underware will be a generic set of libraries and tools designed to produce 3D online games. It will borrow from existing Free Software tools and bind them together, filling the gaps where functionalities are missing. Simultaneously, a demo game will be developed, whose main purpose is to demonstrate and test every aspect of Underware.

License : GNU General Public License V2 or later
Registration Date : Sunday 02/01/04 at 22:41 CET
Development Status : 4 - Beta
Website: www.mekensleep.org

Program Code

Code

```
#THE COMBAT CLASS

from __future__ import division
import AppCharacter, time, random


class Combat:

    CtH = 90
    OAODval= 0.015
    tdodge = 0
    tparry = 0
    tblock = 0
    mcrit = 0

    #Copy combat values so as not to overwrite
    def setValues(self, mychar, mytarget):
        self.tdodge = mytarget.getDodge()
        self.tparry = mytarget.getParry()
        self.tblock = mytarget.getBlock()
        self.mcrit = mychar.getCrit()

    #formula according to level and type
    def hitChance(self, mychar, mytarget):
        if (mytarget.getType() == 'PC') and ((mytarget.getLvl() -
mychar.getLvl())>5):
            self.CtH = 85+((mychar.CWP-mytarget.CDP)+50)*1

        elif (mytarget.getType() == 'MOB') and ((mytarget.getLvl() -
mychar.getLvl())>1):
            self.CtH = 89+((mychar.CWP-mytarget.CDP)+10)*1.5

        else:
            self.CtH = 90+((mychar.CWP-mytarget.CDP)*0.1)

        #Automatic success and fail
        if self.CtH>99:
            self.CtH=99
        elif self.CtH<2:
            self.CtH=2

    # Start a battle
    def Battle(self, mychar, mytarget):
        self.setValues(mychar, mytarget)
        self.hitChance(mychar, mytarget)
        self.gainProf(mychar)
        random.seed(None)
        if ((random.random() *100) < self.CtH):
            self.OAvsOD(mychar, mytarget)
            self.setOutcomes()
            self.findOutcome(mychar, mytarget)
        else:
            print mychar.Name, "misses", mytarget.Name

    # Determine if I gain in a proficiency or OA/OD upon striking
    def gainProf(self, mychar):
        random.seed(None)
        WPChance = (1 + mychar.getAtts(1) * 0.1 + (mychar.getLvl()* 10 -
mychar.getWP()))
        DPChance = (1 + mychar.getAtts(1) * 0.1 + (mychar.getLvl()* 10 -
mychar.getDP()))
        OAChance = (1 + mychar.getAtts(1) * 0.1 + (mychar.getLvl()* 10 -
mychar.getOA()))
        ODChance = (1 + mychar.getAtts(1) * 0.1 + (mychar.getLvl()* 10 -
mychar.getOD()))
        #If I can gain, then try
        if (mychar.getLvl() * 10) > mychar.getWP():
            if ((random.random() *100) < WPChance):
                mychar.setWP(mychar.getWP()+1)
        if (mychar.getLvl() * 10) > mychar.getDP():
            if ((random.random() *100) < DPChance):
                mychar.setDP(mychar.getDP()+1)
        if (mychar.getLvl() * 10) > mychar.getOA():
            if ((random.random() *100) < OAChance):
                mychar.setOA(mychar.getOA()+1)
        if (mychar.getLvl() * 10) > mychar.getOD():
            if ((random.random() *100) < ODChance):
                mychar.setOD(mychar.getOD()+1)

    # Apply outcome rules to outcomes
    def setOutcomes(self):
        deflect = self.tdodge + self.tparry + self.tblock
        defcontribs = 0

        if (deflect>50):
            if self.tdodge:
                defcontribs += 1
            if self.tparry:
                defcontribs += 1
            if self.tblock:
                defcontribs += 1

            reduct=(deflect-50)/defcontribs

            if self.tdodge:
                self.tdodge -= reduct
            if self.tparry:
                self.tparry -= reduct
            if self.tblock:
                self.tblock -= reduct

        self.checkPositiveOutcomes()
        if (self.mcrit)>50:
            if (self.mcrit) > (100-(self.tdodge+self.tparry+self.tblock)):
                self.mcrit=100-(self.tdodge+self.tparry+self.tblock)

    #Apply OAvsOD rules
    def OAvsOD(self, mychar, mytarget):
        MOA = mychar.getOA()
        MOD = mychar.getOD()
        TOA = mytarget.getOA()
        TOD = mytarget.getOD()
        if (MOA>TOD):
            temp = (MOA-TOD)* self.OAODval
            self.mcrit += temp
            self.tdodge -= temp
            self.tparry -= temp
            self.tblock -= temp
        if (MOA<TOD):
            temp = (TOD-MOA)* self.OAODval
            self.mcrit -= temp
            self.tdodge += temp
            self.tparry += temp
            self.tblock += temp

    #Ensure that all outcomes are zero or more
    def checkPositiveOutcomes(self):
        if self.tdodge < 0:
            self.tdodge=0
        if self.mcrit < 0:
            self.mcrit=0
        if self.tparry < 0:
            self.tparry=0
        if self.tdodge < 0:
            self.tblock=0

    #upon a hit, find the outcome and apply damage accordingly
    def findOutcome(self, mychar, mytarget):
        random.seed(None)
        roll = (random.random() *100)
        if roll < self.tdodge:
            #dodged!
            print mytarget.Name, "dodged", mychar.Name
        elif roll < (self.tdodge + self.tparry):
            #paried
            print mytarget.Name, "parried", mychar.Name
        elif roll < (self.tdodge + self.tparry + self.tblock):
            #blocked
            damage=(random.randrange(mychar.getLowDam(),
mychar.getHighDam(),1))
            damage /= mychar.getAttackSpeed()
            damage += (mychar.getAtts(4) * 0.1)
            damage -= mytarget.getAbsorb()
            damage *= mytarget.getRRpercent(mychar)
            damage = round(damage, 3)
            if damage < 1:
                damage = 1
            print mytarget.Name, "blocked", mychar.Name, "but is still
inflicted with damage", damage
            mytarget.setHealth((mytarget.getHealth()-damage))
            if (mytarget.getHealth() < 0):
                print mychar.Name, "wins the encounter! Setting back health
values for a new Round"
                mytarget.setHealthSelf()
                mychar.setHealthSelf()
        elif roll < (self.tdodge + self.tparry + self.tblock+self.mcrit):
            #crit
            damage=(random.randrange(mychar.getLowDam(),
mychar.getHighDam(),1))
            damage /= mychar.getAttackSpeed()
            damage += (mychar.getAtts(4) * 0.1)
            damage *= mytarget.getRRpercent(mychar)
            damage = round(damage, 3)
            if damage < 1:
                damage = 1
            print mychar.Name, "critcally hits", mytarget.Name, "with damage",
damage
            mytarget.setHealth((mytarget.getHealth()-damage))
            if (mytarget.getHealth()< 0):
                print mychar.Name, " wins the encounter! Setting back health
values for a new Round"
                mytarget.setHealthSelf()
                mychar.setHealthSelf()
        else:
            #hit
            print mychar.Name, "hits", mytarget.Name
            damage=random.randrange(mychar.getLowDam(),
mychar.getHighDam(),1)*mychar.getWMulti()
            damage /= mychar.getAttackSpeed()
            damage += (mychar.getAtts(4) * 0.1)
            damage *= mytarget.getRRpercent(mychar)
            damage = round(damage, 3)
            if damage < 1:
                damage = 1
            print mychar.Name, "hits", mytarget.Name, "with damage", damage
            mytarget.setHealth((mytarget.getHealth()-damage))
            if mytarget.getHealth() < 0:
                print mychar.Name, " wins the enounter! Setting back health
values for a new Round"
                mytarget.setHealthSelf()
                mychar.setHealthSelf()
```

```
#THE CHARACTER CLASS

from __future__ import division

class Character:
    Int = 1
    Wit = 1
    Conc = 1
    Sani = 1
    Might = 1
    Agi = 1
    Fat = 1
    Vital = 1
    HP = 1
    Ener = 1
    Ins = 1
    Will = 1
    Pre = 1
    CWP = 1
    CDP = 1
    Atts = [Int , Wit, Conc, Sani, Might, Agi, Fat, Vital, HP, Ener, Ins, Will,
Pre]
    Health = 0

    Slash =0
    Peirc=0
    Blud=0
    Expl=0
    Resolve=0
    Hardi=0
    Ref=0
    Fir=0
    Wat=0
    Air=0
    Earth=0
    Light=0
    Dark=0
    ResList =
[Slash,Peirc,Blud,Expl,Resolve,Hardi,Ref,Fir,Wat,Air,Earth,Light,Dark]
    Name = ''
    RR=0
    aggro = {}
    lvl=1
    tohitbonus=0
    chartype='PC'
    inRange = 1
    infront = 1
    behind = 1
    inSight = 1
    inCombat=1
    lowdam=0
    highdam=0
    attackSpeed=0
    wMulti=1
    block=0
    absorb=0
    parry=0
    dodge=0
    crit=0

    def __init__(self, initialAtts):
        self.Atts = initialAtts

    def setResList(self, reslistvals):
        self.ResList = reslistvals
#Weapon dependant abilities
    def setLowDam(self, damage):
        self.lowdam=damage

    def setHighDam(self, damage):
        self.highdam=damage
    def getLowDam(self):
        return self.lowdam

    def getHighDam(self):
        return self.highdam

    def setAttackSpeed(self, Aspeed):
        self.attackSpeed=Aspeed

    def getAttackSpeed(self):
        return self.attackSpeed

#definsive and offensive capabilities
    def setBlock(self, value):
        self.block=value

    def getBlock(self):
        return self.block

    def setAbsorb(self, value):
        self.absorb=value

    def getAbsorb(self):
        return self.absorb

    def setParry(self, value):
        self.parry=value

    def getParry(self):
        return self.parry

    def setDodge (self, value):
        self.dodge=value

    def getDodge(self):
        return self.dodge

    def setCrit(self, value):
        self.crit=value

    def getCrit(self):
        return self.crit
#Overall attack and defense
    def setOA(self, value):
        self.OA=value

    def setOD(self, value):
        self.OD=value

    def getOA(self):
        return self.OA

    def getOD(self):
        return self.OD
#proficiencies
    def setWP(self, value):
        self.CWP = value

    def setDP(self, value):
        self.CDP = value

    def getWP(self):
        return self.CWP

    def getDP(self):
        return self.CDP
#Multiplier
    def setWMulti(self, value):
        self.wMulti = value

    def getWMulti(self):
        return self.wMulti
```

```
#Resistance
    def setRR(self, value):
        self.RR = value

    def getRRpercent(self, mytarget):
        return (self.RR /(self.RR + (mytarget.getLvl() * 100)))

    def setInRange(self, value):
        self.inRange = value

    def setInfront(self, value):
        self.infront = value

    def setBehind(self, value):
        self.behind = value

    def setInSight(self, value):
        self.insight = value

    def setInCombat(self, value):
        self.inCombat = value

    def isInRange(self):
        return self.inRange

    def isInfront(self):
        return self.infront

    def isBehind(self):
        return self.behind

    def isInSight(self):
        return self.inSight

    def isInCombat(self):
        return self.inCombat

    def setAtts(self, attPos, value):
        self.Atts[attPos]= value

    def getAtts(self, attPos):
        return self.Atts[attPos]

    def setLvl(self, newlvl):
        self.lvl = newlvl

    def getLvl(self):
        return self.lvl

    def setType(self,newtype):
        self.chartype = newtype

    def getType(self):
        return self.chartype

    def addNewChartoAggro(self, name, value):
        self.aggro[name] = value

    def addAggro(self, name, value):
        self.aggro[name] += value

    def calcAttInfluence(self):
        self.ResList[6] *= (1+ self.Atts[1]*0.025)
        self.ResList[6] *= (1+ self.Atts[5]*0.025)
        self.ResList[4] *= (1+ self.Atts[11]*0.05)
        self.ResList[5] *= (1+ self.Atts[7]*0.05)
        self.absorb += self.Atts[4] * 0.05
        self.crit += self.Atts[12] * 0.02
        self.block += self.Atts[5] * 0.035
        self.dodge += self.Atts[5] * 0.035

    def setHealth(self, value):
        self.Health = value

    def setHealthSelf(self):
        self.Health = (self.getAtts(7) + (self.getAtts(8)*10))

    def getHealth(self):
        return self.Health
```

```
#THE INTERFACE

import pyui, time
from AppCharacter import Character
from AppCombat import Combat

attribEditSpace = 2

WarriorAttList = [20,26,22,20,35,20,30,30,26,20,30,30,20]
RogueAttList = [20,30,22,20,20,40,20,23,18,15,30,26,35]

Char1 = Character(WarriorAttList)
Char2 = Character(RogueAttList)
Conflict = Combat()

Char1.setResList([100,89,105,67,30,98,20,10,10,10,10,10,10])
Char2.setResList([30,40,25,33,30,35,75,30,40,20,45,30,46])
Char1.Name = 'Left Character'
Char2.Name = 'Right Character'
Char1.setLowDam(30)
Char2.setLowDam(28)
Char1.setRR(89)
Char2.setRR(30)
Char1.setHighDam(40)
Char2.setHighDam(32)
Char1.setAttackSpeed(1.5)
Char2.setAttackSpeed(1)
Char1.setBlock(4)
Char1.setParry(4)
Char1.setDodge(4)
Char1.setCrit(4)
Char1.setOA(9)
Char1.setOD(10)
Char2.setBlock(0)
Char2.setParry(4)
Char2.setDodge(13)
Char2.setCrit(25)
Char2.setOA(8)
Char2.setOD(7)
Char1.setWP(10)
Char2.setWP(5)
Char1.setDP(9)
Char2.setDP(4)
Char1.setHealth(290)
Char2.setHealth(203)

def on2Press(button):
    Char1.calcAttInfluence()
    Char2.calcAttInfluence()
    print "Values have been adjusted!"

def onPress(button):
    Conflict.Battle(Char1, Char2)
    Conflict.Battle(Char2, Char1)

def onc1RREnter(edit):
    Char1.setRR(edit)

def onc2RREnter(edit):
    Char2.setRR(edit)

def onc1OAEnter(edit):
    Char1.setOA(edit)

def onc2OAEnter(edit):
    Char2.setOA(edit)

def onc1LDEnter(edit):
    Char1.setLowDam(edit)

def onc2LDEnter(edit):
    Char2.setLowDam(edit)

def onc1HDEnter(edit):
    Char1.setHighDam(edit)

def onc2HDEnter(edit):
    Char2.setHighDam(edit)

def onc1MULEnter(edit):
    Char1.WMulti(edit)

def onc2MULEnter(edit):
    Char2.Wmulti(edit)

def onc1WSPEnter(edit):
    Char1.setAttackSpeed(edit)

def onc2WSPEnter(edit):
    Char2.setAttackSpeed(edit)

def onc1CEnter(edit):
    Char1.Crit(edit)

def onc2CEnter(edit):
    Char2.Crit(edit)

def onc1ODEnter(edit):
    Char1.setOD(edit)

def onc2ODEnter(edit):
    Char2.setOD(edit)

def onc1PEnter(edit):
    Char1.setParry(edit)

def onc2PEnter(edit):
    Char2.setParry(edit)

def onc1DEnter(edit):
    Char1.setDodge(edit)

def onc2DEnter(edit):
    Char2.setDodge(edit)

def onc1BEnter(edit):
    Char1.setBlock(edit)

def onc2BEnter(edit):
    Char2.setBlock(edit)

def onc1AEnter(edit):
    Char1.setAbsorb(edit)

def onc2AEnter(edit):
    Char2.setAbsorb(edit)

def onc1WPEnter(edit):
    Char1.setWP(edit)

def onc2WPEnter(edit):
    Char2.setWP(edit)

def onc1DPEnter(edit):
```

```
    Char1.setDP(edit)

def onc2DPEnter(edit):
    Char2.setDP(edit)

def onInt1Enter(edit):
    Char1.setAtts(0,edit)

def onInt2Enter(edit):
    Char2.setAtts(0,edit)

def onWit1Enter(edit):
    Char1.setAtts(1,edit)

def onWit2Enter(edit):
    Char2.setAtts(1,edit)

def onConc1Enter(edit):
    Char1.setAtts(2,edit)

def onConc2Enter(edit):
    Char2.setAtts(2,edit)

def onSani1Enter(edit):
    Char1.setAtts(3,edit)

def onSani2Enter(edit):
    Char2.setAtts(3,edit)

def onMight1Enter(edit):
    Char1.setAtts(4,edit)

def onMight2Enter(edit):
    Char2.setAtts(4,edit)

def onAgi1Enter(edit):
    Char1.setAtts(5,edit)

def onAgi2Enter(edit):
    Char2.setAtts(5,edit)

def onFat1Enter(edit):
    Char1.setAtts(6,edit)

def onFat2Enter(edit):
    Char2.setAtts(6,edit)

def onVit1Enter(edit):
    Char1.setAtts(7,edit)

def onVit2Enter(edit):
    Char2.setAtts(7,edit)

def onHP1Enter(edit):
    Char1.setAtts(8,edit)

def onHP2Enter(edit):
    Char2.setAtts(8,edit)

def onEner1Enter(edit):
    Char1.setAtts(9,edit)

def onEner2Enter(edit):
    Char2.setAtts(9,edit)

def onIns1Enter(edit):
    Char1.setAtts(11,edit)

def onIns2Enter(edit):
    Char2.setAtts(11,edit)

def onWill1Enter(edit):
    Char1.setAtts(12,edit)

def onWill2Enter(edit):
    Char2.setAtts(12,edit)

def onPre1Enter(edit):
    Char1.setAtts(13,edit)

def onPre2Enter(edit):
    Char2.setAtts(13,edit)

class CSInterface:

    def drawMentalFrame(self):
        attChar1Frame = pyui.widgets.Frame(0, 30, 100, 200, "Mental atts")
        attChar1Frame.setLayout(pyui.layouts.GridLayoutManager(2,6))
        labelInt = pyui.widgets.Label("Int ")
        attChar1Frame.addChild(labelInt)
        c1IntEdit = pyui.widgets.NumberEdit("20", attribEditSpace, onInt1Enter,
3)
        attChar1Frame.addChild(c1IntEdit)
        labelWit = pyui.widgets.Label("Wit ")
        attChar1Frame.addChild(labelWit)
        c1WitEdit = pyui.widgets.NumberEdit("26", attribEditSpace, onWit1Enter,
3)
        attChar1Frame.addChild(c1WitEdit)
        labelKnow = pyui.widgets.Label("Know ")
        attChar1Frame.addChild(labelKnow)
        c1KnowEdit = pyui.widgets.Edit("26", attribEditSpace)
        attChar1Frame.addChild(c1KnowEdit)
        labelReas = pyui.widgets.Label("Reas ")
        attChar1Frame.addChild(labelReas)
        c1ReasEdit = pyui.widgets.Edit("27", attribEditSpace)
        attChar1Frame.addChild(c1ReasEdit)
        labelConc = pyui.widgets.Label("Conc ")
        attChar1Frame.addChild(labelConc)
        c1ConcEdit = pyui.widgets.NumberEdit("22", attribEditSpace,
onConc1Enter, 3)
        attChar1Frame.addChild(c1ConcEdit)
        labelSani = pyui.widgets.Label("Sani ")
        attChar1Frame.addChild(labelSani)
        c1SaniEdit = pyui.widgets.NumberEdit("20", attribEditSpace,
onSani1Enter, 3)
        attChar1Frame.addChild(c1SaniEdit)
        attChar1Frame.pack()

        attChar2Frame = pyui.widgets.Frame(924, 30, 100, 200, "Mental atts")
        attChar2Frame.setLayout(pyui.layouts.GridLayoutManager(2,6))
        label2Int = pyui.widgets.Label("Int ")
        attChar2Frame.addChild(label2Int)
        c2IntEdit = pyui.widgets.NumberEdit("20", attribEditSpace, onInt2Enter,
3)
        attChar2Frame.addChild(c2IntEdit)
        label2Wit = pyui.widgets.Label("Wit ")
        attChar2Frame.addChild(label2Wit)
        c2WitEdit = pyui.widgets.NumberEdit("30", attribEditSpace, onWit2Enter,
3)
        attChar2Frame.addChild(c2WitEdit)
        label2Know = pyui.widgets.Label("Know ")
        attChar2Frame.addChild(label2Know)
        c2KnowEdit = pyui.widgets.Edit("26", attribEditSpace)
        attChar2Frame.addChild(c2KnowEdit)
        label2Reas = pyui.widgets.Label("Reas ")
        attChar2Frame.addChild(label2Reas)
        c2ReasEdit = pyui.widgets.Edit("27", attribEditSpace)
        attChar2Frame.addChild(c2ReasEdit)
        label2Conc = pyui.widgets.Label("Conc ")
        attChar2Frame.addChild(label2Conc)
        c2ConcEdit = pyui.widgets.NumberEdit("22", attribEditSpace,
onConc2Enter, 3)
        attChar2Frame.addChild(c2ConcEdit)
        label2Sani = pyui.widgets.Label("Sani ")
        attChar2Frame.addChild(label2Sani)
        c2SaniEdit = pyui.widgets.NumberEdit("20", attribEditSpace,
onSani2Enter, 3)
        attChar2Frame.addChild(c2SaniEdit)
        attChar2Frame.pack()

    def drawPhysicalFrame(self):
        att2Char1Frame = pyui.widgets.Frame(0, 230, 100, 200, "Physical atts")
        att2Char2Frame = pyui.widgets.Frame(924, 230, 100, 200, "Physical
atts")

        att2Char1Frame.setLayout(pyui.layouts.GridLayoutManager(2,6))
        att2Char2Frame.setLayout(pyui.layouts.GridLayoutManager(2,6))

        labelMight = pyui.widgets.Label("Might ")
        label2Might = pyui.widgets.Label("Might ")
        att2Char1Frame.addChild(labelMight)
        att2Char2Frame.addChild(label2Might)

        c1MightEdit = pyui.widgets.NumberEdit("35", attribEditSpace,
onMight1Enter, 3)
        c2MightEdit = pyui.widgets.NumberEdit("20", attribEditSpace,
onMight2Enter, 3)
        att2Char1Frame.addChild(c1MightEdit)
        att2Char2Frame.addChild(c2MightEdit)

        labelAgi = pyui.widgets.Label("Agi ")
        label2Agi = pyui.widgets.Label("Agi ")
        att2Char1Frame.addChild(labelAgi)
        att2Char2Frame.addChild(label2Agi)

        c1AgiEdit = pyui.widgets.NumberEdit("20", attribEditSpace, onAgi1Enter,
3)
        c2AgiEdit = pyui.widgets.NumberEdit("40", attribEditSpace, onAgi2Enter,
3)
        att2Char1Frame.addChild(c1AgiEdit)
        att2Char2Frame.addChild(c2AgiEdit)

        labelFat = pyui.widgets.Label("Fat ")
        label2Fat = pyui.widgets.Label("Fat ")
        att2Char1Frame.addChild(labelFat)
        att2Char2Frame.addChild(label2Fat)

        c1FatEdit = pyui.widgets.NumberEdit("30", attribEditSpace, onFat1Enter,
3)
        c2FatEdit = pyui.widgets.NumberEdit("20", attribEditSpace, onFat2Enter,
3)
        att2Char1Frame.addChild(c1FatEdit)
        att2Char2Frame.addChild(c2FatEdit)

        labelVital = pyui.widgets.Label("Vital ")
        label2Vital = pyui.widgets.Label("Vital ")
        att2Char1Frame.addChild(labelVital)
        att2Char2Frame.addChild(label2Vital)

        c1VitalEdit = pyui.widgets.NumberEdit("30", attribEditSpace,
onVit1Enter, 3)
        c2VitalEdit = pyui.widgets.NumberEdit("23", attribEditSpace,
onVit2Enter, 3)
        att2Char1Frame.addChild(c1VitalEdit)
        att2Char2Frame.addChild(c2VitalEdit)

        labelHP = pyui.widgets.Label("HP ")
        label2HP = pyui.widgets.Label("HP ")
        att2Char1Frame.addChild(labelHP)
        att2Char2Frame.addChild(label2HP)

        c1HPEdit = pyui.widgets.NumberEdit("26", attribEditSpace, onHP1Enter,
3)
        c2HPEdit = pyui.widgets.NumberEdit("18", attribEditSpace, onHP2Enter,
3)
        att2Char1Frame.addChild(c1HPEdit)
        att2Char2Frame.addChild(c2HPEdit)

        labelApp = pyui.widgets.Label("App ")
        label2App = pyui.widgets.Label("App ")
        att2Char1Frame.addChild(labelApp)
        att2Char2Frame.addChild(label2App)

        c1AppEdit = pyui.widgets.Edit("26", attribEditSpace)
        c2AppEdit = pyui.widgets.Edit("24", attribEditSpace)
        att2Char1Frame.addChild(c1AppEdit)
        att2Char2Frame.addChild(c2AppEdit)

        att2Char1Frame.pack()
        att2Char2Frame.pack()

    def drawSpiritualFrame(self):
        att3Char1Frame = pyui.widgets.Frame(0, 430, 100, 200, "Spiritual atts")
        att3Char2Frame = pyui.widgets.Frame(924, 430, 100, 200, "Spiritual
atts")

        att3Char1Frame.setLayout(pyui.layouts.GridLayoutManager(2,6))
        att3Char2Frame.setLayout(pyui.layouts.GridLayoutManager(2,6))

        labelSpirEn = pyui.widgets.Label("Enrg ")
        label2SpirEn = pyui.widgets.Label("Enrg ")
        att3Char1Frame.addChild(labelSpirEn)
        att3Char2Frame.addChild(label2SpirEn)

        c1SpirEnEdit = pyui.widgets.NumberEdit("20", attribEditSpace,
onEner1Enter, 3)
        c2SpirEnEdit = pyui.widgets.NumberEdit("15", attribEditSpace,
onEner2Enter, 3)
        att3Char1Frame.addChild(c1SpirEnEdit)
        att3Char2Frame.addChild(c2SpirEnEdit)

        labelLuck = pyui.widgets.Label("Luck ")
        label2Luck = pyui.widgets.Label("Luck ")
        att3Char1Frame.addChild(labelLuck)
        att3Char2Frame.addChild(label2Luck)

        c1LuckEdit = pyui.widgets.Edit("27", attribEditSpace)
        c2LuckEdit = pyui.widgets.Edit("21", attribEditSpace)
        att3Char1Frame.addChild(c1LuckEdit)
        att3Char2Frame.addChild(c2LuckEdit)

        labelIns = pyui.widgets.Label("Ins ")
        label2Ins = pyui.widgets.Label("Ins ")
        att3Char1Frame.addChild(labelIns)
        att3Char2Frame.addChild(label2Ins)

        c1InsEdit = pyui.widgets.NumberEdit("30", attribEditSpace, onIns1Enter,
3)
        c2InsEdit = pyui.widgets.NumberEdit("30", attribEditSpace, onIns2Enter,
3)
        att3Char1Frame.addChild(c1InsEdit)
        att3Char2Frame.addChild(c2InsEdit)

        labelWill = pyui.widgets.Label("Will ")
        label2Will = pyui.widgets.Label("Will ")
        att3Char1Frame.addChild(labelWill)
        att3Char2Frame.addChild(label2Will)

        c1WillEdit = pyui.widgets.NumberEdit("30", attribEditSpace,
onWill1Enter, 3)
        c2WillEdit = pyui.widgets.NumberEdit("26", attribEditSpace,
onWill2Enter, 3)
        att3Char1Frame.addChild(c1WillEdit)
        att3Char2Frame.addChild(c2WillEdit)

        labelPre = pyui.widgets.Label("Pre ")
        label2Pre = pyui.widgets.Label("Pre ")
        att3Char1Frame.addChild(labelPre)
        att3Char2Frame.addChild(label2Pre)

        c1PreEdit = pyui.widgets.NumberEdit("20", attribEditSpace, onPre1Enter,
3)
        c2PreEdit = pyui.widgets.NumberEdit("35", attribEditSpace, onPre2Enter,
3)
        att3Char1Frame.addChild(c1PreEdit)
        att3Char2Frame.addChild(c2PreEdit)

        labelCha = pyui.widgets.Label("Cha ")
        label2Cha = pyui.widgets.Label("Cha ")
        att3Char1Frame.addChild(labelCha)
        att3Char2Frame.addChild(label2Cha)

        c1ChaEdit = pyui.widgets.Edit("22", attribEditSpace)
        c2ChaEdit = pyui.widgets.Edit("28", attribEditSpace)
        att3Char1Frame.addChild(c1ChaEdit)
        att3Char2Frame.addChild(c2ChaEdit)

        att3Char1Frame.pack()
        att3Char2Frame.pack()

    def drawCombatFrame(self):
        combatFrame = pyui.widgets.Frame(0, 630, 200, 140, "Combat Specs")
        combatFrame.setLayout(pyui.layouts.GridLayoutManager(2,6))

        combatFrame2 = pyui.widgets.Frame(824, 630, 200, 140, "Combat Specs")
        combatFrame2.setLayout(pyui.layouts.GridLayoutManager(2,6))

        c1WPEdit = pyui.widgets.NumberEdit("10", attribEditSpace, onc1WPEnter,
3)
        c2WPEdit = pyui.widgets.NumberEdit("5", attribEditSpace, onc2WPEnter,
3)
        c1DPEdit = pyui.widgets.NumberEdit("9", attribEditSpace, onc1DPEnter,
3)
        c2DPEdit = pyui.widgets.NumberEdit("4", attribEditSpace, onc2DPEnter,
3)

        c1OAEdit = pyui.widgets.NumberEdit("9", attribEditSpace, onc1OAEnter,
3)
        c2OAEdit = pyui.widgets.NumberEdit("8", attribEditSpace, onc2OAEnter,
3)
        c1ODEdit = pyui.widgets.NumberEdit("10", attribEditSpace, onc1ODEnter,
3)
        c2ODEdit = pyui.widgets.NumberEdit("7", attribEditSpace, onc2ODEnter,
3)

        c1WPLabel = pyui.widgets.Label("Weapon prof")
        c2WPLabel = pyui.widgets.Label("Weapon prof")
        c1DPLabel = pyui.widgets.Label("Defence prof")
        c2DPLabel = pyui.widgets.Label("Defence prof")

        c1OALabel = pyui.widgets.Label("Overall Atck")
        c2OALabel = pyui.widgets.Label("Overall Atck")
        c1ODLabel = pyui.widgets.Label("Overall Def")
        c2ODLabel = pyui.widgets.Label("Overall Def")

        combatFrame.addChild(c1WPLabel)
        combatFrame.addChild(c1WPEdit)
        combatFrame.addChild(c1DPLabel)
        combatFrame.addChild(c1DPEdit)
        combatFrame.addChild(c1OALabel)
        combatFrame.addChild(c1OAEdit)
        combatFrame.addChild(c1ODLabel)
        combatFrame.addChild(c1ODEdit)

        combatFrame2.addChild(c2WPLabel)
        combatFrame2.addChild(c2WPEdit)
        combatFrame2.addChild(c2DPLabel)
        combatFrame2.addChild(c2DPEdit)
        combatFrame2.addChild(c2OALabel)
        combatFrame2.addChild(c2OAEdit)
        combatFrame2.addChild(c2ODLabel)
        combatFrame2.addChild(c2ODEdit)

        combatFrame.pack()
        combatFrame2.pack()


    def drawDefFrame(self):
        defFrame = pyui.widgets.Frame(200, 630, 140, 140, "Defensive Specs")
        defFrame.setLayout(pyui.layouts.GridLayoutManager(2,6))

        defFrame2 = pyui.widgets.Frame(684, 630, 140, 140, "Defensive Specs")
        defFrame2.setLayout(pyui.layouts.GridLayoutManager(2,6))

        c1PEdit = pyui.widgets.NumberEdit("4", attribEditSpace, onc1PEnter, 3)
        c2PEdit = pyui.widgets.NumberEdit("4", attribEditSpace, onc2PEnter, 3)
        c1DEdit = pyui.widgets.NumberEdit("4", attribEditSpace, onc1DEnter, 3)
        c2DEdit = pyui.widgets.NumberEdit("13", attribEditSpace, onc2DEnter, 3)

        c1BEdit = pyui.widgets.NumberEdit("4", attribEditSpace, onc1BEnter, 3)
        c2BEdit = pyui.widgets.NumberEdit("0", attribEditSpace, onc2BEnter, 3)
        c1AEdit = pyui.widgets.NumberEdit("2", attribEditSpace, onc1AEnter, 3)
        c2AEdit = pyui.widgets.NumberEdit("0", attribEditSpace, onc2AEnter, 3)

        c1RREdit = pyui.widgets.NumberEdit("89", attribEditSpace, onc1RREnter,
3)
        c2RREdit = pyui.widgets.NumberEdit("30", attribEditSpace, onc2RREnter,
3)

        c1PLabel = pyui.widgets.Label("Parry")
        c2PLabel = pyui.widgets.Label("Parry")
        c1DLabel = pyui.widgets.Label("Dodge")
        c2DLabel = pyui.widgets.Label("Dodge")

        c1BLabel = pyui.widgets.Label("Block")
        c2BLabel = pyui.widgets.Label("Block")
        c1ALabel = pyui.widgets.Label("Absorb")
        c2ALabel = pyui.widgets.Label("Absorb")
        c1RRLabel = pyui.widgets.Label("Resist")
        c2RRLabel = pyui.widgets.Label("Resist")

        defFrame.addChild(c1PLabel)
        defFrame.addChild(c1PEdit)
        defFrame.addChild(c1DLabel)
        defFrame.addChild(c1DEdit)
        defFrame.addChild(c1BLabel)
        defFrame.addChild(c1BEdit)
        defFrame.addChild(c1ALabel)
        defFrame.addChild(c1AEdit)
        defFrame.addChild(c1RRLabel)
        defFrame.addChild(c1RREdit)


        defFrame2.addChild(c2PLabel)
        defFrame2.addChild(c2PEdit)
        defFrame2.addChild(c2DLabel)
        defFrame2.addChild(c2DEdit)
        defFrame2.addChild(c2BLabel)
        defFrame2.addChild(c2BEdit)
        defFrame2.addChild(c2ALabel)
        defFrame2.addChild(c2AEdit)
        defFrame2.addChild(c2RRLabel)
        defFrame2.addChild(c2RREdit)

        defFrame.pack()
        defFrame2.pack()

    def drawOffFrame(self):
        offFrame = pyui.widgets.Frame(340, 630, 140, 140, "Offensive Specs")
        offFrame.setLayout(pyui.layouts.GridLayoutManager(2,6))

        offFrame2 = pyui.widgets.Frame(544, 630, 140, 140, "Offensive Specs")
        offFrame2.setLayout(pyui.layouts.GridLayoutManager(2,6))

        c1LDEdit = pyui.widgets.NumberEdit("30", attribEditSpace, onc1LDEnter,
3)
        c2LDEdit = pyui.widgets.NumberEdit("28", attribEditSpace, onc2LDEnter,
3)
        c1HDEdit = pyui.widgets.NumberEdit("40", attribEditSpace, onc1HDEnter,
3)
        c2HDEdit = pyui.widgets.NumberEdit("32", attribEditSpace, onc2HDEnter,
3)

        c1WSPEdit = pyui.widgets.NumberEdit("1.5", attribEditSpace,
onc1WSPEnter, 3)
        c2WSPEdit = pyui.widgets.NumberEdit("1", attribEditSpace, onc2WSPEnter,
3)
        c1MULEdit = pyui.widgets.NumberEdit("2.5", attribEditSpace,
onc1MULEnter, 3)
        c2MULEdit = pyui.widgets.NumberEdit("2", attribEditSpace, onc2MULEnter,
3)

        c1CEdit = pyui.widgets.NumberEdit("4", attribEditSpace, onc1CEnter, 3)
        c2CEdit = pyui.widgets.NumberEdit("25", attribEditSpace, onc2CEnter, 3)

        c1LDLabel = pyui.widgets.Label("Low dmg")
        c2LDLabel = pyui.widgets.Label("Low dmg")
        c1HDLabel = pyui.widgets.Label("high dmg")
        c2HDLabel = pyui.widgets.Label("high dmd")

        c1WSPLabel = pyui.widgets.Label("Weap Spd")
        c2WSPLabel = pyui.widgets.Label("Weap Spd")
        c1MULLabel = pyui.widgets.Label("Multi")
        c2MULLabel = pyui.widgets.Label("Multi")

        c1CLabel = pyui.widgets.Label("Crit")
        c2CLabel = pyui.widgets.Label("Crit")

        offFrame.addChild(c1LDLabel)
        offFrame.addChild(c1LDEdit)
        offFrame.addChild(c1HDLabel)
        offFrame.addChild(c1HDEdit)
        offFrame.addChild(c1WSPLabel)
        offFrame.addChild(c1WSPEdit)
        offFrame.addChild(c1MULLabel)
        offFrame.addChild(c1MULEdit)
        offFrame.addChild(c1CLabel)
        offFrame.addChild(c1CEdit)

        offFrame2.addChild(c2LDLabel)
        offFrame2.addChild(c2LDEdit)
        offFrame2.addChild(c2HDLabel)
        offFrame2.addChild(c2HDEdit)
        offFrame2.addChild(c2WSPLabel)
        offFrame2.addChild(c2WSPEdit)
        offFrame2.addChild(c2MULLabel)
        offFrame2.addChild(c2MULEdit)
        offFrame2.addChild(c2CLabel)
        offFrame2.addChild(c2CEdit)

        offFrame.pack()
        offFrame2.pack()

    def drawButtonFrame(self):
        newFrame = pyui.widgets.Frame(380, 530, 270, 100, "Buttons")
        newFrame.setLayout(pyui.layouts.GridLayoutManager(1,2))
        newButton = pyui.widgets.Button("Take a round of battle!", onPress)
        newButton2 = pyui.widgets.Button("Influence values with your
attributes!", on2Press)
        newFrame.addChild(newButton2)
        newFrame.addChild(newButton)
        newFrame.pack()
    def drawInterface(self):
        def menuExit(item):
            pyui.quit()

        menu = pyui.widgets.Menu("Main menu")
        menu.addItem("Exit", menuExit)

        mBar = pyui.widgets.MenuBar()
        mBar.addMenu(menu)

        self.drawMentalFrame()
        self.drawPhysicalFrame()
        self.drawSpiritualFrame()
        self.drawCombatFrame()
        self.drawDefFrame()
        self.drawOffFrame()
        self.drawButtonFrame()
```

```
# MAIN

from pyui import core
from AppCharacter import Character
from AppCombat import Combat
import sys, pygame, AppInterface, pyui, time

class Application:
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def run(self):
        running = 1

        AppUI=AppInterface.CSInterface()
        AppUI.drawInterface()

        while running:
            core.draw()
            if core.update():
                pass
            else:
                running = 0

            for event in pygame.event.get():
                if event.type == pygame.QUIT: sys.exit()

def run():
    width = 1024
    height = 768
    core.init(width, height)
    pygame.init()


    app = Application(width, height)
    app.run()
    core.quit()
    raw_input()

if __name__ == '__main__':
    run()
```