D&D 5E Mass Combat Simulation

Aidan Ohnemus
Western Oregon University
D&D 5E Mass Combat Simulation

By
Aidan P. Ohnemus

An Honors Thesis Submitted in Partial Fulfillment of the
Requirements for Graduation from the
Western Oregon University Honors Program

Prof. Shannon,
Thesis Advisor

Dr. Gavin Keulks,
Honors Program Director

June 2019
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>3</td>
</tr>
<tr>
<td>Acknowledgements</td>
<td>4</td>
</tr>
<tr>
<td>Introduction and Inspiration</td>
<td>5</td>
</tr>
<tr>
<td>Implementation</td>
<td>17</td>
</tr>
<tr>
<td>Decisions</td>
<td>25</td>
</tr>
<tr>
<td>Development Process</td>
<td>28</td>
</tr>
<tr>
<td>Conclusion</td>
<td>34</td>
</tr>
<tr>
<td>Project Code</td>
<td>36</td>
</tr>
<tr>
<td>Bibliography</td>
<td>37</td>
</tr>
</tbody>
</table>
Abstract

Dungeons and Dragons 5th edition, as well as many tabletop RPGs in general, struggles with very large scale combat encounters. Mechanics are either changed to make it more suited to the increased scale, affecting the game balance in the process, or an entirely different game is used to represent the combat. This paper outlines the process undertaken to create a program that simulates combat encounters in 5th Edition D&D at a scale normally unfeasible without changing the game’s mechanics. I compare my program and its goals with the solutions that come closest to it, including alternate rulesets, virtual tabletops, and a balance testing program.
Acknowledgements

I would like to thank my instructors for teaching me the skills necessary to make this project in the first place. Information Systems instructors Chris Brooks and Ted Beers helped with their lessons on work management, agile and scrum. Despite not actually using agile or any formal system to organize my work, the insights into work management have been beneficial. I’m grateful for what Becka Morgan taught me about C++ in my computer science courses, as I needed them to learn a language to the depth needed to actually develop a program like this. I’d like to thank my friends and family for their support, and for being understanding when I had to cancel plans in order to put time into this project. I would especially like to thank my friend RJ for trying to help me get my work schedule together as deadlines started approaching. Many things have come up for me during the creation of this program, and if I didn’t have these people in my life I don’t think this program or the thesis would be where it is now.
Introduction and Inspiration

I’ve been playing Dungeons and Dragons (D&D) and other tabletop role playing games since I started attending classes here at Western. I don’t remember when or where it came up, but I learned very early on that army-scale combat is one thing players will try to avoid. No official system even exists in the most current edition of D&D, which is 5th edition. Doing combat at this scale with the normal rules, which are intended for encounters where 30 people is quite a lot, could require rolling thousands of dice. Some third-party or official, but experimental rule sets simplify the mechanics by grouping up combatants and changing rules, but this means you’re no longer playing the same kind of game. Some people dispense with D&D entirely, playing another game like Risk, or some modified version of it.

I’ve dealt with one real use of a mass combat system, and it was in 5E. We didn’t use any systems I mention in this document. It was a combination of a system the Dungeon Master (DM), the person who runs the game and controls non player characters, found online with some work of his own. Whether or not the battle was won was determined by the earning of points, which were gained by completing certain objectives, eliminating major enemy units, and so on. Interestingly, the point system was completely useless to our party, because we managed to eliminate all three of the enemy army’s leaders in one massive, chaotic attack. What made that game work was that each of our individual characters felt useful in their own ways, to a degree that even surprised the DM. Some players focused on the elimination of key targets rather
than raw attrition, whether it’s through elimination of siege weapons, destruction of
supplies, or assassination of key figures. Others had characters suited to area control,
zoning off entire regions through their sheer control of the battlefield.

What’s important is that the DM handled much of the tedious math and work
with groups of enemies by abstracting the mechanics significantly. Often times,
creatures or groups of creatures would roll to see what percentage of the enemies they
eliminate, modifying that number based on how well they attack in a normal sense. It
varied wildly partially because our tactics were a bit bizarre, but also because these rolls
and actions were the DMs best attempt at allowing the game to flow at any reasonable
pace, so they had to be made very different from the base game.

I bring this up to explain why mass combat is so important to me. That game was
immense fun, but it also took a significant amount of time to proceed through each
round of combat even with the changes mentioned, and the DM changed the length of
time that each round lasted in order to make it work better. Normally a round is 6
seconds, he extended it to around a minute, which drastically changed the way the
game is balanced. Spells with durations were more effective than they should be
normally, some abilities were stronger or weaker than they would be normally. It was
still fun, but it made me wonder what could have happened if the massive groups of
soldiers could have had their AI automated while all the major characters were
controlled manually. What would have happened if the siege had played out more like
normal D&D? I’m convinced that the tactics of that battle would have broken down due
to the way the game works. Normal scale combat encounters in D&D regularly last less
than 5 rounds, which is only 30 seconds. Combat is incredibly fast in this game, which is fitting because the characters you play are supernaturally gifted.

What happens when a single soldier is targeted by a massive group of archers because the enemy general realize that he is an adventurer? A fighter may be able to handle it, but a weak character like a wizard could suddenly find themselves on the receiving end of several hundred arrows, which is an advisable tactic considering that same wizard to blast a massive crowd of soldiers with one spell, especially if they are closely grouped together. There’s many things about D&D that renders traditional medieval style sieges almost useless once you have magic casters, yet they’re commonly used by many. With a program that can automate and simulate these battles, I can show how and why these things are the case. A system like this can allow investigation of how and why mass combat is different.

So then, what does this system need to do? Here’s a list of user stories that cover the functionality I set out to create.

1) Users can create units to fight other units
2) Users can remove units to manually change the battle, or as a part of the battle’s setup
3) Users can tell units to automatically fight, so that the user can manage more units than would otherwise be feasible
4) Users can include the basic stats and attributes from D&D 5th edition in units so that they can start to simulate D&D and test its mechanics
5) Users can include all stats and attributes from D&D 5th edition in units so that they can accurately simulate D&D
6) Users can change a unit’s statistics and attributes on the field so that they can apply modifiers the system doesn’t have options for
7) Users can give units the ability to cast spells to accurately simulate D&D 5th edition without manually applying complicated spells
8) Users can create, save and load spells from the program to give units
9) Users can access a pre-existing library of spells from D&D so that they don’t have to enter many, or any, themselves.
10) Users can run repeated simulations of encounters and track the results in order to test tactics, or to discover how D&D’s rules work at an unintended scale. 
11) Users can save and load units so that they can build a library of units rather than manually entering them. 
12) Users can access a pre-existing library of common units from D&D so that they don’t have to enter any creature statistics.

So all together, I want to be able to do everything I can in 5th Edition D&D while still following the rules, and in addition, for units to be able to automatically fight. The overall motivation is to make mass combat in 5th edition less tedious. There are a number of systems that already exist that cover a number of these requirements, as well as useful features of their own. There are three main ones you’ll hear about online, each with its own set of advantages.

The first utility many people turn to when it comes to D&D virtualization is Roll20 [4], a free online tool that is one of the most popular of its type. It contains map editing, a character sheet creation tool, built-in monsters including ones in official packs that come from partnership with Wizards of the Coast (The makers of D&D) themselves, player-side dynamic lighting, virtual dice rolling, and video/voice chat. Most of these features are quality of life tools, along with its main benefits of being a tabletop you’re able to use online, and you don’t have to buy physical miniatures. However, the library of creatures are of interest when comparing it to my project, as these creatures have all their statistics and actions built-in, and you can create them yourself if you have custom ones to add.

The key difference here is the lack of AI. In the spirit of a true tabletop, the dungeon master controls all monsters as they usually do, and players (generally) control
their own pieces. Damage rolls can be automated, but hit point deduction and the actual math is generally tracked by players and the DM still. Given that Roll20 is such a good virtualization, it’s a useful alternative to traditional table tops. One of its notable features is a true random dice roller that pulls its randomization from the quantum fluctuations of a beam of light. My program still uses the traditional pseudo-random numbers common to most programs with random outcomes, but for the most part, that’s not a problem. For D&D, pseudo-randomization is acceptable as much as there is a bias against it in the community. True random numbers are more important in security, as you can’t guess them by assuming the seed and algorithm a system might be using in order to break some security system or encryption. That’s not relevant to D&D. Still, it is a good selling point.

Roll20, while useful, is not comparable when it comes to what makes my system distinct. The program has a virtual tabletop grid, character tokens, action automation, health tracking, and so on, all features the Tabletop RPG focused programs like Roll20 have. They’re more primitive in my system, and some features are missing due to development issues, but the necessary pieces for automation are there, which is the next stage that said programs do not have.

Roll20 meets all the requirements I laid out for my project except for stories related to automation, #3 and #10. It supports spellcasting to a point, but it won’t run the math automatically. It has the following additional stories:

1. Users can create macros to automate movements, actions, and math so that they can prepare events or repeat complex tasks more easily
2. Users can access macros and scripts created by the community so that they can save time and enhance their game

3. Users can roll truly random numbers within the program so that they can play D&D fairly and easily

4. Users can access a D&D game online, so that people who can’t meet physically can easily play D&D together.

5. Users can access a wiki filled with information about D&D 5th edition, so that they can easily check rules, learn about the world, and run the game.

6. Users can access hex, standard grid, or isometric views so that DMs can choose a visual style that suits them and their players.

7. Users have access to a number of different graphics for characters and the environment, so that the game can be more visually appealing.

8. Users can create, edit and save maps so that they can prepare games for their players ahead of time

9. Users can recruit players from the internet to join their games, so that users interested in running a game that don’t have enough friends, can find some online.

Paid

1. Users can share the compendium of their game’s content with other players, so that only some people need to buy the content necessary for the game.

2. Users can purchase professionally made pre-existing maps and campaigns so that they can easily run a well-made game for their players.

3. Users can play the game using a tablet so that they can set up a large game grid in a table to play with their friends, or just for their convenience.

4. Users can add dynamic lighting and line of sight to their maps to enhance the experience for their players, and avoid spoiling content beyond their view.

The different graphics options, support for hex grids, and professional, pre-existing content are selling points that Roll20 has used to make their system shine. The recruitment listings are especially useful, as it takes Roll20 from a system for running games, to almost being a game itself. It’s similar to the online server hosting video games use. The paid features can be accessed by either the Plus or Pro accounts, 4.99 and 9.99 a month respectively. This does mean that, if you’re comparing my tool and Roll20 as free products, those aren’t relevant.
There is one program that does simulate D&D 5E combat. Even though it does so with a different motive and at a different scale, what it’s done is important. D&D Battle Simulator [2], a python script written by Matteo Ferla, a UK based PHD in Biochemistry working at the University of Oxford, runs by default, 1001 combat encounters between a set of creatures, with the number of rounds going even higher, around 5000 in some cases and likely far more. The average damage, average hits, average misses and average rounds, along with the numbers of victories, close-calls, and perfect rounds are displayed to the user. All combatants are assumed to be adjacent, and a set of assumptions about realistic combat behavior are made, rather than having every combatant act like a master strategist. Units attack the weakest enemy, and are organized into teams based on their moral alignment. Spellcasting is mostly removed as well, save for a few cantrips (at-will spells that can be used repeatedly, and are generally not incredibly powerful).

The intention of this program is to test D&D 5e’s GM guide table, which calculates how hard an encounter should be based on a chosen difficulty level, the number of players, and the Challenge Ratings of the monsters. The reason for this is because the creator believes that this table is poorly balanced, either creating encounters that are too easy or far too deadly. His tests have confirmed this, at least within his testing environment. This usage comes close to one of the uses I mentioned, where I would run the mass combat several hundred or thousand times to get the average odds of either side winning, as well as measurements of casualties. It could also
be useful to check for balancing issues, as Matteo has done, though his program tests for balance in small-scale battles.

One of the primary motivations behind building the system was to test D&D 5E and see how certain mechanics effect game balance and strategy. The core mechanic that makes me think there’s something to find in testing, is critical hits. One of the reasons why large crowds of small enemies are considered more deadly at times is because of critical hits, or blows that deal twice as many dice in damage. A common peasant that scores a critical hit with a standard sword is going to do more damage than you would expect, and when there’s 100s of them, they can theoretically take down a dragon.

To some degree Matteo has done this. His model starts to have problems in mass combat because the removal of a game grid becomes less and less easy to justify when you’re trying to claim that 3000 peasants are all within melee range of a dragon. Adding a grid to a simulation like this creates its own set of challenges. While Matteo’s system simulates some degree of intelligent decision making, mass-scale combat decision making gets into the realms of more advanced combat tactics and ability synergies, something neither his system nor mine intends to include. Also, I cannot get the level of precision that a PHD working at Oxford can get, so this system beats me in that respect.

To summarize:

1. Users can simulate combat between creatures in a grid-less space to determine the probability of victory or defeat for each group
2. Users can enter custom combatants into the system to have greater control over the creatures they’re simulating in combat.
3. Users can add creatures from a set of default combatants so that they can more easily create an encounter
That’s about all, unlike Roll20 it does not include many of the features I set out to make with my system. It includes #10, the story about repeated simulations, though with a different intent. For all its usefulness, Matteo’s utility is still for testing imbalance in combat encounters. It’s a tool for a DM who wants to hone their skills, to set up encounters so that players feel like they got a good challenge without being overwhelmed. It is a niche tool, and one whose benefits aren’t immediately obvious to people who don’t have a pretty firm grasp on being a DM in D&D. It does little to help those who want to solve the problem of the tedium of mass-scale combat. No programs have attempted to solve this problem from my research, however there are rulesets that have.

While we’re stepping away from programs, it would be strange to not talk about written rulesets for a game centered on playing with written rules, pen, and paper. Mike Mearls, current head of D&D’s R&D division and the designer of 5th edition, also writes small additional rulesets for 5th edition called “Unearthed Arcana”. These are experimental classes, feats, monsters, rules, and other things for D&D 5e that, if successful and balanced, may be added to later official books. They’re PDFs on the Wizards website, and the one of interest here is called “When Armies Clash” [3]. It’s an experimental ruleset for handling mass combat in 5th edition. Combatants are grouped into two categories: stands and solos. A stand is a group of 10 identical combatants, while a solo is a single person, generally a player, or a powerful or important non-player character. Each stand acts together as if it was one entity, cutting the complexity of managing any creatures in stands by 10. Stands are then grouped into units, which can
contain any number of stands, and units have two categories of their own that affect how they behave in combat, meant to create an abstract representation of combat tactics. Large scale combat isn’t just a lot of people in a combat encounter; movement, grouping and speed all change the way combat flows. Because of that, the way these rules change D&D’s combat are forgivable, but are where Mearls’ rules deviate from what I am trying to accomplish. That being, an accurate simulation of how the D&D combat rules behave at massive scale, with no abstractions that affect those results, or in this case additions.

Mearls’ system, and mass combat rulesets like it, change the nature of combat when they group multiple individuals together and have them act as one. If a stand attacks and misses, all 10 individuals miss. If they hit, all 10 hit, and if they critically hit, all 10 of them do. These differences are acceptable when dealing with how large groups of combatants interact with one another, but as soon as a solo player is introduced, things become a problem. If one stand of decently trained combatants critically hits a player character, 10 people have crit one person. In normal D&D, that’s a death sentence, especially if you’re character who tends to sit in the back line. So in order to make it fair, you will switch to normal D&D combat at that scale. This means, in some way you’re playing a bit of a different game at the large scale. And even with this abstraction, managing the rolls can still become tedious for the DM, as mass combat rounds can last for a very long time due to the number of units present that cannot be combined due to how they would reasonably move in combat.
One very important thing this system does bring up, however, is morale. At the scale of armies fighting, it’s fair to assume that unit morale will come up. Units will begin to waver as their odds look worse, affecting their abilities in combat. On the other hand, a charismatic leader can change everything. People trying to use mass combat will very likely want morale to play into their game, and the way it is handled in Mearls’ system is a good example of how to do it. For this and other ways of thinking about large-scale combat, this system is a useful reference and despite not being a program, is one of the best options available in 5th edition for solving the problem I am trying to address. So, if the question is where I fit in, it is close to this document. I have not managed to integrate morale and other systems like this into the system, as they were far-off stretch goals anyways. They are useful, but do go beyond the scope of trying to simulate D&D 5e’s normal scale combat at a much larger scale. Barring a few creatures’ specific abilities, morale isn’t a basic mechanic in this game, so it isn’t unreasonable to omit.

To summarize:

1. Users can group creatures together and play D&D with those groups so that they can reasonably run a mass combat encounter.
2. Users can use special combat rules to run a 5th edition D&D mass combat encounter with unit groups, like a siege or major battle.
3. Users can simulate, in an analog fashion, the morale of creatures in a mass combat encounter to add additional fun mechanics and simulate realistic gameplay.

It may only be three features, but keep in mind this is an addition to the base gameplay system. These rules can be used on Roll20 or a system like it, or on the traditional game grid. This is the abstraction of game mechanics that I sought to avoid when creating my system, to set myself apart from this very thing.
Virtual tabletops, balance-testing combat simulations, and mass combat rulesets, are all different parts of the large world of 5th edition D&D that have pieces similar to my program. While similar in intent to Mearls’ “When Armies Clash” and in function to Dr. Ferla’s “D&D Battle Simulator”, there is still nothing that does quite what I’ve set forth to do. While it isn’t a suite including the functionality and polish of every tool I’ve covered, it is a tool, one that I can work on further even beyond this thesis.
Implementation

Upon starting the program, a command line window will open and you will be met with a view like this.

```
- - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - -
```

1: Change board size
2: Add Unit
3: Add Unit from File
4: Remove Unit
5: Move Unit
6: Make one unit attack another
7: Display unit statuses
8: File Management
9: Launch Test
10: Launch Game
11: Wipe Board
12: Spawn Test Set
0: Exit Program.

Options are entered by typing in a number and pressing enter. Above the options is a map. Dashes are empty cells on the map, under the hood those are null pointers. The @ symbol is a unit, color coded to indicate what team they belong to. This one is a member of team 2. The map size is configurable by the user using the main menu. The map is two dimensional, there isn’t a Z axis that I’ve just failed to render. Units can be
added, removed, and moved around. Already, you can see from this menu that user stories #1 and #2 are met, allowing creation and removal of units. Units are created with a default arrangement of stats, but they do have basic stats enough to satisfy requirement #4. Check the unit class diagram in the class UML on page 26 for a list, they’re in the private variable section at the top of the unit class. Unfortunately I’ve only implemented the baseline attributes, story #5 includes special attributes like class features, which is a whole can of worms I have yet to approach.

With option 10 you can launch the actual game, which creates an automated gameplay session. It rolls the initiative of all units on the field, which is how you order who goes first in D&D. Here we have my first deviation from the game’s basic mechanics, but this one does not affect gameplay. Normally, you roll a 20 sided die to determine your initiative, adding your unit’s initiative modifier. In this system, the number generated is between 1 and 30,001 because there’s far more units, and because I’m using a computer there’s really no reason not to do this. In fact, if I don’t do this it gets far more complicated to make initiative work, as it would mean the range of valid initiative values go from 1 plus the lowest initiative modifier on the field, to 20 plus the highest initiative modifier on the field. At best that’s a range of say, 1 to 40, and that would be extreme. The only way this affects the math is I had to multiply each unit’s initiative modifier by 1,500 in order for that stat to scale effectively. After all units have rolled their initiatives, they are loaded into a list, ordered from the highest initiative to the lowest. Here is a view of the game option in action:
Here you can see the location of each unit on the grid, the coordinates of each unit, their team number, and their health. The only option is to advance to the next turn, so that you can watch the fight unfold between the combatants you’ve set up.

Speaking of which, we have now gotten to requirement #3, the actual automation of combat, and the prime focus of this system. A function is run for each unit’s AI. Currently, the unit does not move on its own, and will only attack enemies next to them if they’re placed next to an enemy in the first place. After that, they will use a ranged weapon. The AI automatically runs the process of randomly generating a number equivalent to a 20 sided die, following the rules of D&D, and checking that roll plus any relevant stats against the enemy’s AC, or armor class, which determines how hard it is to hit someone. If the attack is higher than the AC, random damage is rolled,
and dealt to the opponent. The game then checks if this has killed the enemy and if it has, removes them from the board.

This is where one of my major issues in development cropped up. The turn operates by looping through the initiative order. Removing elements from anything you’re looping through is not advisable, because while there’s likely a way to do it safely, it tends to cause you to accidentally skip units you shouldn’t or try to run AI on non-existent units. This was introducing many problems for me I had trouble solving, and reworked the system so that units were removed from the game board, but not from the initiative order. Then, the program was rewritten so it doesn’t run AI on units with a health value of 0 or lower. They were kept in this limbo until all units have gone through one turn, after which you exit the initiative loop and all dead units are removed from the initiative.

This is a flowchart I built to design the units AI. The functions for moving towards the enemy have unfortunately not been finished, but it does still run through this process.
First, the unit checks if it is in melee with an enemy. If so, strike the enemy with the lowest HP adjacent to them. If not, check and see if the unit is flagged as being “ranged”. This option exists to designate units as ranged or not, so you can have dedicated archers that will stand there and shoot their enemies while other units charge the enemy and attack with a melee weapon. If they are ranged, run the ranged AI. If not, try and move towards an enemy and close to melee. If this works, the unit strikes the enemy with the lowest HP near them. Otherwise, after exhausting all melee-related
options, and they have a ranged weapon, they also run the ranged AI. If they have no ranged weapons, they’ll pass their turn.

I’ve separated the ranged AI as its own block when explaining this both because it flows better in the paragraph and flow chart, and also because it is literally its own function in the program. The AI first checks to see if an enemy is within normal range, which is the distance you can fire your weapon at an enemy without taking any penalties to your chance to hit. If there are enemies in that range, fire at the one with the lowest health. Otherwise, the unit will try to move towards the nearest enemy. From there, they will check again if anyone is in normal range and fire at the weakest if they can. If that’s still not true, they will check for enemies in long range and fire at the weakest enemy they can. If all that fails, they’ll stand there and do nothing.

This is how I’ve satisfied the requirements in user story #3. Units can be told, with the game launch option, to automatically fight. Their allies and enemies are determined by their team. Because the map size can be expanded, this does mean that you can get into the territory of automating a number of units that would be unreasonable to manage normally. Unfortunately, because units only use ranged weapons and do not move, it is a limited fulfillment. You will notice no option to modify any stats on the field manually, requirement #6. I have yet to fulfill that.

After running through everyone’s turn, the system pauses for user input, allowing them to look at how the game has changed, the statuses of all units on the field, and get an idea of what side is currently winning. This repeats automatically until a
team wins, after which the game is ended and the user is returned to the main menu, with the ending battlefield state intact. As stated, there’s no automated movement. Units that don’t start directly adjacent to each other will use their ranged options, generally firing at normal range at whoever the weakest enemy is.

Now let’s talk about magic, requirements 7 through 9. Magic is entirely absent which, while a major component of the game, is also a project in and of itself. The spells in D&D were designed with the openness and room for creativity that tabletop RPGs offered in mind. Spells like fireball, which is just a big explosion, would be relatively simple to program. However, a number of spells change the state of units in unique ways that would require the unit to be flagged in their own unique way. You’d need a way to determine whether or not a creature is held, enlarged, whether they’ve empowered their strike with a branding smite, whether they are currently surrounded in the effects of booming blade, and so on. There’s several hundred spells in D&D, many of which have these kinds of unique conditions, or spells that inflict the same conditions with different durations. Spells can affect vision, like fog or invisibility, which means the AI needs to account for line of sight to the target. While it’s a shame they haven’t made it into this thesis project release, I saw this coming from the beginning.

So far, I’ve satisfied requirements 1 through 3. Under the hood, #4 is met in some capacity because units do indeed have all basic attributes and statistics. They’re mostly lacking special attributes, like class features, feats, and special monster abilities. They have all the normal numerical attributes of almost all units, like physical stats, health, movement speed, and AC. As stated, check the class UML on page 26 for details.
However, because this has to be done by actually entering it as code in the program, it’s hard to call that done. #5 is just an extended form of #4, including the special traits I mentioned. These stats also cannot be edited live, as desired in story #6. Spellcasting is not present, so #7-9 don’t apply. #10 is partially working, though it is unfortunately bugged. It irregularly crashes while running tests, it’s the next item on my list of tasks but unfortunately I have run low on time. #11 and #12 regard the saving and loading of units.

Story #11 is the saving and loading of units, a feature that was close to developed early on when writing the program. However, due to changes in the unit class, and a shift in priorities, it has been left behind and moved down in my list of priorities. Finally #12, which is the requirement for a large library of existing units for the user to pull from, is also not finished as I have no library built nor a system to properly pull them. As of right now, units are stored as CSV files, and that is my plan going into the future as it is a convenient flat file format. I did this so that I could build a library from a text editor or excel, rather than the program itself, to make it faster for myself and savvy users who prefer to add their monsters manually.

These requirements cover my whole dream for this program, and if they were all met, I genuinely think it would be a viable competitor to Roll20. It would appeal less to the common user who likes the fancy graphics, and more to the kind of person who plays games like Dwarf Fortress, a somewhat well-known game that uses text graphics and keyboard commands like mine, simulates an entire world’s history, and is infamously complex and hard to play. Automatic spell casting has its perks, so if you can
get past the primitive design, this system has potential. Because my goals were that
high, I do not consider the game’s current state to be a complete failure, as I was
shooting to create something that could act as an alternative to a crowdfunded product
that has paid employees.

**Decisions**

I had to make a number of choices with my approach to this system. I chose C++
because it was the language I was the most familiar with. While I was tempted to use
Java, as it was something new and I felt it would be beneficial to learn a new language, I
had never attempted a project like this before so it would be a new experience for me
regardless of the language I chose. C++ also has the advantage of being an old and well
known language, which has its ups and downs. The downside is it’s a bit clunky
compared to some the sleek new languages like Python, and you can more easily break
things. The upside is that the reason you can break things so easily is that you have a lot
of control over what you’re doing. I can’t imagine I would ever run into something that I
couldn’t manage to find a solution for in C++, given enough time. There’s a lot of
libraries for C++, as well as for plain C, which can also work. In fact, the CSV Parser I’m
using, though not fully implemented yet, was originally written in C by someone who
released it under the MIT license for free use by anyone [5]. I chose Windows because
it’s what I use for my home PC, and there’s nothing about Linux that’s more or less
useful for this program.
The program is organized into five classes: board, csvparser, game, unit, and util. Each class is a group of code split across two unique files, organized this way for various reasons. Unit is the core of the program, handling the attributes and many functions for units. Board contains the game board and a number of functions for interacting with the game board, like unit creation, unit movement, and reading values from units on the board. The game class contains a board, and a collection of functions and variables to manage the game, like the initiative order. It is also the class that holds the menu and its various option functions. The util class is only there for the ClearScreen function, an admittedly cheap way of using code I found online for clearing a windows command line screen [1]. I put it in its own class so I don’t have to copy paste it in any other class I want to use. As mentioned earlier I also found csvparser online, which is exactly what it sounds like, a set of tools for parsing through csv files more easily. It’s intended to be used for saving and loading units and all their respective stats, but that feature has not been updated with new developments, so it is not fully functional as of right now.

Finally, the main.cpp doesn’t do much beyond setting up some variables and running the menu, which runs out of the game class. Here is a UML class diagram detailing the classes, excluding csvparser due to it not being mine, and not being relevant to the current version.
The Development Process

I began this project in a very different way than I’ve since finished it. Initially, work was organized into a large mass of user stories on a Trello board. I took a hiatus from development due to being employed, and did not end up using the system when I resumed writing code. Part of the issue is that it felt like a fairly needless formality. I was making this tool for myself and people like me, writing down what users want and why was useful for brainstorming but ultimately I knew what I wanted this tool to do. The segmentation of work may have been beneficial if I could have fully committed to it, or
to the Getting Things Done system that Professor Brooks encouraged us to use in one of my classes. Or, I could have committed to Scrum and used pivotal tracker, in line with my senior capstone project that I did in parallel with this project. In the end, I didn’t use any of those tools to aid my work. I found I was over-planning, rather than actually doing the work. Once I started to work however, it felt like I wasn’t ever really progressing with the project. This is where it truly hit me how important project management is, especially when the project has a distant deadline with no feature release dates along the way, no structure or plan to work off of to discourage procrastination and build motivation.

What I focused on first was experimenting and trying to figure out how best to represent the very pen and paper focused system of D&D, as a program. The unit class came first, because I wanted to figure out how to model what a character is. From there I built a basic game grid and created the board class, working on those two as I developed the groundwork for the program. I kept adding more and more D&D stats that I knew I’d need to keep track of for the system, eventually building the game class so that I could properly test these functions by creating a simple command line interface. I wanted this interface to just be the beginning, and to move on to something more advanced. However, the command line turned to work perfectly fine for my purposes anyways, so I’ve kept using it.

Prioritization was a bit skewed in the initial stages, and I’ve grown to better understand what the most important elements of this program are. I focused a lot on the saving and loading system originally, because I assumed I did not want to be
manually entering units in the code. However, that turned out to be perfectly functional. Saving and loading is absolutely on the list of major, important features. However, to demonstrate the basics of the system, or run some simulations, that’s just a convenience. This is one of the reasons I pushed against the constant state of planning, in my experience programming, you start to grasp what you actually need to do a lot better once you dive into the code and start trying to build something. Certain things are easier or harder than you expected. In order to get the results you want to demonstrate that other parts of the program are functional, you discover it’s more important to build the interface than it is to implement other features.

One of the things that’s come up when making this system is how hard it can be to choose where to put functions across all classes. The game class has a board object in it, and the board is made up of a 2D vector containing pointers to unit objects. Currently, the unitTakeTurn function is in the game class. It wouldn’t make sense to put it in the unit class, as it relies on accessing the game board in order to decide what to do and where to go. However, that means the function could have been included in the board class or the game class. I put it in the game class just because it is intended to include game logic, and the AI in unitTakeTurn belonged there. The board class could have technically just been a part of the game class, but they were separated for organization.

While this whole experience has been educational, the unit class has given me some specific challenges. D&D requires you to keep track of a lot of different variables. You need your x and y coordinates for the grid, your HP, the team you’re on, your
initiative roll, how fast you move, how much movement you have left, whether or not you are ranged or have a ranged weapon, your 6 attributes, your 6 attribute modifiers, your armor class, the damage of your weapon or weapons, and your ranged weapon range increments. Some of these aren’t necessarily tracked in D&D, for instance there are no ranged or non-ranged roles or any places on a character sheet to mark whether or not you have a ranged weapon. You simply look at the sheet, or the monster block in the case of most non player characters, and determine if you have a ranged option. I never would have thought that these kinds of variables were necessary until I attempted to make this program.

One of the constructors for the unit class is excessively large, with an input for every variable that isn’t derived from another input. It made me question whether it was better to have a default or simple constructor, then change all of those attributes using other functions. It’s a question I don’t have an answer to, most aid online talks about the very specific issues other people are facing, and the nature of my problem is fairly different. Another challenge was figuring out how to store damage, as damage is represented by a die size, number of dice, and a modifier. The problem comes when you have multiple kinds of dice for an attack, such as a sneak attack die that is a different size than the die used for the weapon itself. I used a struct, where one value is the die size, and the other is the die count. Each attack variable is a vector of these structs, it is a variable sized list of pairs of values.

Most of the other functions are accessor functions to get private variables, or adjust variables like moveLeft to indicate that a unit has moved previously. There’s also
the rollInitiative function, which generates a random number from 1 to 30,001, plus the
dexterity modifier multiplied by 1500. I used the rand function which is a bit outdated in
some people’s eyes, as there are fancier randomizer functions out there. However I
found them more complicated than they were worth, and so went with the simple and
classic rand with a seed set to the time.

The findEnemiesInRange function was a fairly large roadblock I had to deal with.
In the end the issue was due to something as simple as switching the X and Y variables
when trying to retrieve values from the game grid, but there were other issues along the
way. Its difficulty was more from testing, the code itself isn’t terribly complicated as it’s
mostly just a series of if statements and a for loop, nothing special. The issue came from
ensuring that the function, which searches for units within a range around a given unit,
didn’t go outside the size of the grid. Once it starts doing that, it tries to use pointers to
units that don’t exist, and was creating undefined behavior. When drawing borders for
the search in all four directions, it will cap the range at the borders of the grid. It pulls a
vector of all the units in that space with another function, then sifts through those
values and only adds units that are on a different team and grid space than the origin
unit. The function that pulls all units in that zone is fairly simple, accepting four borders
and iterating through a box looking for all spots on the grid that aren’t null pointers.

There’s a number of other minor parts of the program I’m proud, or at least
happy with. To differentiate members of different teams I have the text output different
colors using ANSI terminal codes. This and the use of windows specific commands like
the one I use to wait for system input has limited how the program can be used, but
they have absolutely come in handy. Those terminal codes are also fairly common, so I don’t think that will cause issues if I later design the program to run better if compiled for linux. There is an issue where, the first time the codes are rendered, it needs a second to recognize them as terminal codes and actually prints them as text, so they have to be rendered again. So far, it just doesn’t run any AI until the user has entered input after the initial, odd looking page renders. Between pages the screen is generally cleared, which helps it feel a lot less like you’re using a command line.

For all the time put into this system, there’s not much in it that looks hideously complicated. I’m not accessing any advanced or obscure libraries, or performing complex calculus. D&D’s math is often done by hand anyways, so any calculations that a person would be expected to do have to be reasonable and not detract from the fun. That only leaves so much complexity for the code to have, most of it is trying to recreate logic and reason that a human would use for making decisions. That’s why there’s such a heavy reliance on if statements in this system. The most advanced math I’ve seen done in a D&D game was the time I used the Pythagorean theorem to barely get out of a situation I wouldn’t have been able to if I had counted my movements like a game grid, and the time we attacked as a group and dealt enough damage to a single person to level a large city. While the second involved some obnoxious numbers, again, it wasn’t terribly complicated besides the fairly ridiculous numbers of dice, and the variety of dice, that were rolled in that event.
Conclusion

There are many things I wish I had done differently in hindsight, which is hard to avoid when making such large projects like these. This was educational not just as a programmer, but as a first experience managing such a long term, solo project. I hadn’t truly appreciated how incredibly difficult it can be to segment work like this and reliably progress on it, day after day, while other things are all trying to take your time. I would have started this program far earlier, putting it in a more developed state than it is now. AI for movement, a way of supporting larger maps more easily, and a more fleshed out unit loading and saving system would be ideal. However, I would not let its development continue long enough to get into my final term, but rather I’d prefer if it had been finished months beforehand in order to avoid the pressure of completing the thesis so late. I also would have liked to learn a better rendering system for the program. Terminal rendering is all well and good for a simple system, but given more time I would have been able to look into the Simple DirectMedia Layer system for C++, which is the tool I initially wanted to work with until I realized I did not have the time to learn it and re-write all my displays, accepting the delays that additional mistakes might bring.

Despite this, this is still the largest independent project I have ever done. I’ve had to set aside any of my perfectionism with this project because, as I have learned from my classes with Chris and Ted, scope creep can be incredibly dangerous. I could keep adding features and functionality to this system until I’m trying to sell it. I’m still proud of what I’ve managed to create, and I’ve put enough thought, time and planning
into features beyond its current state that it can be expanded even more. If I need a project to prove my effectiveness as a developer, this will likely continue to be my pet project to add to as I grow as a professional. I did not use agile or any real formalized system of iterative development with this project, and from my experiences I see more clearly why it’s important. Tasks this large need to be segmented up into achievable pieces in order to manage them well, and also to help it feel more achievable.
Project Code

The code is hosted on GitHub, and will be updated as I progress on the project. It is at the following URL: https://github.com/AOhnemus/DnD5EMassCombatSim.
Bibliography


