UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Viido Kaur Lutsar

World of Warcraft Classic Warrior Simulator

Bachelor's Thesis (9 ECTS)

Supervisors: Ulrich Norbisrath, PhD
Mark Muhhin, MSc

Tartu 2021

Mängu World of Warcraft Classic Warriori simulaator

Lühikokkuvõte:

Lõputöö kirjeldab mängu World of Warcraft® Classic Warrior *class'*i simuleerivat tarkvaralahendust ja selle loomist. World of Warcraft Classic on MMORPG (*massively multiplayer online role-playing game),* mille keeruline *combat*-süsteem teeb raskeks oma tegelase efektiivsuse hindamise. Loodud tarkvaralahendus pakub sellele probleemile lahendust, simuleerides mängu *combat*-süsteemi, ning pakub kasutajasõbralikku liidest simulatsiooni parameetrite muutmiseks. See võimaldab kasutajatel oma tegelast optimiseerida ja mängus paremaid tulemusi saavutada.

Võtmesõnad:

World of Warcraft, arvutisimulatsioon, diskreetne sündmuspõhine simulatsioon, tarkvaraarendus

CERCS: P170 - arvutiteadus, arvutusmeetodid, süsteemid, juhtimine.

World of Warcraft Classic Warrior Simulator

Abstract:

This thesis describes the development of a software application, which simulates the World of Warcraft® Classic Warrior class. World of Warcraft Classic is a massively multi-player online role-playing game with a complicated combat system, which makes it difficult to measure your character's performance. The software provides a solution to that problem by simulating the game's combat system while providing a user-friendly interface to change the parameters of the simulation, allowing users to maximize their character's performance by finding the optimal setup for their character.

Keywords:

World of Warcraft, computer simulation, discrete event-driven simulation, software development

CERCS: P170 - computer science, numerical analysis, systems, control.

Table of Contents

1. Introduction

World of Warcraft® Classic is a MMORPG (massively multiplayer online role-playing game) created by Blizzard Entertainment®. One of the gameplay elements is raiding, in which players group up to fight more challenging enemies. Since the content is more challenging, good performance is highly valued. This has created a competitive community, where players strive to have the best performance.

The goal of this thesis is to create a computer simulator for the Warrior class in World of Warcraft Classic. The simulator will have a user-friendly graphical interface, which allows the user to choose different simulation parameters and display the results of the simulation. This tool helps players optimize their gameplay, leading to better performance.

Previously, the main tool used for this was an Excel spreadsheet, which calculates the fight results by approximating the game mechanics and averaging the results. A simulator would allow to replicate the exact game mechanics, and simulate the fight enough times to get an accurate result, making it a better choice than a spreadsheet, because some game mechanics are difficult to model accurately using a spreadsheet.

There exists a high demand for such tools – there are over 60 000 members in a Classic Warrior Discord intended for Warrior theorycrafting and discussion. These are all potential users. While there already exist multiple simulators for this, there is no single simulator, which includes all the features users would like to use.

Chapter 2 focuses on the background that is necessary to understand this topic. It will provide basic knowledge about World of Warcraft Classic and its gameplay elements. It also gives an overview of computer simulation and discrete-event simulation.

Chapter 3 will describe in detail the combat mechanics of the World of Warcraft Classic Warrior class and the formulas that the game's combat system uses. These are all features that the simulator will need to implement in order to replicate how the game's combat system works.

Chapter 4 contains an in-depth analysis of similar existing tools. Based on this analysis, the goals for this project are formulated.

Chapter 5 describes the planning process for development of the application and the implementation of the application.

Chapter 6 describes the build and deployment process for the application and the various tools used for this. Instructions for installing and running the application are also provided.

Chapter 7 evaluates the created application based on the goals set in section 4.6 and describes the challenges faced during the development process. It also talks about the future plans regarding the application.

In the process of writing this thesis, an application for simulating the Classic World of Warcraft Warrior class was created successfully according to the goals set in section 4.6. The application is available at https://github.com/Viido/WarriorSim-Classic with installation and usage instructions.

Some of the more specific terms in this thesis are defined in the Glossary (Appendix I).

2. Background

This chapter provides the background information that is necessary to understand the topic of this thesis. It describes World of Warcraft Classic and its gameplay elements. It also gives an overview about computer simulation and discrete-event simulation.

2.1 World of Warcraft Classic

World of Warcraft® Classic is a MMORPG (massively multiplayer online role-playing game) developed by Blizzard Entertainment®. It is a version of the main game (World of Warcraft) before the release of its first expansion, The Burning Crusade. Classic recreates the game as it was during patch 1.12.1 (September 2006) and is available to play alongside the current version of the game.

The original version of the game was at its peak popularity during October, 2010, reaching over 12 million players worldwide [1]. Since then, the game has seen a steady decline in player count, but with the release of Classic, there has been a resurgence in popularity.

Players control a character within the game world called Azeroth, exploring its landscape, completing quests, fighting monsters and interacting with other players or NPCs (non-player characters). During character creation, they can choose between two factions, Alliance and Horde, eight races: humans, dwarves, gnomes and night elves for Alliance and orcs, trolls, tauren and undead for Horde. The character can be one of nine different classes: Druid, Hunter, Mage, Paladin, Priest, Rogue, Shaman, Warlock and Warrior.

The gameplay can be divided into two different sections: PvE (player vs. environment), PvP (player vs. player). In PvE, players can fight different types of monsters, group up for dungeons and raids, which contain more difficult monsters and valuable loot[1]. In PvP, players can engage in open-world combat or join battlegrounds to fight groups of other players for objectives.

The combat system of the game is fairly complex and a character's power depends on many different factors: the character's stats, equipment, talents, buffs, debuffs and abilities. The damage a character deals is calculated using many different formulas, which does not make it clear for the player how much damage they are going to deal or which pieces of gear are

[1] Upon defeating a monster, players can loot the monster for valuable items, such as equipment and weapons.

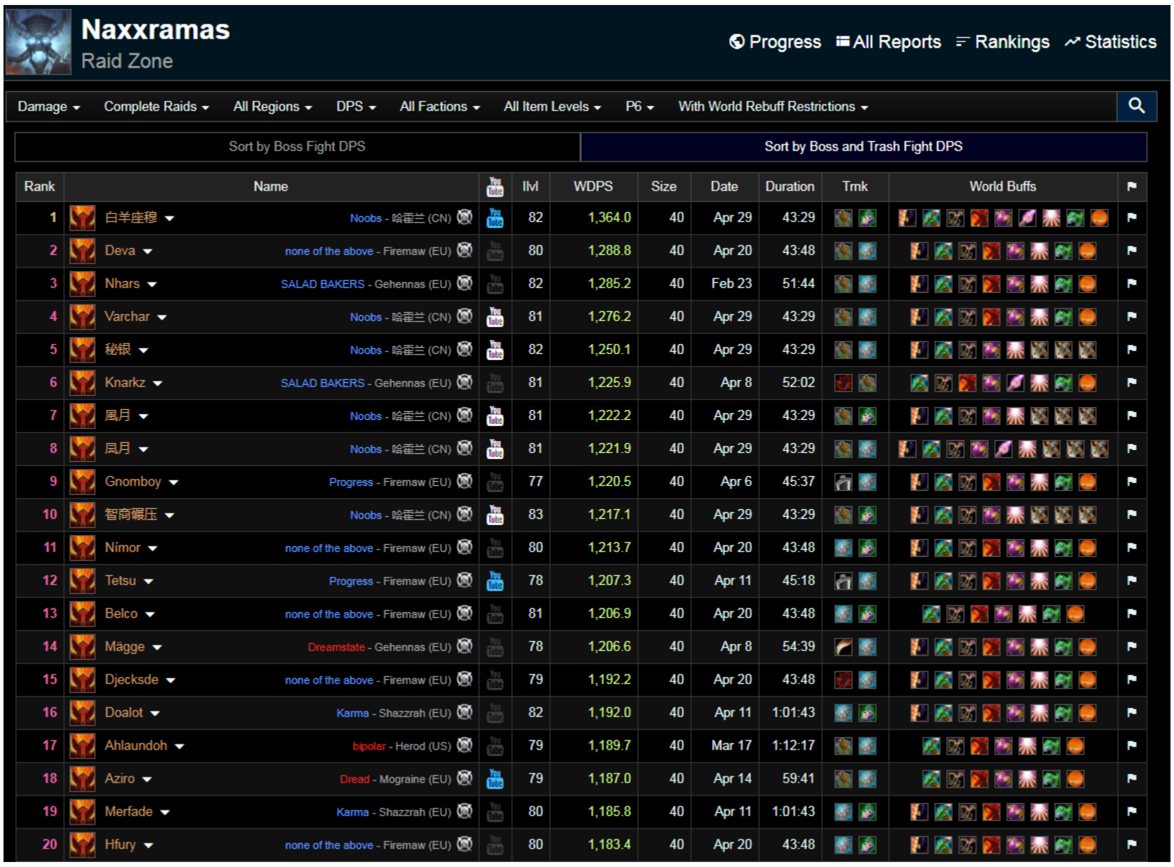
the most efficient. This is especially true for the Warrior class, since they have more abilities which affect their damage output compared to other classes.

For the hardcore[2] PvE community this is a problem, since their goal is to maximize their character's efficiency and compete with others. This problem can be solved with a computer simulator, which takes into account all combat parameters and simulates the combat mechanics to provide accurate results. This knowledge would allow players to make optimal decisions in-game and compare the efficiency of different gear pieces.

2.2 Motivation

Raiding in a serious Classic World of Warcraft guild is a competitive environment. Players strive to be the best for recognition, both inside the guild and world or region wide. Raids are usually logged using the Warcraftlogs site (Figure 1), which provides detailed breakdown of the raids and also rankings for the players, which further incentivises them to perform better. There is also the factor of distributing loot received from raids, since in many guilds loot is prioritized to the top performing players. This tool would help players make educated decisions in-game, helping them achieve their goals.

[2] Hardcore is a term used to describe the most competitive and committed players.



Rank	Name		Ilvl	WDPS	Size	Date	Duration	Trnk	World Buffs
1	白羊座穆 Noobs - 哈霍兰 (CN)		82	1,364.0	40	Apr 29	43:29		
2	Deva none of the above - Firemaw (EU)		80	1,288.8	40	Apr 20	43:48		
3	Nhars SALAD BAKERS - Gehennas (EU)		82	1,285.2	40	Feb 23	51:44		
4	Varchar Noobs - 哈霍兰 (CN)		81	1,276.2	40	Apr 29	43:29		
5	秘银 Noobs - 哈霍兰 (CN)		82	1,250.1	40	Apr 29	43:29		
6	Knarkz SALAD BAKERS - Gehennas (EU)		81	1,225.9	40	Apr 8	52:02		
7	風月 Noobs - 哈霍兰 (CN)		81	1,222.2	40	Apr 29	43:29		
8	凨月 Noobs - 哈霍兰 (CN)		81	1,221.9	40	Apr 29	43:29		
9	Gnomboy Progress - Firemaw (EU)		77	1,220.5	40	Apr 6	45:37		
10	智商碾压 Noobs - 哈霍兰 (CN)		83	1,217.1	40	Apr 29	43:29		
11	Nímor none of the above - Firemaw (EU)		80	1,213.7	40	Apr 20	43:48		
12	Tetsu Progress - Firemaw (EU)		78	1,207.3	40	Apr 11	45:18		
13	Belco none of the above - Firemaw (EU)		81	1,206.9	40	Apr 20	43:48		
14	Mägge Dreamstate - Gehennas (EU)		78	1,206.6	40	Apr 8	54:39		
15	Djecksde none of the above - Firemaw (EU)		79	1,192.2	40	Apr 20	43:48		
16	Doalot Karma - Shazzrah (EU)		82	1,192.0	40	Apr 11	1:01:43		
17	Ahlaundoh bipolar - Herod (US)		79	1,189.7	40	Mar 17	1:12:17		
18	Aziro Dread - Mograine (EU)		79	1,187.0	40	Apr 14	59:41		
19	Merfade Karma - Shazzrah (EU)		80	1,185.8	40	Apr 11	1:01:43		
20	Hfury none of the above - Firemaw (EU)		80	1,183.4	40	Apr 20	43:48		

Figure 1. Worldwide DPS rankings for the Naxxramas raid on the Warcraftlogs site

Due to how the class system is balanced in Classic, Warriors are the strongest damage dealing class. Because of this, competitive guilds bring as many Warriors as possible to a raid, while bringing only the minimum amount of other classes needed. This is also a reason why there are many Warrior simulators, but no simulators for other classes – Warriors are the most competitive class.

There is also personal motivation – I am a raider in one of the top guilds and currently use the existing tools to simulate my character. However, I often find that there are issues or missing features with the tools I use. By creating my own tool, I can ensure that all the features that I want to use are there. This also provides me with the opportunity to work closely with the community to improve the tool.

2.3 Computer simulation

The following paragraph is based on the Stanford Encyclopedia of Philosophy entry by Eric Winsberg [2].

Computer simulation is a method for studying the behaviour of various systems. It is similar to mathematical modelling, but performed on a computer. Simulations can be used to understand and explore systems, which might be too complicated for analytical solutions. Fundamentally, computer simulations try to solve or approximate the mathematical model that represents some kind of a system. If a simulation is believed to behave similarly enough to the system it is modelling, then the former can be studied to learn about the latter. Computer simulation is widely used in many fields of sciences, such as astrophysics, particle physics, materials science, engineering, fluid mechanics, climate science, economics, sociology and many more.

Because the combat system of World of Warcraft is complicated, a simulator is a better fit for studying it accurately, compared to an analytical solution like an Excel spreadsheet. By accurately implementing the combat system and mechanics of the game, a simulator can behave similarly to the combat system of the game, allowing us to study it.

According to the textbook by Norm Matloff [3], simulated systems can be classified into two categories, depending on how the system changes in relation to time: continuous and discrete. In a continuous simulation, the events progress in direct relation to time, while in a discrete simulation, the events happen at specific points in time.

The game’s combat system studied in this thesis falls under the discrete category, because the events happen at specific times and between the events, the system’s state does not change.

2.3.1 Discrete-event simulation

Discrete-event simulation models the behaviour of a system as a sequence of events which occur over a period of time [4]. Each event can change the state of the system and generate new events. In the case of this simulator, the events are different combat events that occur during the combat simulation.

After reviewing the textbook by Norm Matloff [3], I conclude that there are two ways of simulating such a system – using a fixed-increment time progression or an event-oriented time progression. In an event-oriented time progression, the time is advanced directly to the time of the next event. Due to events happening only at specific points in time, it is more efficient to use an event-oriented time progression, because the simulation does not have to process time increments, where the system’s state does not change. In order to implement an event-oriented time progression, an event set is used, which contains the set of pending events. In each iteration of the simulator's while loop, the simulator's time is advanced to the time of the next event, new events are generated and added to the event set.

The implementation of this concept for the simulator is described in section 6.4.3.

3. Combat system

This chapter describes the game mechanics for the Warrior class that need to be implemented in the simulator. While the stat and ability effects are easily observable in-game, the formulas are based on testing and research done by the community. A link to magey's classic-warrior wiki documenting the research is included in Appendix III.

This chapter describes the game mechanics from an in-game character's perspective.

3.1 Stats

Stats determine your character's offensive and defensive capabilities. They can be increased by talents, gear, buffs and decreased by debuffs. This section is based on the Classic stats guide written by Neryssa [5].

3.1.1 Primary stats

There are 5 primary stats in the game: Strength, Agility, Intellect, Stamina and Spirit.

Strength

- Increases your Attack Power by 2 for each point of Strength.
- Increases your Block Value by 1 for each 20 points of Strength.

Agility

- Increases your Armor by 2 for every point of Agility.
- Increases your Crit Chance by 1% for every 20 points of Agility.
- Increases your Dodge Chance by 1% for every 20 points of Agility.

Stamina

- Increases your Health by 10 for every point of Stamina.

Intellect and Spirit do not provide any useful effects for Warriors, since those stats mainly affect caster class archetypes.

3.1.2 Offensive stats

There are 6 melee offensive stats: Armor Penetration, Haste, Attack Power, Crit Chance, Hit Chance and Weapon Skill.

Armor Penetration

- Ignores a flat amount of target’s Armor.

Haste

- Increases your Weapon Speed by a percentage amount.

Attack Power

- Increases your base melee DPS (damage per second) by 1 for every 14 points of Attack Power.

Crit Chance

- Increases your chance to score a critical hit.

Hit Chance

- Reduces the chance to miss your attacks against a target.

Weapon Skill

- Reduces the chance that your attacks against a target result in a miss, dodge, parry, block or glancing blow.
- Increases the damage of glancing blows.

3.1.3 Defensive stats

There are 7 defensive stats: Resistance, Defense Skill, Armor, Block Chance, Block Value, Dodge Chance and Parry Chance.

Resistance

- There are 5 types of resistances in the game: Fire, Frost, Arcane, Nature and Shadow.
- Increases your chance to resist spells.
- Reduces your damage taken from spells.

Defense Skill

- Reduces your chance of being critically hit or hit.
- Increases your Dodge Chance, Parry Chance and Block Chance.

Armor

- Reduces your physical damage taken.

Block Chance

- Increases your chance to block an incoming attack by a percentage amount.

Block Value

- Increases the amount of damage you block.

Dodge Chance

- Increases your chance to dodge an incoming attack.

Parry Chance

- Increases your chance to parry an incoming attack.

3.2 Abilities

Warrior abilities use rage as a resource. Rage can be generated from auto-attacks and damage taken and has a limited to a maximum amount of 100 rage. Ability casts trigger a 1.5 second global cooldown, during which other abilities cannot be casted. Some abilities are exempt from this rule.

Bloodthirst

- 6 second cooldown.
- Costs 30 Rage.
- Deals damage equal to 45% of your Attack Power.

Whirlwind

- 10 second cooldown.
- Costs 25 rage.
- Deals weapon damage to up to 4 enemies within 8 yards.

Heroic Strike

- Costs 15 rage.
- Increases the damage and threat on your next swing by a flat amount.
- Not affected by the global cooldown.

Cleave

- Costs 20 rage.
- Increases the damage and threat on your next swing by a flat amount.
- Can hit up to 2 targets.

- Not affected by the global cooldown.

Hamstring

- Costs 10 rage.
- Deals a flat amount of damage and threat.

Execute

- Costs 15 rage and consumes all extra rage.
- Deals 600 damage and converts each extra point of rage into 15 additional damage.
- Only usable on enemies that have 20% or less health.

According to the guide written by TheKrugLife [6], the optimal ability rotation is to use Bloodthirst and Whirlwind on cooldown while using excess rage for Heroic Strikes. During the last 20% of enemy health, Execute should be used instead. In multi-target situations, Cleave is used instead of Heroic Strike.

3.3 Talents

The talent system is a point-based progression system for the in-game character. As the character levels up, talent points are gained which can be spent to improve abilities, learn new abilities and gain stats. For the Warrior class, there are three different talent trees: Arms, Fury and Protection. The Arms and Fury trees focus on dealing damage, while the Protection tree focuses on defensive capabilities. A talent calculator is included in Appendix III.

3.4 Buffs and debuffs

Buffs are temporary beneficial effects that in-game characters can gain, which improve their stats. Debuffs are the opposite of buffs, they have detrimental effects on your character.

There are many kinds of buffs in Classic such as world buffs, which are very powerful and can be gathered around the world and consumables, such as potions or food, which grant you buffs. A list of world buffs and consumables is included in Appendix III.

3.5 Formulas

The formulas described in this section are based on community research done mainly during the Classic World of Warcraft beta, which was documented in magey’s classic-warrior wiki (see Appendix III). In order to figure out the exact numbers, players logged their combat and later the combat logs were analysed to reverse-engineer the formulas the game uses.

Some of the formulas were also confirmed by Blizzard, this is also documented in the repository.

3.5.1 Attack table

The attack table is used to calculate the outcome of an attack. There are 7 possible outcomes: miss, hit, crit, glancing blow, parry, dodge or block. Parries and blocks can only occur while attacking the enemy from the front and since damage dealers should be attacking from behind, we can ignore them. Glancing blows can only occur on auto-attacks. When Heroic Strike or Cleave is queued, the dual-wield penalty to miss chance does not occur.

All attack table rolls cannot be less than 0% and must add up to 100%. Miss, parry, dodge, block and glancing blow rolls take priority over the critical hit roll and the critical hit roll takes priority over the hit roll.

Firstly, let's define the skill difference variable to be used in the attack table formulas:

$$skillDiff = targetDefenseSkill - playerWeaponSkill$$

Miss Chance

If the skill difference is 10 or more, then the following formula is used:

$$missChance = 5\% + skillDiff * 0.2\%$$

Otherwise, the formula is:

$$missChance = 5\% + skillDiff * 0.1\%$$

If the player is dual-wielding, the following penalty applies to auto-attacks:

$$missChance = missChance + 19\%$$

Dodge Chance

$$dodgeChance = 5\% + skillDiff * 0.1\%$$

Glancing Blow Chance

$$glancingChance = 10\% + (targetLevel - playerLevel) * 10\%$$

Critical Chance

Critical Chance is simply summed up from your stats. However, some penalties apply.

If the target is higher level than the player, then Critical Chance is reduced by 1% for each difference in level.

If the target is 3 levels higher than the player, then 1.8% Critical Chance is subtracted from the Critical Chance you gain from auras.

3.5.2 Damage calculation

Auto-attacks

The damage of an auto-attack is calculated from the weapon's base damage, base speed and the character's Attack Power.

$$damage = baseDamage + \frac{attackPower}{14} * baseSpeed$$

For off-hand auto-attacks a 50% damage penalty applies, which can be reduced by the Dual-Wield Specialization talent.

Abilities

If an ability is an instant attack based on weapon damage, then the damage gain from Attack Power is normalized. The normalized speed is 1.7 for daggers, 2.4 for one-handed weapons and 3.3 for two-handed weapons.

$$damage = baseDamage + \frac{attackPower}{14} * normalizedSpeed$$

Attack Table Rolls

Misses, parries and dodges deal zero damage. Critical hits deal double damage, which can be increased by the Impale talent for abilities. Blocks subtract the target's block value from the damage amount.

Glancing blows use the following formula for damage:

$$lowEnd = damage * (1.3 - 0.05 * (targetDefenseSkill - playerWeaponSkill))$$

$$highEnd = damage * (1.2 - 0.03 * (targetDefenseSkill - playerWeaponSkill))$$

A glancing blow can deal damage in between the low end and high end values. The low end is capped at a maximum of 91% and the high end is capped at a maximum of 99%.

Armor Mitigation

The damage dealt is also reduced by the target's Armor.

$$armorMitigation = \frac{targetArmor}{targetArmor + 400 + 85 * attackerLevel}$$

Rage Generation

Rage can be generated from auto-attack damage or damage taken.

$$rageGenerated = \frac{damageDone}{230.6} * 7.5$$

$$rageGenerated = \frac{damageTaken}{230.6} * 2.5$$

3.6 Extra attacks

Extra attacks are instant main-hand swings triggered by certain proc[3] items and abilities. An extra attack resets the main-hand swing timer. If Heroic Strike or Cleave is queued while the extra attack occurs, then they will be cast immediately.

Proc abilities such as Windfury Totem and Sword Specialization can proc other extra attacks, but not themselves. They can proc once in a chain of extra attack procs.

Proc items such as Ironfoe or Hand of Justice can self-proc and also proc other extra attacks. They can proc multiple times in a chain of extra attack procs.

Multiple extra attack procs are only possible from instant abilities that use your main-hand.

Whirlwind and Cleave have a single roll to proc extra attacks, no matter how many targets they hit.

3.7 Impact on the development of the simulator

It is important that the character is fully configurable in the simulator, so that users can reproduce the same configurations as their characters have in-game. It should be intuitive and easy to configure a character in the simulator – this requires a clean user interface.

The accuracy of the simulator depends on the combat mechanics being implemented correctly. Unit tests will have to be written to ensure that the combat mechanics are implemented correctly.

[3] Proc is an acronym for "programmed random occurrence". It is a common term used in games for effects that have a random chance to occur.

In the next chapter, it will be analysed how other existing tools deal with these problems and how it should be implemented for the created simulator. Chapter 7 will evaluate how this was achieved.

4. Analysis of existing tools

This chapter will describe different existing tools and analyse them. In the end all the pros and cons will be listed in a comparison grid. Based on this analysis, the scope of the simulator written for this thesis will be formed. Links to the tools are included in Appendix II.

4.1 WarriorSim

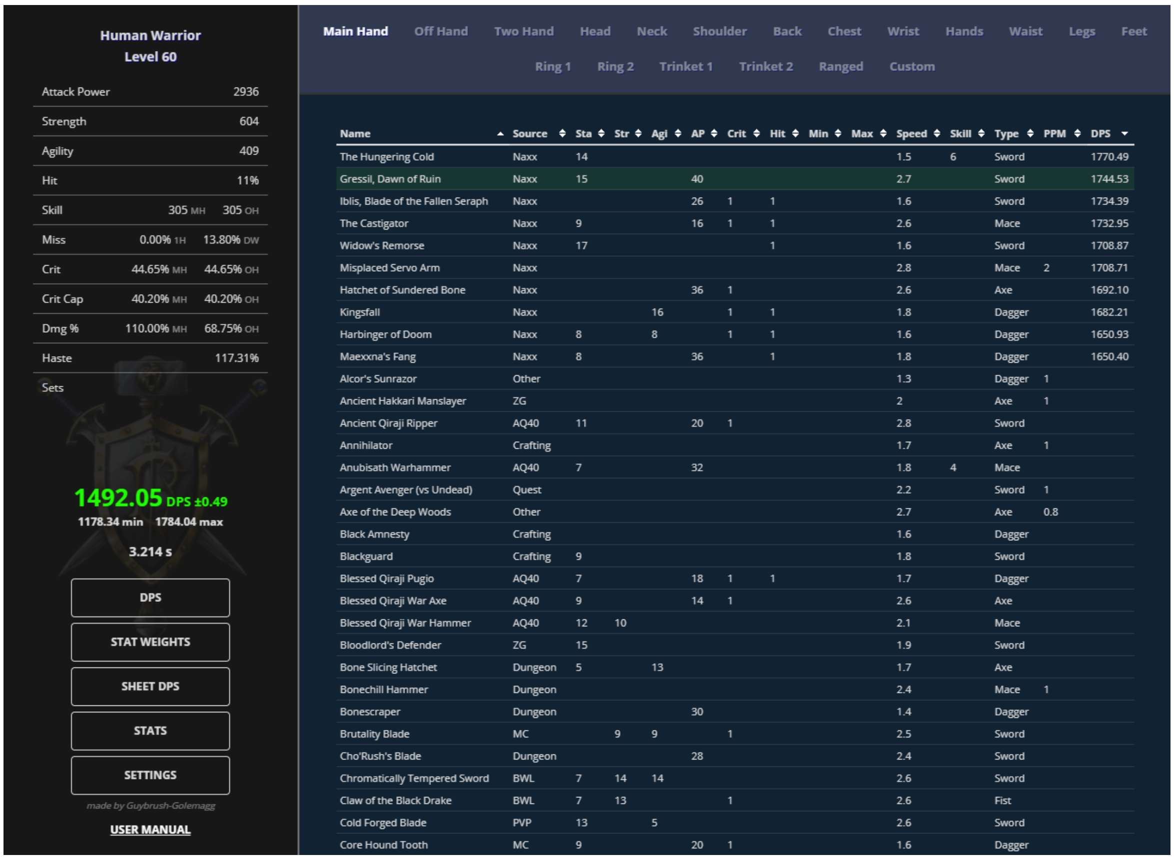
WarriorSim (Figure 2) is a web application written in JavaScript and C++ by GuybrushGit and other contributors. It has a graphical user interface for selecting fight parameters, gear, buffs, talents and rotation. The interface is user-friendly and looks clean. Boss debuffs are not implemented, however boss armor value can be specified manually. While there is a graphical talent selection, it does not implement the talent selection logic as it works in-game.

The main view displays character stats and contains the gear selection in a list format. While there is no search function for items, the gear is separated by item slots and is sortable by the item's name, source, stats and the simulated DPS. Displayed items can also be filtered by source in the settings menu.

This simulator has two ways to run the simulations, you can either simulate a fight with the current setup or simulate each item in the selected item slot. It also allows you to simulate gear enchants separately. However, it does not offer a way to easily compare different sets of gear and parameters.

It does not allow you to import or export profiles, but the current configuration is saved between uses.

The software does not have unit tests or combat logs for debugging and the GitHub page does not provide a documentation on how the mechanics and formulas work. This makes hard to verify the accuracy of the tool.



Human Warrior
Level 60

Attack Power		2936
Strength		604
Agility		409
Hit		11%
Skill	305 MH	305 OH
Miss	0.00% 1H	13.80% DW
Crit	44.65% MH	44.65% OH
Crit Cap	40.20% MH	40.20% OH
Dmg %	110.00% MH	68.75% OH
Haste		117.31%
Sets		

1492.05 DPS ±0.49
1178.34 min 1784.04 max
3.214 s

Name	Source	Sta	Str	Agi	AP	Crit	Hit	Min	Max	Speed	Skill	Type	PPM	DPS
The Hungering Cold	Naxx	14								1.5	6	Sword		1770.49
Gressil, Dawn of Ruin	Naxx	15			40					2.7		Sword		1744.53
Iblis, Blade of the Fallen Seraph	Naxx				26	1	1			1.6		Sword		1734.39
The Castigator	Naxx	9			16	1	1			2.6		Mace		1732.95
Widow's Remorse	Naxx	17					1			1.6		Sword		1708.87
Misplaced Servo Arm	Naxx									2.8		Mace	2	1708.71
Hatchet of Sundered Bone	Naxx				36	1				2.6		Axe		1692.10
Kingsfall	Naxx			16		1	1			1.8		Dagger		1682.21
Harbinger of Doom	Naxx	8		8		1	1			1.6		Dagger		1650.93
Maexxna's Fang	Naxx	8			36		1			1.8		Dagger		1650.40
Alcor's Sunrazor	Other									1.3		Dagger	1	
Ancient Hakkari Manslayer	ZG									2		Axe	1	
Ancient Qiraji Ripper	AQ40	11			20	1				2.8		Sword		
Annihilator	Crafting									1.7		Axe	1	
Anubisath Warhammer	AQ40	7			32					1.8	4	Mace		
Argent Avenger (vs Undead)	Quest									2.2		Sword	1	
Axe of the Deep Woods	Other									2.7		Axe	0.8	
Black Amnesty	Crafting									1.6		Dagger		
Blackguard	Crafting	9								1.8		Sword		
Blessed Qiraji Pugio	AQ40	7			18	1	1			1.7		Dagger		
Blessed Qiraji War Axe	AQ40	9			14	1				2.6		Axe		
Blessed Qiraji War Hammer	AQ40	12	10							2.1		Mace		
Bloodlord's Defender	ZG	15								1.9		Sword		
Bone Slicing Hatchet	Dungeon	5		13						1.7		Axe		
Bonechill Hammer	Dungeon									2.4		Mace	1	
Bonescraper	Dungeon				30					1.4		Dagger		
Brutality Blade	MC		9	9		1				2.5		Sword		
Cho'Rush's Blade	Dungeon				28					2.4		Sword		
Chromatically Tempered Sword	BWL	7	14	14						2.6		Sword		
Claw of the Black Drake	BWL	7	13			1				2.6		Fist		
Cold Forged Blade	PVP	13		5						2.6		Sword		
Core Hound Tooth	MC	9			20	1				1.6		Dagger		

Figure 2. WarriorSim

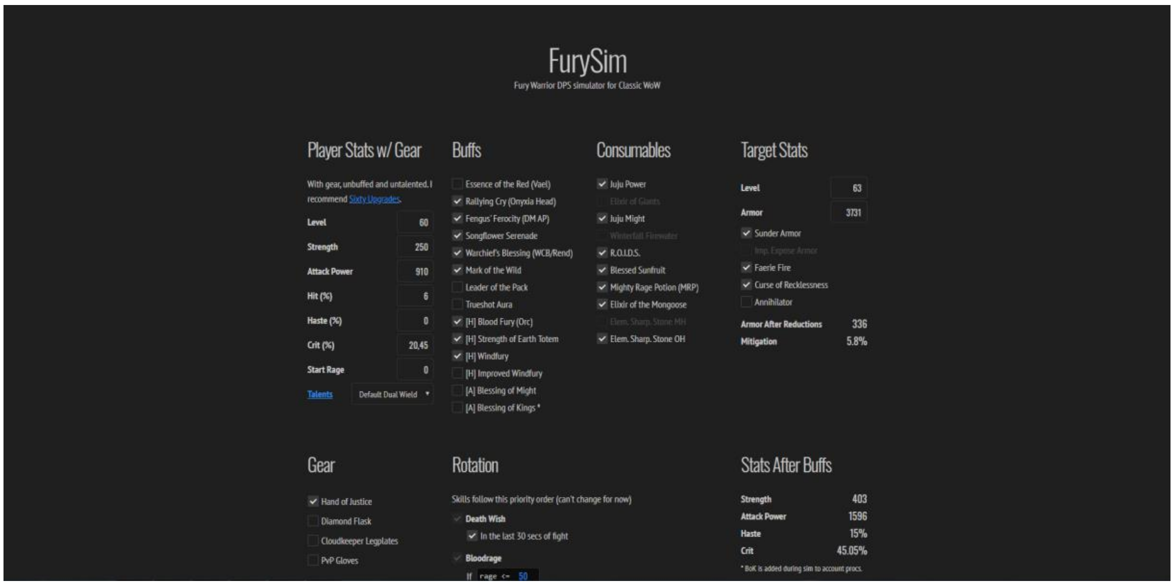
I like that the tool has a clean user interface, which makes it easy to use, however, it is hard to get an overview of your character's configuration, because it is spread across different tabs. It would be better if the talent selection logic was implemented in order to avoid users creating unrealistic talent setups by mistake. It is also important to have unit tests and combat logging to be able to guarantee accuracy of the simulation.

4.2 FurySim

FurySim (Figure 3) is a web application written in JavaScript by rafaelpimpa using the Vue.js framework and TypeScript. It has a graphical user interface for selecting fight parameters, buffs, boss debuffs and rotation. However, stats have to be manually input and talents have to be imported from Wowhead's talent calculator or chosen from three default options. Since multiple parameters have to be either imported or manually edited, this is not a very user-friendly application.

There is only one view, so you have to scroll down to see all the configurations. It is fairly easy to get a clear overview of your character's configuration and I like the user interface overall. A big drawback is that there is no item selection.

This tool allows you to import and export profiles and the GitHub page provides documentation for the implemented mechanics. However, the tool has no unit tests and no combat logs for debugging.



FurySim

Fury Warrior DPS simulator for Classic WoW

Player Stats w/ Gear

With gear, unbuffed and untalented. I recommend Sixty Upgrades.

Level	60
Strength	250
Attack Power	910
Hit (%)	6
Haste (%)	0
Crit (%)	20,45
Start Rage	0
Talents	Default Dual Wield

Buffs

- ☐ Essence of the Red (Vael)
- ✓ Rallying Cry (Onyxia Head)
- ✓ Fengus' Ferocity (DM AP)
- ✓ Songflower Serenade
- ✓ Warchief's Blessing (WCB/Rend)
- ✓ Mark of the Wild
- ☐ Leader of the Pack
- ☐ Trueshot Aura
- ✓ [H] Blood Fury (Orc)
- ✓ [H] Strength of Earth Totem
- ✓ [H] Windfury
- ☐ [H] Improved Windfury
- ☐ [A] Blessing of Might
- ☐ [A] Blessing of Kings *

Consumables

- ✓ Juju Power
- ☐ Elixir of Giants
- ✓ Juju Might
- ☐ Winterfall Firewater
- ✓ R.O.I.D.S.
- ✓ Blessed Sunfruit
- ✓ Mighty Rage Potion (MRP)
- ✓ Elixir of the Mongoose
- ☐ Elem. Sharp. Stone MH
- ✓ Elem. Sharp. Stone OH

Target Stats

Level	63
Armor	3731

- ✓ Sunder Armor
- ☐ Imp. Expose Armor
- ✓ Faerie Fire
- ✓ Curse of Recklessness
- ☐ Annihilator

Armor After Reductions	336
Mitigation	5.8%

Gear

- ✓ Hand of Justice
- ☐ Diamond Flask
- ☐ Cloudkeeper Legplates
- ☐ PvP Gloves

Rotation

Skills follow this priority order (can't change for now)

- ✓ Death Wish
  - ✓ In the last 30 secs of fight
- ✓ Bloodrage
  - If `rage <= 50`

Stats After Buffs

Strength	403
Attack Power	1596
Haste	15%
Crit	45.05%

* BoK is added during sim to account procs.

Figure 3. FurySim.

I like that the tool provides a clear overview of your character configuration, however it is missing graphical item and talent selections. Like with WarriorSim, the tool has the problem with verifying accuracy, since it has no unit tests or combat logging.

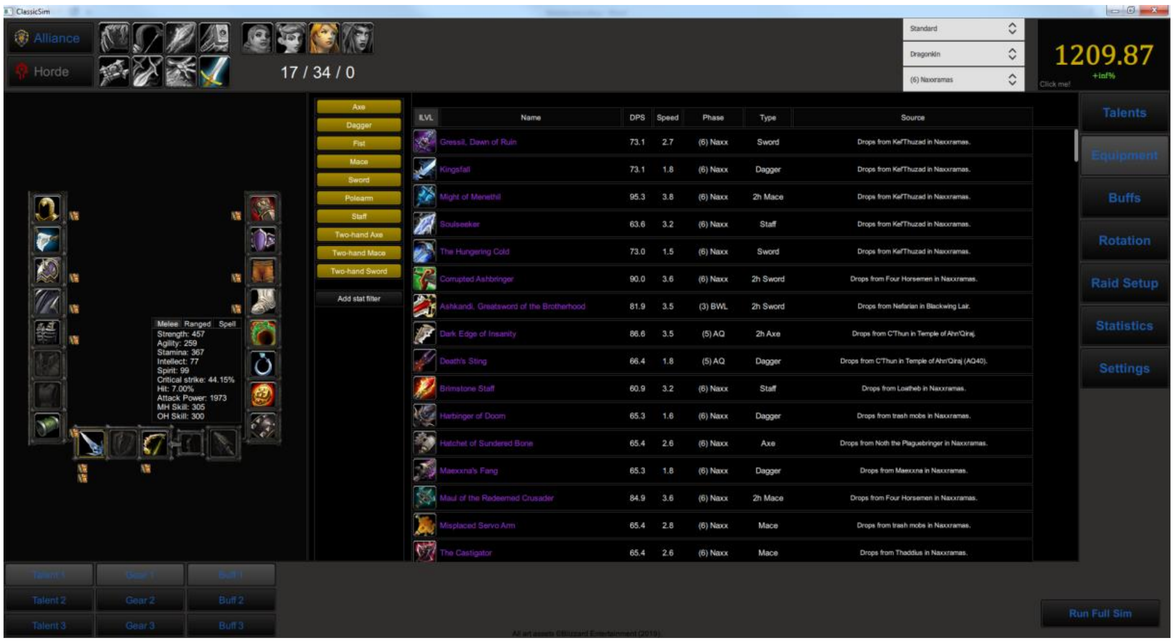
4.3 ClassicSim

ClassicSim (Figure 4) is a desktop application written in C++ by timhul. It has a graphical user interface for selecting fight parameters, talents, buffs, boss debuffs, gear and rotation. It also supports multithreading and saves your configurations. The rotations can't be configured, but the tool offers multiple pre-set rotations.

In the main view, the character configuration and simulator settings are placed in different tab. The selected tab is displayed in the middle. It allows you to configured three different character setups, but only simulates one setup at once. This makes it easier to work with multiple setups.

The simulation results are displayed under the statistics section, which provides detailed breakdowns of the simulation, however it does not allow you to easily compare the results of different setups, since only one set of parameters can be simulated at once.

The GitHub page provides in-depth documentation of the implemented mechanics and there are extensive unit tests, however, there is no combat logging.



Melee Ranged Spell
Strength: 457
Agility: 259
Stamina: 367
Intellect: 77
Spirit: 99
Critical strike: 44.15%
Hit: 7.00%
Attack Power: 1973
MH Skill: 305
OH Skill: 300

ILVL	Name	DPS	Speed	Phase	Type	Source
	Gressil, Dawn of Ruin	73.1	2.7	(6) Naxx	Sword	Drops from Kel'Thuzad in Naxxramas.
	Kingsfall	73.1	1.8	(6) Naxx	Dagger	Drops from Kel'Thuzad in Naxxramas.
	Might of Menethil	95.3	3.8	(6) Naxx	2h Mace	Drops from Kel'Thuzad in Naxxramas.
	Soulseeker	63.6	3.2	(6) Naxx	Staff	Drops from Kel'Thuzad in Naxxramas.
	The Hungering Cold	73.0	1.5	(6) Naxx	Sword	Drops from Kel'Thuzad in Naxxramas.
	Corrupted Ashbringer	90.0	3.6	(6) Naxx	2h Sword	Drops from Four Horsemen in Naxxramas.
	Ashkandi, Greatsword of the Brotherhood	81.9	3.5	(3) BWL	2h Sword	Drops from Nefarian in Blackwing Lair.
	Dark Edge of Insanity	86.6	3.5	(5) AQ	2h Axe	Drops from C'Thun in Temple of Ahn'Qiraj.
	Death's Sting	66.4	1.8	(5) AQ	Dagger	Drops from C'Thun in Temple of Ahn'Qiraj (AQ40).
	Brimstone Staff	60.9	3.2	(6) Naxx	Staff	Drops from Loatheb in Naxxramas.
	Harbinger of Doom	65.3	1.6	(6) Naxx	Dagger	Drops from trash mobs in Naxxramas.
	Hatchet of Sundered Bone	65.4	2.6	(6) Naxx	Axe	Drops from Noth the Plaguebringer in Naxxramas.
	Maexxna's Fang	65.3	1.8	(6) Naxx	Dagger	Drops from Maexxna in Naxxramas.
	Maul of the Redeemed Crusader	84.9	3.6	(6) Naxx	2h Mace	Drops from Four Horsemen in Naxxramas.
	Misplaced Servo Arm	65.4	2.8	(6) Naxx	Mace	Drops from trash mobs in Naxxramas.
	The Castigator	65.4	2.6	(6) Naxx	Mace	Drops from Thaddius in Naxxramas.

Figure 4. ClassicSim.

I like that this tool has the option to save multiple setups at once, but for some reason, this is limited to only three setups. I would prefer if the tool gave a better overview of your character by not separating the different configuration options into different tabs. This tool is so far the only tool with unit tests and it also has documentation for the combat mechanics, so I can trust the accuracy of this tool better. However, it also does not have combat logging, which would make it easier to debug the combat simulation.

4.4 Wow fury sim

Wow fury sim (Figure 5) is a web application written by albinoyoda in C++. It has a basic HTML based user interface for selecting gear, talents, buffs, rotation and fight parameters. It also has support for multi-target fights, unlike the other tools. Configurations need to be manually saved and loaded using a file, instead of automatically saved the configuration between uses.

The main view is separated into collapsible sections, which makes it difficult to get a clear overview of the character's configuration. It has three modes for simulation: standard, in which the selected configuration is simulated, multi-mode, in which multiple items can be selected and the simulator finds the best combination of them and comparison mode, in which a configuration file can be uploaded and compared against the current configuration.

It also has simulator settings which compute stat weights, damage per rage values for abilities and values of different talents.

This tool has both unit tests and combat logging, but there is no full documentation for combat mechanics, only a link to a source, which they are based on.

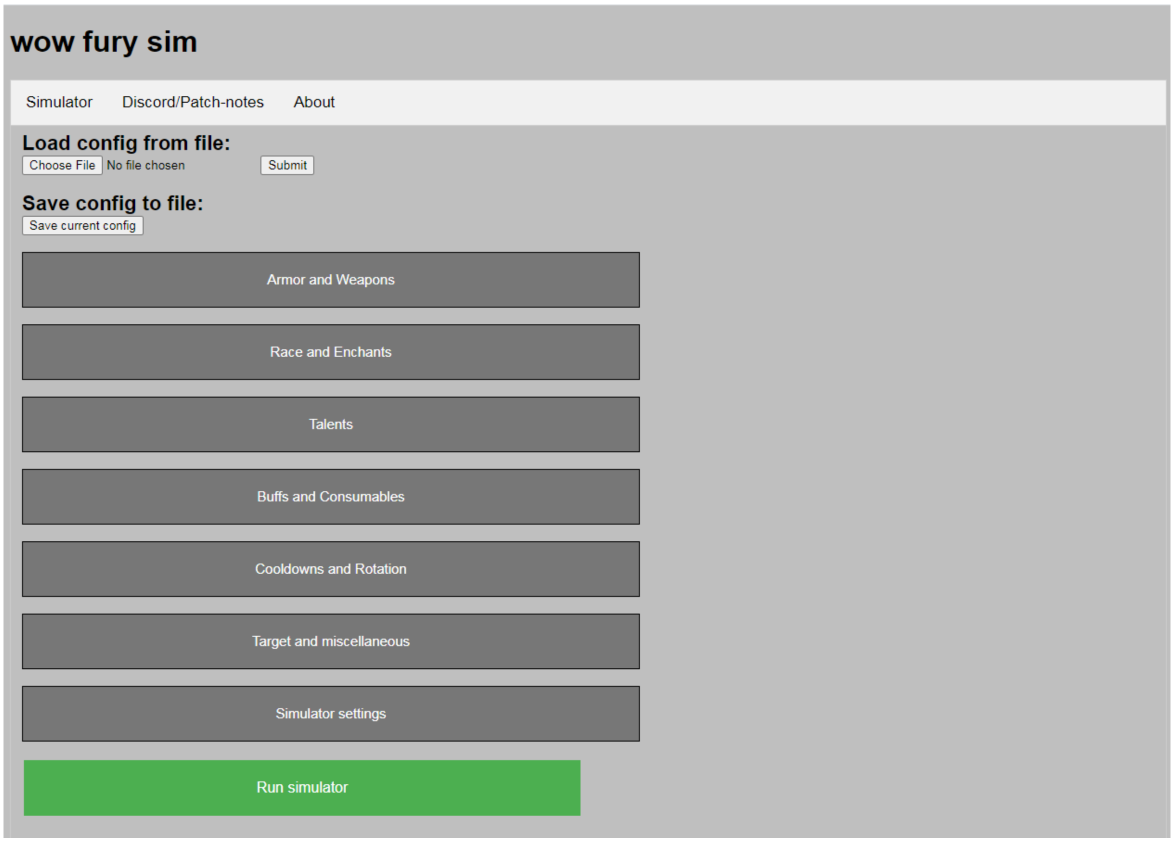


Figure 5. Wow fury sim

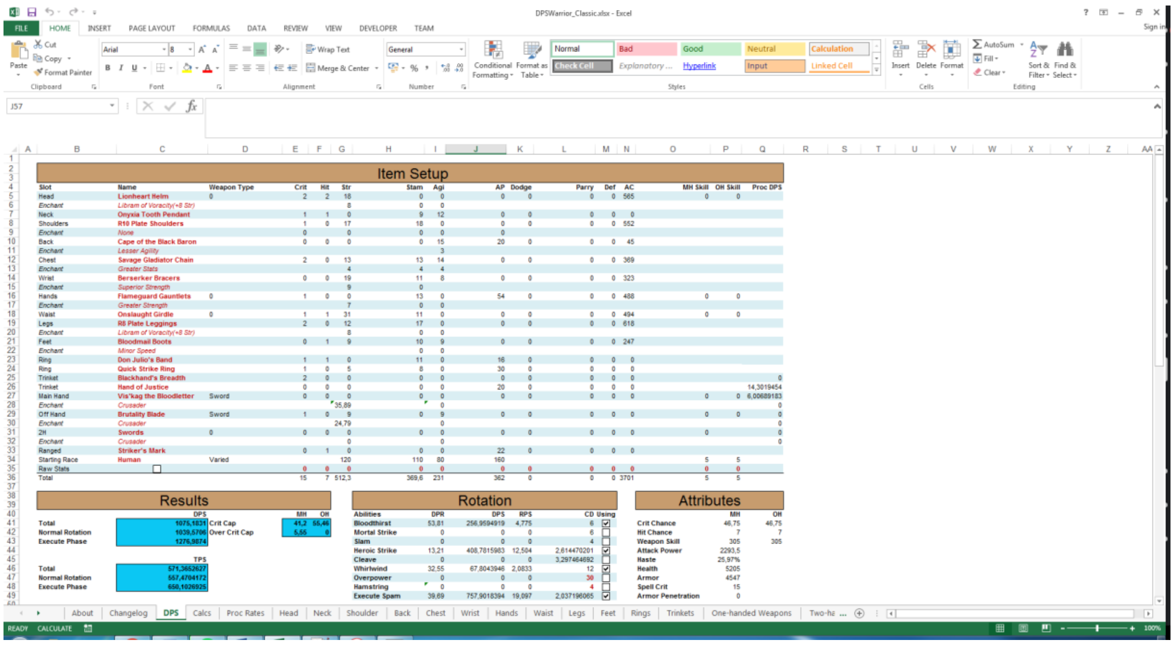
One of the drawbacks for this tool is the basic user interface, which is not very user friendly. It does not save configurations between uses and users have to do it manually. While it provides many different options on how to simulate the fight, I personally have not found a need for them. The support for multi-target fights is a great feature, since these types of fights happen often in the game. With the simulator having unit tests and combat logging, it is also possible to verify its accuracy.

4.5 DPSWarrior_Classic

DPSWarrior_Classic (Figure 6) is an Excel spreadsheet created during the original version of the game. It has been improved and modified by various developers and most recently by the author of this thesis. It allows you to select fight parameters, talents, buffs, boss debuffs, gear and rotation.

The main sheet displays all the fight parameters and the results of the calculation. It allows the calculation of one set of parameters at once, which requires the users to note down the results of each calculation to compare different sets of parameters. However, since the calculation runs faster than a simulation, it is fairly user-friendly to compare different sets of parameters, but the accuracy of the results is not very high, since some combat mechanics are hard to model in an Excel spreadsheet.

There is no documentation for the implemented mechanics but there is a changelog.


DPSWarrior_Classic.xlsx - Excel

Item Setup

Slot	Name	Weapon Type	Crit	Hit	Str	Stam	Agi	AP	Dodge	Parry	Def	AC	MH Skill	OH Skill	Proc DPS
Head	Lionheart Helm	0	2	2	18	0	0	0	0	0	0	565	0	0	
Enchant	*Libram of Voracity(+8 Str)*				8	0	0								
Neck	Onyxia Tooth Pendant		1	1	0	9	12	0	0	0	0	0			
Shoulders	R10 Plate Shoulders		1	0	17	18	0	0	0	0	0	552			
Enchant	*None*		0		0	0	0	0							
Back	Cape of the Black Baron		0	0	0	0	15	20	0	0	0	45			
Enchant	*Lesser Agility*						3								
Chest	Savage Gladiator Chain		2	0	13	13	14	0	0	0	0	369			
Enchant	*Greater Stats*				4	4	4								
Wrist	Berserker Bracers		0	0	19	11	8	0	0	0	0	323			
Enchant	*Superior Strength*				9	0									
Hands	Flameguard Gauntlets	0	1	0	0	13	0	54	0	0	0	488	0	0	
Enchant	*Greater Strength*				7	0	0								
Waist	Onslaught Girdle	0	1	1	31	11	0	0	0	0	0	494	0	0	
Legs	R8 Plate Leggings		2	0	12	17	0	0	0	0	0	618			
Enchant	*Libram of Voracity(+8 Str)*				8	0	0								
Feet	Bloodmail Boots		0	1	9	10	9	0	0	0	0	247			
Enchant	*Minor Speed*					0	0								
Ring	Don Julio's Band		1	1	0	11	0	16	0	0	0	0			
Ring	Quick Strike Ring		1	0	5	8	0	30	0	0	0	0			
Trinket	Blackhand's Breadth		2	0	0	0	0	0	0	0	0	0			0
Trinket	Hand of Justice		0	0	0	0	0	20	0	0	0	0			14,3019454
Main Hand	Vis'kag the Bloodletter	Sword	0	0	0	0	0	0	0	0	0	0	0	0	6,00689183
Enchant	*Crusader*				35,89		0								0
Off Hand	Brutality Blade	Sword	1	0	9	0	9	0	0	0	0	0	0	0	0
Enchant	*Crusader*				24,79		0								0
2H	Swords	0	0	0	0	0	0	0	0	0	0	0	0		0
Enchant	*Crusader*				0		0								0
Ranged	Striker's Mark		0	1	0	0	0	22	0	0	0	0			
Starting Race	Human	Varied			120	110	80	160					5	5	
Raw Stats	☐		0	0	0	0	0	0	0	0	0	0	0	0	
Total			15	7	512,3	369,6	231	362	0	0	0	3701	5	5	

Results

	DPS		MH	OH
Total	1075,1831	Crit Cap	41,2	55,46
Normal Rotation	1039,5706	Over Crit Cap	5,55	0
Execute Phase	1276,9874			

	TPS
Total	571,3652627
Normal Rotation	557,4704172
Execute Phase	650,1026925

Rotation

Abilities	DPR	DPS	RPS	CD	Using
Bloodthirst	53,81	256,9594919	4,775	6	✓
Mortal Strike	0	0	0	6	☐
Slam	0	0	0	4	☐
Heroic Strike	13,21	408,7815983	12,504	2,614470201	✓
Cleave	0	0	0	3,297464692	☐
Whirlwind	32,55	67,8043946	2,0833	12	✓
Overpower	0	0	0	30	☐
Hamstring	0	0	0	4	☐
Execute Spam	39,69	757,9018394	19,097	2,037196065	✓

Attributes

	MH	OH
Crit Chance	46,75	46,75
Hit Chance	7	7
Weapon Skill	305	305
Attack Power	2293,5	
Haste	25,97%	
Health	5205	
Armor	4547	
Spell Crit	15	
Armor Penetration	0	

About | Changelog | DPS | Calcs | Proc Rates | Head | Neck | Shoulder | Back | Chest | Wrist | Hands | Waist | Legs | Feet | Rings | Trinkets | One-handed Weapons | Two-ha ...

Figure 6. DPSWarrior_Classic.

The main drawback of this tool is that it cannot be as accurate as a simulator, because it is very difficult to model the combat system of this game using a spreadsheet. I like that it provides a clear overview of the character’s configuration.

4.6 Conclusions

Table 1. Comparison of existing tools

Tool	WarriorSim	FurySim	ClassicSim	Wow fury sim	DPSWar-rior_Classic
Talent selection	+	-	+	+	+
Buff selection	+	+	+	+	+
Debuff selection	-	+	+	-	+
Rotation selection	+	+	+	+	+
Performance[4]	2.95s	53.05s	7.5s	13.46s	-
Documentation	-	+	+	+	-
User-friendly[5]	4/5	3/5	4/5	3/5	3/5
Comparison of different sets	-	-	-	+	-
Comparison of one item slot	+	-	-	-	-
Saving different sets	-	-	+	-	-
Detailed results breakdown	+	+	+	+	-
Multi-target simulation	-	-	-	+	-
Combat logging	-	-	-	+	-
Unit tests	-	-	+	+	-

[4] The performance was measured in seconds by running a simulation with 100 000 iterations on each tool. See Appendix IV for more details about the experiment.

[5] This is a subjective rating based on personal preferece.

Although there exist multiple tools for modelling the Warrior class, none of them are perfect and do not have all features a user would require, as is illustrated in Table 1. Based on the analysis in this chapter, the following goals are formed for the created simulator:

1. User-friendly GUI for configuring talents, buffs, debuffs, rotation, gear, fight parameters and simulator settings.
2. Configuration must be saved between sessions.
3. The simulator must provide an average DPS metric for the simulated fight.
4. The simulator must support multi-target fights.
5. The simulator must have comparable performance to other tools (100 000 iterations in less than 10 seconds).
6. The combat mechanics must be verified using unit tests.
7. The simulator must have an option to generate combat logs for debugging purposes.

Meeting these requirements will be evaluated in section 7.1.

5. Implementation

This chapter describes the planning process for the development and the implementation of the application.

5.1 Planning

5.1.1 Web application vs. desktop application

One of the concerns for this type of simulator is performance. The more iterations you are able to run in a simulation, the more accurate the average result is going to be. On the other hand, you also can't run too many iterations, because for better user experience, you want the simulation to run fast. The more optimized the code, the faster the simulations run, leading to more accurate results while not damaging user experience by being too slow. Desktop applications generally provide better performance than web applications and are thus better suited for a simulator.

Deploying the application would be easier in a web application, since users would always use the newest version, while on desktop applications they would need to update their software to get the benefits of the newest version. However, with a desktop application, one does not have to worry about hosting a web application.

Since performance is an important concern for a simulator and I'm not experienced in writing web applications, I decided to create the simulator as a desktop application.

5.1.2 Language choice

When choosing the programming language for this project, I had three options: Java, C++ or Python. The main concerns in this choice were experience with the language, performance and ease of development.

In high school I learned C++ as my first programming language, however I have not used it much after high school. In the first year of university, I learned Python and I have also used it for a short time while working as a software developer. However, I am most comfortable with using Java, one of the factors being that most of the programming courses taught in university were based on Java. I also prefer Java in general, since it is easier to develop due to not having to deal with memory management like in C++.

Performance wise, both C++ and Java win out over Python, because Python is an interpreted language. As such, it was not suitable for this project, since simulation speed is an important factor. While C++ is regarded as one of the best languages for high performance computing, Java is not so far behind. With the introduction of just-in-time compilation and the optimizations the java compiler does under the hood, nowadays Java is competitive with C++ in terms of performance [7].

Java also makes it easier to write code, because you don't have to deal with memory management, since Java uses automatic garbage collection. I also like that Java is a more verbose language, because this makes it easier to work with large codebases due to the code being more understandable when revisiting it.

Since I'm more experienced with Java and would not have to spend time relearning the basics, and with performance being comparable to C++, I chose Java as the programming language for this project.

5.2 Project structure

The project is divided into two directories: *src/main* and *src/test*. The *main* directory contains all the application code and the *test* directory contains tests. The *main* directory itself is divided into two directories: *java* and *resources*. The *java* directory contains all java classes while the *resources* directory contains data, image files, CSS and FXML files.

Related java classes and resource files are also separated into packages:

- The *sim.data* package contains JSON files describing the data and java classes which provide access to the data.
- The *sim.engine* package contains code related to game mechanics and simulation.
- The *sim.main* package contains main controller, which controls all the components of the application.
- The *sim.items*, *sim.rotation*, *sim.settings*, *sim.stats*, *sim.talents* packages contain the code and resources related to the respective GUI elements.

5.3 Data

A complex game like Classic World of Warcraft uses a lot of data: items, spells, talents, enchants all have names, icons, modify different stats and have various effects associated with them. For a simulator, not all of this data is needed, so I included only the necessary data needed for simulating the Warrior class.

All the data for the project is stored in JSON files under */src/main/sim/data* directory. I chose to use JSON, because all the data used is static and does not change during runtime, so there was no need to use an actual database. When the application is started, I use the Gson library to load all the data into respective Java objects, which can be accessed by the application.

5.4 GUI

The graphical user interface is created using JavaFX and by following the model-viewer-controller design pattern. Each component of the user interface has a controller, which manipulates the model and updates the user interface when users configure their character setups.

In my application, the model consists of Settings and CharacterSetup classes. The Settings class stores simulation configuration and CharacterSetup, which stores all the character specific configuration. When users modify their character in the application, the controller classes update the configuration in the model and display the necessary information to the user.

The views are defined using CSS and FXML files, but in some cases when there are many similar UI components, they are created programmatically. For example, for the item selection view, the ItemsController class contains the ListView that displays the list of items to be selected for a specific item slot. However, since there are 17 different item slots, the ItemSlot objects are created programmatically.

5.5 Simulation engine

The simulation engine is started when the user clicks the Simulate button. A new simulation object is created, which receives the Settings and CharacterSetup objects that the user has configured using the GUI.

5.5.1 Simulation.java

First, the engine splits the amount of iterations between each processor core the user's system has available and then creates new threads which simulate the fight the given amount of times. The fight results on each thread are merged together by averaging them after every iteration and when all the threads have finished working, the results of each thread are also merged together to provide a single average output. The progress bar is updated as the simulation runs.

5.5.2 Warrior.java

This class represents the state of the Warrior character during a combat encounter. During the fight, various events can modify the state of the Warrior, such as the stats. Combat calculations depend on the state of the Warrior and the required values are retrieved from this class.

5.5.3 Fight.java

The Fight class represents a single iteration of the combat encounter. It contains the event queue and event handler for the combat events. Each event is assigned a time and the event queue is implemented as a PriorityQueue<Event> in which events are ordered based on their assigned time.

When a fight is started, the Fight.run() method is called and the initial events are added to the event queue. The events are then processed in a while loop, which removes the next event from the event queue, sets the current time to the event's time and sends the event to the event handler. The event handler handles the event and adds new events to the event queue. The while loop breaks and the fight ends when the current time exceeds the specified fight duration in the settings. This is my implementation of discrete-event simulation described in chapter 2.3.1.

5.5.4 Abilities

Abilities are implemented using the abstract Ability class, which extends the Event class. This way, they can be added to event queue and processed by the event handler a similar way, while the behaviour and effects of each ability are defined in their respective classes, which extend the Ability class.

5.6 External libraries

- The Gson library is used to read the JSON data files into objects.
- The Log4j2 library is used generate log files while the application runs.
- The JUnit library is used to write tests for the project.
- The JFoenix library is a JavaFX material design library, which is used for some UI components.

6. Build, deployment and installation

This chapter describes the build and deployment process for the application and the various tools used for this.

6.1 Gradle

Since the application uses many different libraries, I needed to use a build tool to make the dependency management and build process as smooth as possible.

I use Gradle to take care of all the build related tasks, such as setting up external libraries, testing, running and deploying the application. Gradle also makes it easy to set up the project for other developers, since the Gradle Wrapper ensures that the project is standardized on a given Gradle version.

6.2 JLink and jpackage

I use a Gradle plugin called Badass JLink Plugin which can create a custom runtime image containing only the necessary dependencies for my application and use that runtime image to package native installers for Windows, Linux and macOS using the jpackage tool. This makes the installation process easy and familiar to end users, since they do not have to deal with JAR files and can run the application as any other application.

6.3 GitHub actions

One of the issues with the jpackage tool is that there is no support for cross-platform compilation. However, this can easily be solved using GitHub actions, which can be used to automate the software development workflow and run the jpackage command on each of the operating systems needed on GitHub hosted runners. Each time a push or pull request is made to the GitHub repository, my GitHub action builds and tests the application and then creates the installers using jpackage.

6.4 Installation

Users can install the application on Windows, Linux or macOS by using native installers available at https://github.com/Viido/WarriorSim-Classic/releases.

After the installation is completed, the application can be started by using the application launcher. For Windows, the launcher is located in the installation directory and named

"WarriorSim-Classic.exe". For Linux, the launcher is called "warriorsim-classic" and is located in the installation directory's */bin/* directory. For macOS, the launcher called "WarriorSim-Classic" is located under */Contents/MacOS/* in the installation directory.

6.4.1 Using a local repository

The project can be cloned locally and ran using Gradle. JDK version 11+ is needed, however JDK version 16 is not supported yet.

Use the following commands to clone the repository and run the application:

```
git clone https://github.com/Viido/WarriorSim-Classic.git

cd WarriorSim-Classic

./gradlew run
```

For Windows, use gradlew.bat instead:

```
gradlew.bat run
```

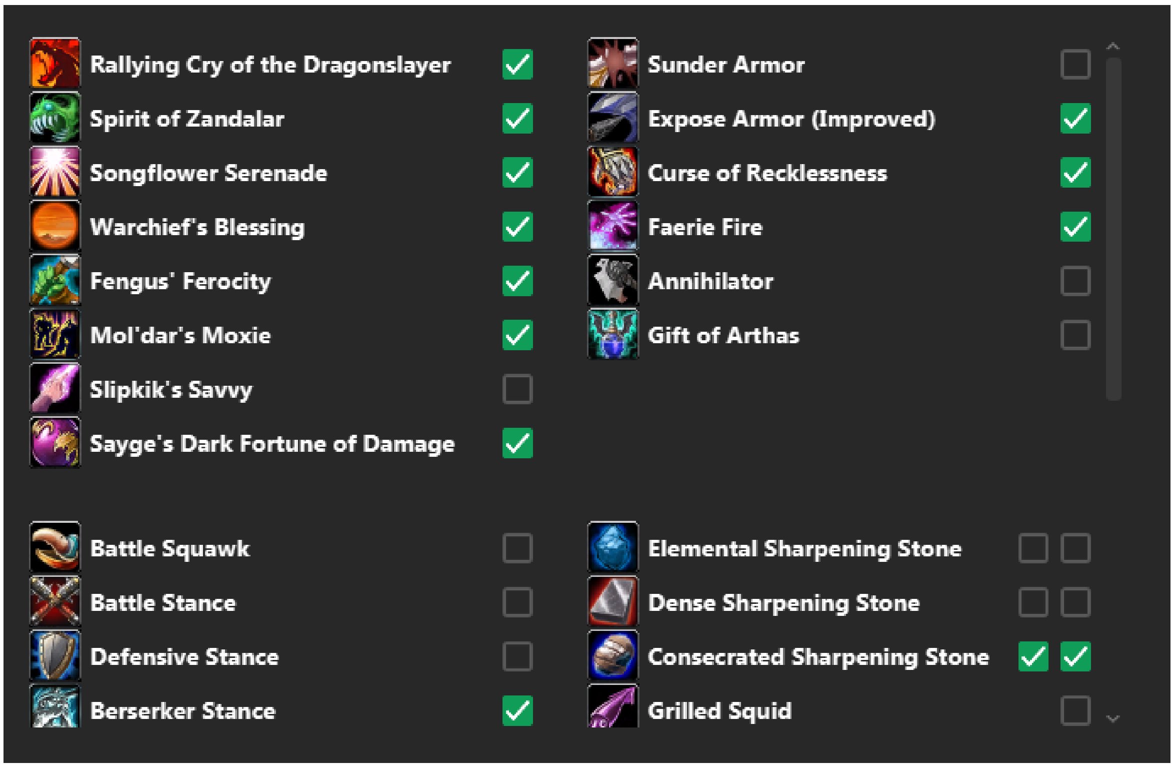
7. Evaluation

This chapter evaluates the created application based on the goals set in section 4.6 and describes the challenges faced during the development process. It also talks about the future plans regarding the application.

7.1 Goals

1. User-friendly GUI for configuring talents, buffs, debuffs, rotation, gear, fight parameters and simulator settings.

   Buffs and debuffs (Figure 7) can be selected using checkboxes:



Rallying Cry of the Dragonslayer	✓	Sunder Armor	☐
Spirit of Zandalar	✓	Expose Armor (Improved)	✓
Songflower Serenade	✓	Curse of Recklessness	✓
Warchief's Blessing	✓	Faerie Fire	✓
Fengus' Ferocity	✓	Annihilator	☐
Mol'dar's Moxie	✓	Gift of Arthas	☐
Slipkik's Savvy	☐		
Sayge's Dark Fortune of Damage	✓		
Battle Squawk	☐	Elemental Sharpening Stone	☐ ☐
Battle Stance	☐	Dense Sharpening Stone	☐ ☐
Defensive Stance	☐	Consecrated Sharpening Stone	✓ ✓
Berserker Stance	✓	Grilled Squid	☐

Figure 7. Buffs and debuffs selection.

Talents are configurable using a talent calculator (Figure 8) which behaves the same way as it does in-game and on online tools:

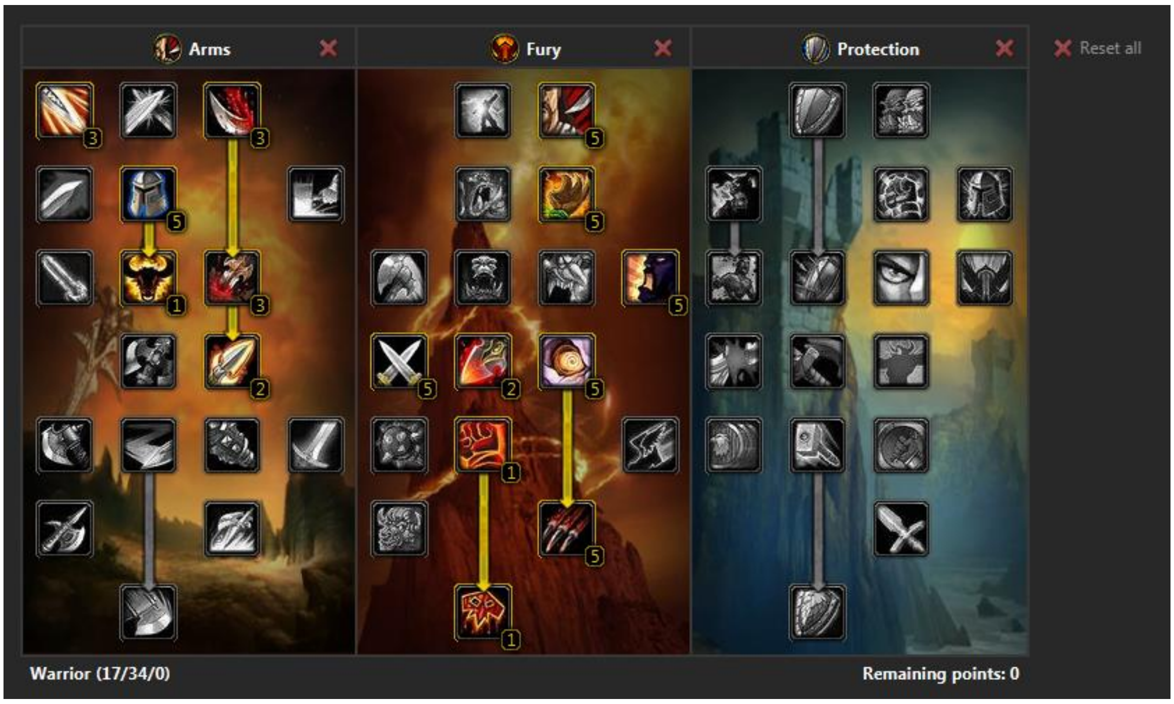
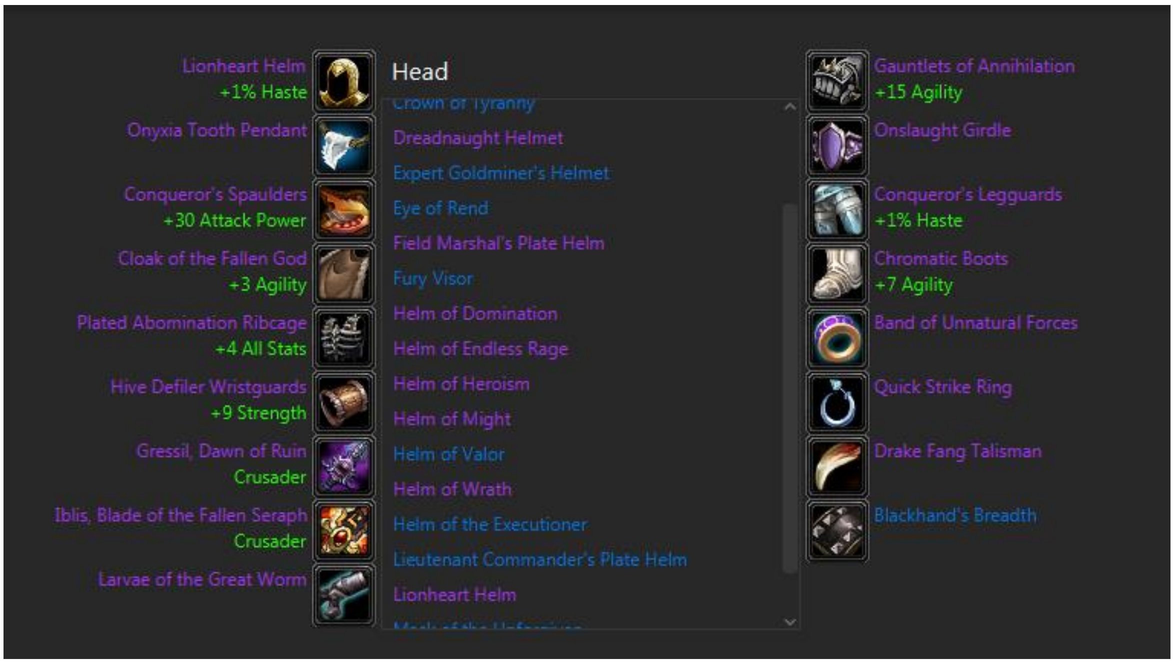


Figure 8. Talent calculator.

The item selection (Figure 9) is familiar to users from in-game:



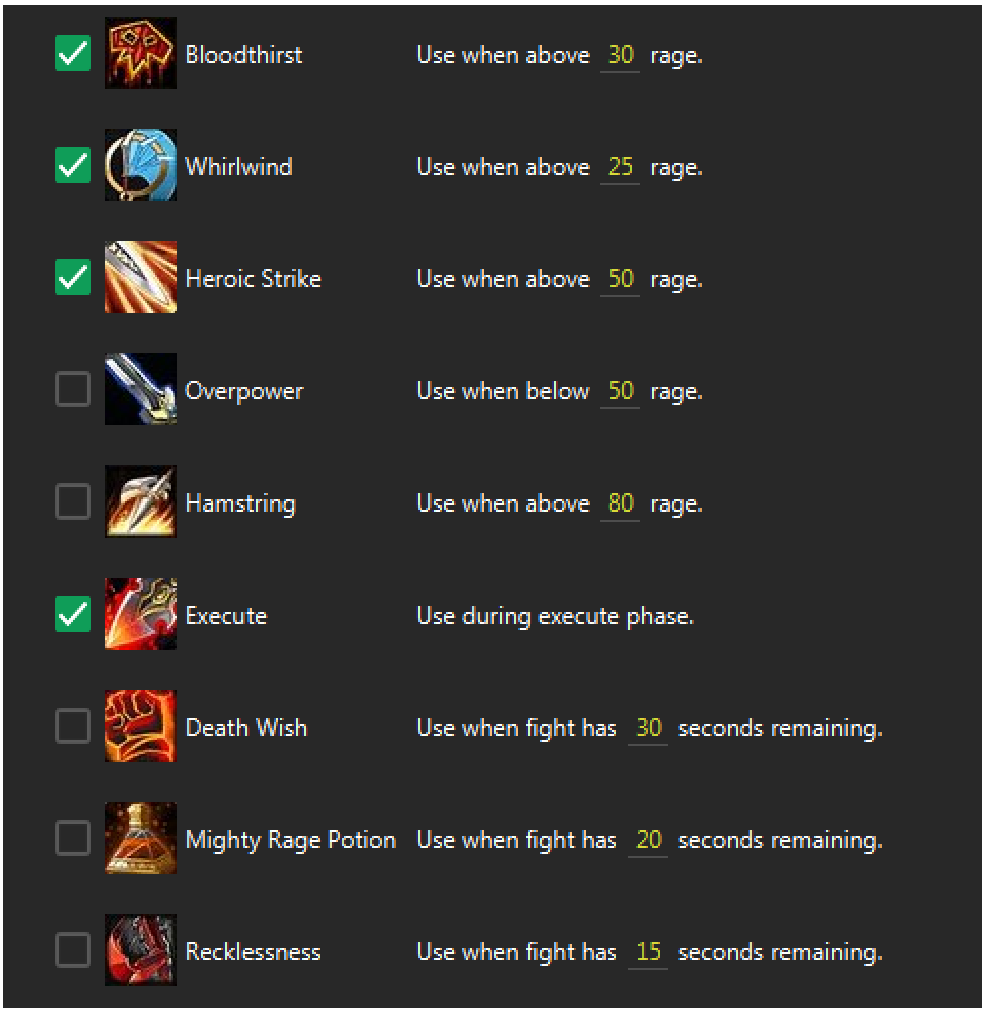
Lionheart Helm
+1% Haste
Onyxia Tooth Pendant
Conqueror's Spaulders
+30 Attack Power
Cloak of the Fallen God
+3 Agility
Plated Abomination Ribcage
+4 All Stats
Hive Defiler Wristguards
+9 Strength
Gressil, Dawn of Ruin
Crusader
Iblis, Blade of the Fallen Seraph
Crusader
Larvae of the Great Worm

Head
Crown of Tyranny
Dreadnaught Helmet
Expert Goldminer's Helmet
Eye of Rend
Field Marshal's Plate Helm
Fury Visor
Helm of Domination
Helm of Endless Rage
Helm of Heroism
Helm of Might
Helm of Valor
Helm of Wrath
Helm of the Executioner
Lieutenant Commander's Plate Helm
Lionheart Helm

Gauntlets of Annihilation
+15 Agility
Onslaught Girdle
Conqueror's Legguards
+1% Haste
Chromatic Boots
+7 Agility
Band of Unnatural Forces
Quick Strike Ring
Drake Fang Talisman
Blackhand's Breadth

Figure 9. Item selection.

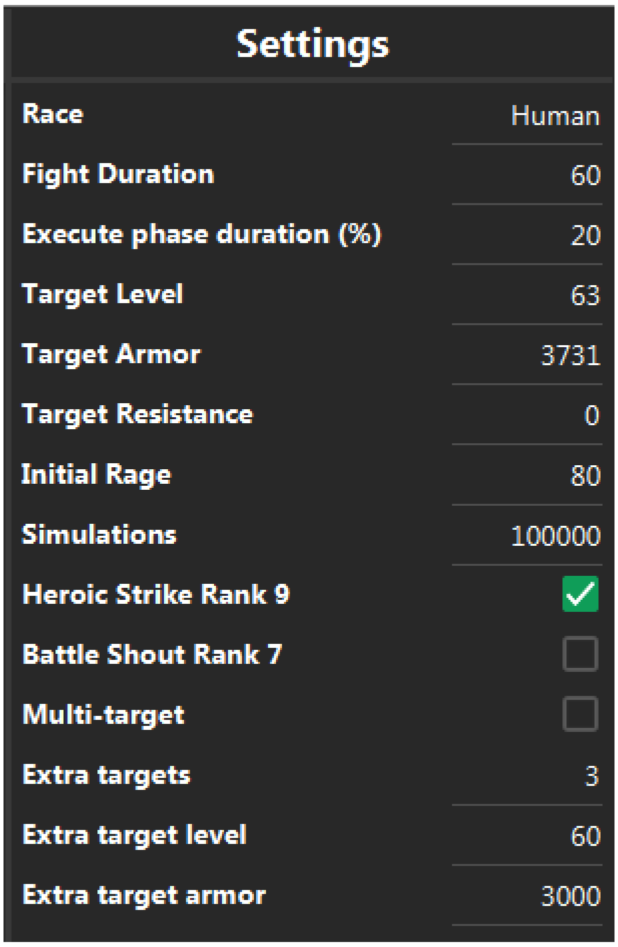
Rotation (Figure 10) can be configured by enabling abilities with checkboxes and inputting thresholds, which determine in what situation the abilities are used:



✓	Bloodthirst	Use when above 30 rage.
☐	Whirlwind	Use when above 25 rage.
✓	Heroic Strike	Use when above 50 rage.
☐	Overpower	Use when below 50 rage.
☐	Hamstring	Use when above 80 rage.
✓	Execute	Use during execute phase.
☐	Death Wish	Use when fight has 30 seconds remaining.
☐	Mighty Rage Potion	Use when fight has 20 seconds remaining.
☐	Recklessness	Use when fight has 15 seconds remaining.

Figure 10. Rotation configuration.

Fight parameters and simulator settings (Figure 11) can be configured using text fields and checkboxes:



Settings	
Race	Human
Fight Duration	60
Execute phase duration (%)	20
Target Level	63
Target Armor	3731
Target Resistance	0
Initial Rage	80
Simulations	100000
Heroic Strike Rank 9	✓
Battle Shout Rank 7	☐
Multi-target	☐
Extra targets	3
Extra target level	60
Extra target armor	3000

Figure 11. Settings and fight parameters configuration.

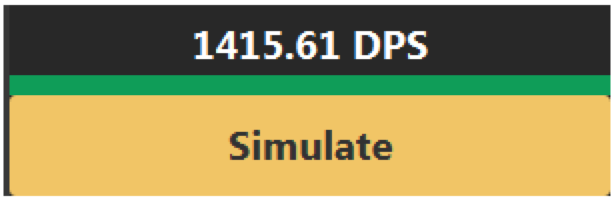
The application provides a clear overview of the configuration options, which are intuitive to use.

2. Configuration must be saved between sessions.

   When a user exits the application, the configuration is automatically saved to a file called "settings.ser". The configuration is automatically loaded the next time the user starts the application.

3. The simulator must provide an average DPS metric for the simulated fight.

   After the simulate button is clicked and the simulation has finished running, the average DPS value is updated and visible to the users (Figure 12).

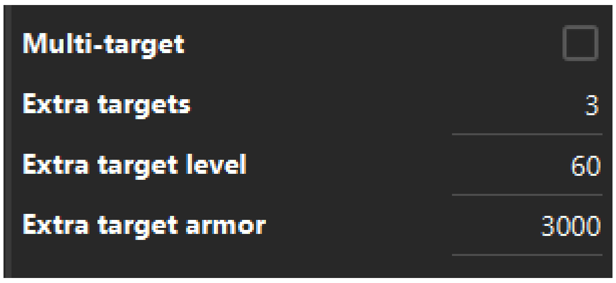


1415.61 DPS

Figure 12. Average DPS value.

4. The simulator must support multi-target fights.

The simulator has options for multi-target fights (Figure 13). It allows the user configure the amount of extra targets and the target's armor and level.

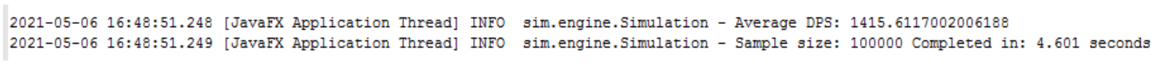


Multi-target ☐
Extra targets 3
Extra target level 60
Extra target armor 3000

Figure 13. Multi-target fight configuration.

5. The simulator must have comparable performance to other tools (100 000 iterations in less than 10 seconds).

   The simulator uses multi-threading and event-driven time progression to achieve good performance. The simulation generates a "combat.log" file, which displays detailed results of the simulation, amount of iterations and the amount of time the simulation took (Figure 14). The same performance experiment (see Appendix IV) was carried out as for the other tools and the simulator completed 100 000 iterations in 4.601 seconds.



```
2021-05-06 16:48:51.248 [JavaFX Application Thread] INFO  sim.engine.Simulation - Average DPS: 1415.6117002006188
2021-05-06 16:48:51.249 [JavaFX Application Thread] INFO  sim.engine.Simulation - Sample size: 100000 Completed in: 4.601 seconds
```

Figure 14. The time it took to complete the simulation.

6. The combat mechanics must be verified using unit tests.

   The project supports adding unit tests using the JUnit 5 framework. Currently, there exist tests for basic mechanics. More tests will be added in the future.

7. The simulator must have an option to generate combat logs for debugging purposes.

   When the amount of iterations is set to 1, the application generates a detailed log of the combat events that occurred during the simulation. This is valuable information for debugging the combat simulation.

Based on this analysis, all the goals for the application were met.

7.2 Challenges

This was the first project of this scale for me. I quickly realized that I had to learn a lot on the fly while writing the project. I had little experience with developing front-end, so I had to learn how to use JavaFX and concepts such as MVC. A lot of time was spent on thinking about how to structure the project and how different components should interact with each other.

One of the issues I encountered was the JavaFX application thread freezing and lagging when the multi-threaded simulation was running. There seemed to be no options to limit the CPU usage of Java threads, so I ended up just stopping the threads for a small duration in intervals, which allowed more resources to be available for the JavaFX application thread. However, this was not a perfect solution, since the simulation would run slightly slower, as the threads were being stopped.

Another challenge was the build and deployment process. I did a lot of research on what the recommended options were and found that you could make a modular java project, in which you could define in “module-info.java” which dependencies your project uses. Then you could build a custom run-time image of your application containing only the required dependencies using jlink and create installers using that run-time image with jpackage. Luckily, I found a Gradle plugin, which took care of most of that. However, I encountered issues with the JFoenix library not being able to read the required JavaFX packages. I solved this by adding a bunch of JVM arguments which export the required JavaFX packages to JFoenix.

In retrospect, I could have kept the scope of the project smaller, because I went well over the 9 ECTS of the bachelor’s thesis in terms of hours spent working on the project. This would have been better for completing the bachelor’s thesis in time. However, I chose to extend the deadline and worked on delivering a more complete project. I estimate I spent over 400 hours working on the practical part of the thesis.

I used the cloc[6] tool to count the lines of code in the repository. As of the latest commit[7], there are 5843 lines of Java, 323 lines of CSS and 10953 lines of JSON data. GitHub history shows a total of 50 commits, with 40594 additions and 20020 deletions.

7.3 Personal development

Writing a project of this scale greatly improved my skills as a software developer. I had to overcome many challenges and learned not only how to write better code, but also how to make architectural decisions and how to build and deploy working applications for the end user.

[6] https://github.com/AlDanial/cloc

[7] https://github.com/Viido/WarriorSim-Classic/commit/023c72c9ce41cdf8d92756f921b3ec23f8f63469

7.4 User feedback

I shared the project with another developer I know in the Classic community. He managed to get the application running on his system, however there were issues with images loading on some parts of the UI. He filed a bug report[8] on the project's GitHub repository and I was able to identify the issue and fix it.

Because the project is open-source, I hope to receive more feedback like this and work together with the community to improve the project.

7.5 Future

I plan to continue the development of this simulator after the thesis. There are features that I want to add such as support for tanking, multiple sets, detailed ability breakdowns and stat weight calculation. There is also room for optimizations to be made so that the simulation runs faster. I also want to work closely with the Classic Warrior community to improve the tool. I have a good platform to do it, since there are over 60 000 members in the Classic Warrior discord, who are potential users for the tool.

When I have finished working on the new features, I plan on adding support for the next Classic World of Warcraft expansion, The Burning Crusade. For the expansion new items, talents and abilities need to be added, and combat formulas need to be updated.

There is also a new Java VM and JDK called GraalVM[9] that I want to look into using in the future. It has an option for ahead-of-time compilation using its native image technology. This would make the application start up faster and also make the simulation run faster. Currently, you can see the effects of just-in-time compilation: the first time the simulation is run, it takes more time. The next time, the compiler has made optimizations and the simulation runs faster. With ahead-of-time-compilation, it would run fast every time.

[8] https://github.com/Viido/WarriorSim-Classic/issues/1

[9] https://www.graalvm.org/

8. Conclusion

The goal of this thesis was to create a computer simulator for the Classic World of Warcraft Warrior class. A desktop application written in JavaFX was created for this purpose. The thesis described the development process in-depth, analysed the decisions made and described the project structure, components and tools used. The application with installation and usage instructions is available at https://github.com/Viido/WarriorSim-Classic.

The thesis gave an overview of the background of the game, motivation for creating such a tool and also described the theory behind discrete-event simulation, which is used in the simulation engine of this application.

The thesis described the different combat mechanics that exist in the game, which needed to be implemented in the simulator in order to replicate the combat system of the game. These combat mechanics were implemented successfully in the simulator.

An analysis of the existing tools was provided, identifying missing features which would be needed in a complete tool. Based on this analysis, the goals for this application were created, which were met during the development and evaluated in the thesis.

In the future, the project will be continue to be developed by the author of this thesis. There are plans for new features, optimisations to the engine and support for the next Classic expansion. The author will also work together with the community in order to improve the tool.

9. References

[1] Blizzard Press Release. October 7, 2010. https://web.archive.org/web/20111113041947/http://us.blizzard.com/en-us/company/press/pressreleases.html?id=2847881 (Retrieved 01.04.2020)

[2] Winsberg, Eric. (2019). Computer Simulations in Science. The Stanford Encyclopedia of Philosophy (Winter 2019 Edition). https://plato.stanford.edu/archives/win2019/entries/simulations-science

[3] Matloff, N. (2008). Introduction to Discrete-Event Simulation and the SimPy Language. University of California, Davis. Dept. of Computer Science. http://heather.cs.ucdavis.edu/~matloff/156/PLN/DESimIntro.pdf (Retrieved 31.03.2020)

[4] Allen, M., Spencer, A., Gibson, A., *et al.* (2015). Right cot, right place, right time: improving the design and organisation of neonatal care networks – a computer simulation study. https://www.ncbi.nlm.nih.gov/books/NBK293948/

[5] Neryssa. (2019). Overview of Stats on Classic WoW – What Each Stat Does. https://classic.wowhead.com/guides/classic-wow-stats-and-attributes-overview (Retrieved 07.05.2021)

[6] TheKrugLife. (2020). Classic Fury Warrior Abilities & Rotation Guide - WoW Classic 1.13 https://classic.wowhead.com/guides/fury-warrior-dps-rotation-abilities-classic-wow (Retrieved 07.05.2021)

[7] Pancake, Cherri M. Lengauer, Christian. (2001). High-performance Java: Introduction. https://dl.acm.org/doi/fullHtml/10.1145/383845.383866?casa_token=cpYvMszpPdcAAAAA:etL-_v2HQj-upOBGGzzifQNoejsP3t6rcxg8-BAy07D2yJLE7WX9oNB3PT1Y3D9XfHHFcG5sYAxZ8A#body-1

Appendix

I. Glossary

1. Auto-attack – Attacks performed automatically by the character.
2. Battleground – Instanced areas in World of Warcraft intended for player vs. player combat.
3. Buffs – Beneficial effects placed on the player character, which enhance their stats.
4. Class – A character's class is used to distinguish characters based on their abilities.
5. Character – Fictional character in the game, who is controlled directly by the player.
6. Cooldown – The time during which an ability cannot be used. During this time, the ability is on cooldown.
7. Crit – Short for critical hit. A critical hit enhances an attack, causing it to deal increased damage.
8. Debuffs – Opposite effect of buffs.
9. DPS – Damage per second. This is a metric used to describe the damage done by a character. Damage dealing characters are also often referred to as DPS.
10. Dual-wielding – The act of wielding a weapon in both hands.
11. Guild – Players can form in-game communities in order to work together. These communities are called guilds.
12. Level – A number that represents a character's overall skill and experience.
13. Loot – Refers to items looted from monsters.
14. Proc - Proc is an acronym for "programmed random occurrence". It is a common term used in games for effects that have a random chance to occur.
15. PvE – Short for player vs. environment. Refers to content, in which players combat non-player characters or monsters.
16. PvP – Short for player vs. player. Refers to content, in which players combat other players.
17. Raid – In World of Warcraft a raid is a group of more than 5 players. It also refers to instances areas intended for raid groups.
18. Rotation – The order of priority in which player's use their character's abilities.
19. Stats – Short for statistics. Refers to the attributes which describe a character's characteristics.
20. Theorycrafting – Analysis of game mechanics in order to optimize gameplay.

II. Existing Tools

1. WarriorSim

   GitHub: https://github.com/GuybrushGit/WarriorSim

   Web application: https://guybrushgit.github.io/WarriorSim/

2. FurySim

   GitHub: https://github.com/rafaelpimpa/furysim

   Web application: https://furysim.github.io/

3. ClassicSim

   GitHub: https://github.com/timhul/ClassicSim

   Installation instructions are included in the GitHub repository.

4. Wow fury sim

   GitHub: https://github.com/albinoyoda/wow_classic_simulator

   Web application: https://wow-fury-sim.herokuapp.com/

5. DPSWarrior_Classic

   Download link:

   https://drive.google.com/file/d/1Er1CHvFncy_zNLICylZCV0HeDyMTvYxa/view?usp=sharing

III. Useful Links

1. Magey’s classic-warrior GitHub wiki
   https://github.com/magey/classic-warrior/wiki
2. Warrior Talent Calculator
   https://classic.wowhead.com/talent-calc/warrior
3. Buffs and Consumables
   https://classic.wowhead.com/guides/classic-world-buff-consumables

IV. Parameters for the Simulator Performance Experiment

All other programs were closed during the experiment and only one tool was used at a time. The time was measured using a stopwatch, which was started when the simulation button was clicked, and stopped when the simulation ended.

System specifications:

OS: Windows 7

CPU: Intel Core i5-4670k @ 3.40 GHz

GPU: AMD Radeon R9 270X

RAM: 8 GB

Character configuration:

Similar gear, buffs and fight parameters were configured for each tool. Small differences in these configurations did not affect the performance of the tools, so the exact configurations are not provided.

For the rotation, Bloodthirst and Whirlwind were used on cooldown and Heroic Strikes were used when above 50 rage.

Simulation parameters:

Fight duration: 60 seconds

Iterations: 100 000

V. License