RAWR: Algorithm for Dungeons and Dragons Strategy Optimization

Valerie Pietrasz[1] and Thomas White[2]
Stanford University, Stanford, CA, 94305

This paper investigates a problem of critical importance for engineers and computer scientists everywhere – the optimal strategy for 5th Edition Dungeons and Dragons combat encounters. Dungeons and Dragons is an excellent form for investigating decision-making strategy, since it is an environment that is uncertain, but also operates according to known rules. Although much of the game is based on storytelling and therefore out of the scope of decision-making algorithms, it is founded on a robust system for the resolution of combat encounters. Randomized Actions With Rewards (RAWR) attempts to algorithmically solve the optimal combat strategy using an online planning approach. In addition, by creating an easily-extensible framework that implements these encounter rules, it seeks to create a structure that can be used easily for further investigations. These advances have potential to create a useful tool for DnD creators and enthusiasts going forwards.

I. Introduction

Dungeons and Dragons (DnD) is a pen-and-paper role-playing game with a great deal of complexity that is impossible to model, because it relies on storytelling, social dynamics and worldbuilding for much of its appeal. However, within the broader system is a very clearly defined mechanism for simulating combat encounters. Characters fight foes, making choices about a predetermined set of actions they can take and how each one of them affects a set of metrics governing the stamina and capabilities of their foe. This project seeks to simulate a very simplified model of a combat encounter in the DnD 5e ruleset, beginning with an encounter in between an agent and a single foe, in which the agent seeks to identify a strategy that minimizes damage to itself while still eliminating its foe.

This problem is of interest for both practical and academic reasons. Practically, DnD is a common pastime the world over. Simulation of the optimal strategy given a certain framing of an encounter can provide perspective for players looking to hone their game; in addition, although this project only provides a foundation for such an effort, it can ultimately be used as a resource by people attempting to decide if a character or weapon they have created for DnD's system is appropriately challenging, by providing a quick way to determine if an average player would be suitably challenged by it. This could one day be a great resource for game designers looking to iterate quickly.

Academically, DnD combat is an excellent example of a decision-making problem, because of the several levels of uncertainty lead to complex optimization and the large scope of both possible decisions and possible states prioritize online planning and other algorithms which can save computation time[2]. As such, it is an interesting way to investigate the generalized process of solving Markov Decision Process problems.

II. Problem Statement

This project seeks to find an optimum policy for a combat encounter. The encounter is framed by combat in between an agent, representing the player and thus the decision-maker and the foe, representing an antagonist usually controlled by the dungeon master (DM). Success or failure in the encounter is tracked by both party's health points (HP) – when these reach zero the party dies. As such, the optimum policy is framed such that the agent does

[1] Student, Aerospace Engineering
[2] Student, Aerospace Engineering

as much damage as possible while sustaining as little damage as possible, with a large reward for killing the foe and a large penalty for dying itself.

In pursuit of this goal, both parties can take a variety of actions. Some actions directly lower the HP or their antagonist or raise their own HP, but others impact the battle more indirectly, by raising or lowering each side's potential to do damage in the future. Such trade-offs in between present and future reward are common in DnD – for example, a player might be able to either lower the damage that they take, or increase the damage that they give out, but not both. Crucially, this also means that the list of available actions for both parties changes throughout the simulation.

Once both parties have settled on an action, they are settled according to a slightly randomized process. Both parties generate a random number (in an actual game, by throwing dice), boost the random numbers with modifiers derived from their attributes or actions, and then compare the random numbers. If the attacking force is larger, it then makes a slightly random impact on its target's status. Thus, there are two main sources of randomness within the action resolution – whether the target is able to resist the action entierly, and the magnitude of the action's effect. This outcome uncertainty constitutes the first main source of uncertainty in the simulation. In addition, because the agent does not know the ability scores of the foe, upon which its modifiers are predicated, there is a small element of parameter uncertainty as well. [3]

The second source of uncertainty comes from the state of the foe. In both DnD and in this simulation, the foe's state is unknown to the player, but sometimes communicated through very noisy measurements. In a normal game, this is in the form of verbal description by the DM, which provides a qualitative sense of the foe's status. In this simulation, we have chosen to implement this knowledge as a noisy measurement.

Finally, the actions of the foe are not deterministic. In traditional DnD, they would be made based on the DM's discretion. Although they are at least somewhat focused on eliminating the agent, the DM may have other narrative motivations unknown to the player, making it difficult for the player to always assume that their foe will take the action of maximum utility. As such, we have implemented a weighted function for the foe's actions – it will sometimes take the action of maximum utility, but other times will choose an action randomly regardless of utility.

The code is accessible in the references at [1].

III. Approach

From previous experience, we chose to design the programming environment in Python, which provided a useful optimum in between broad commonality and accessibility and power.

To solve the problem, the first goal was to implement a complete model of the encounter that can be easily accessed by optimization algorithms. This had four key elements.

First was the creation of agent and foe classes. This allowed the tracking of a single agent or foe entity and their states, determining both their current status and what actions are available to them. Since all of that information is available in the function, the classes also implement functions that check their available actions, interface with their state, and construct their functions. As such, they are framed such that each and foe is fully replaceable – future implementations of the code could easily try out alternative agents and foes to see how they behave while trusting that their implementation throughout the code works smoothly. This is key to the long-term applicability of code like this one to experimentation with different configurations. It also allowed us to iterate through the space of possible agents and foes to determine the sensitivity to different parameter changes.

Secondly, we implemented a generalized function for handling actions. As discussed in the introduction, DnD actions follow an extremely general recipe, as above: a party takes an action directed at a target, the target attempts to resist it, both generate a random number and compare them, and then a weighted random effect impacts on of the target's stats. We created a class for actions that can be initialized with the details of the implementation – such as the actor, the target, the particular dice to be rolled, and so on. This way, actions can be called using a single line of details inside the agent and foe, allowing all of the specific details about what they can or can't do to remain inside their class functions, while preserving an extensible structure for actually handling the outcomes of those actions.

Third was a system for uniquely parameterizing states. During a DnD encounter, the state is uniquely described by the state of the agent and foe combined, each of which have a number of different state variables, which may even be of different types. To facilitate the ultimate parametrization of a value function, we created an algorithm to hash that information into a single integer, which can then be repeatably unpacked into an agent and foe state. By allowing the state of the entire simulation to be more easily stored and implemented, it became easier to work with the code.

Finally, a top-level set of functions automated the process of taking turns. In real DnD rules, there is an element of randomness in whether the agent or foe can act first, but in this simulation, for simplicity, the agent always does. Our structure made it easy to call a turn and have both parties resolve their actions. Because the agent does not actually

know the state of the foe, an extra detail was added – a second foe object to represent the agent's belief, named the faux foe for clarity and alliterative pleasantness, which is updated with noise based on the state of the foe. As mentioned above, in a real game, the player's understanding of the foe's state would be based on qualitative descriptions by the DM – one of the only intrusions of such qualitative reasoning into the otherwise very rules-bound approach for DnD combat. To add a small amount of color, a knowingly slightly inefficient approach to communicating this information was implemented. Upon being damaged, the foe responds with printed cries of pain, which the agent can noisily interpret to identify the foe's remaining HP.

Once this structure was complete, the next step was to figure out how to implement decision-making and optimize over it. The first approach we considered was Monte-Carlo optimization across the space of actions, ultimately converting to a deterministic policy with the maximum utility. However, ultimately this was deprioritized against a branch-and-bound based online planning approach. Online planning has the advantage of considerably reduced computing time [2], especially in a game where the number of possible states is extremely large but many of them are unreachable. It also more accurately simulates the process of DnD play. Given the very high uncertainty, which piles on during play, as well as constantly changing state itself, DnD is not especially conducive to a single deterministic 'optimum' policy in any case. A program that could more easily adapt to the decisions made by the actors within it was judged much more useful and accurate as a simulation.

Finally, a top-level implantation of an encounter rounded out the package – this encounter structured the optimization process and how it interacted with the underlying structure of the problem. Jupyter notebooks were used to graph the results and served as a high-level interface.

IV. Detailed Implementation

As is usually the case, the code required a number of changes of direction during implementation. For instructive reference, some of the important ones are detailed below.

Our transitions were originally calculated by sampling a random action resolution system. However, we ultimately used a function that calculated the expectation of the action, based on the expected die rolls from all parties, which offered a deterministic picture of the next step instead. An aspect of transitions that proved to be too challenging for the scope of this project is effectively capturing action prerequisites - this is discussed further in results.

After some thought, it turned out that branch and bound was not as useful for this application. Traditionally, the goal with and bound is to lower the computation time with respect to forwards search by pruning some tree directions that fall below the minimum utility of a different branch. However, setting constructive bounds on the utility or action value function proved difficult, especially because of the different models of utility that different actions implemented – some actions were not even available some states, and when they were, many did not directly provide utility but instead just gave other actions additional utility. Given these concerns, the branch and bound algorithm was unlikely to be optimum and unlikely to help very much in computation, leading to a scoping change to a simpler forwards search algorithm.

One substantial point of concern throughout the analysis was the prospect of trivial solutions being generated. In order to minimize this prospect, a fairly complex and interlinked action space was created for the agent, with the hope that the action space would lead to an optimal policy that had to balance a mix of short-term and long-term goals.

A wholly numeric rendition of the dungeon state became a point of contention. It was originally included to make it possible to parameterize the state. However, the transition away from Monte Carlo methods made the need to create a parametric state vector much less pressing. And since every opportunity to use it tended to immediate require either the agent or foe attributes to be unzipped in any case, it seemed to largely only add computation time. Future extensions could find this capability very useful – for example, if past runs were to be stored or transmitted easily. However, in the current implementation, the code ultimately turned out to not have much purpose.

At this point, it became clear that a lot of iteration was needed on the hyperparameters, many of which hadn't seemed to be hyperparameters at the time. For example, the depth of the forwards search and many of the specifics of the agent and foe became things to be played with. To facilitate easy access to all of this work from the top-level function, pandas data frames and a structure of keyword arguments were implemented, so that all the hyperparameters can easily be set from the top level without a requirement to dig through the code.

V. Results

Model setup and initial implementation having been completed, effort turned to tuning the hyperparameters and assessing the performance of the model. Here, the fundamentals of the approach were validated, but a number of issues were uncovered for future work to delve into.

On a functional level, a number of the elements of the code worked exactly as desired. Tracking of the foe's states and the stochastic behavior thereof worked perfectly, with a typical example below.

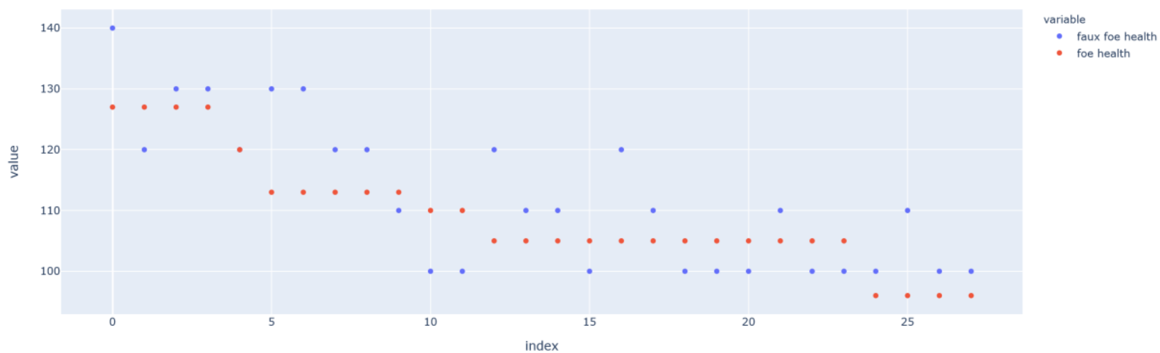

Fig. 1 Foe State versus Believed Foe State Over Time [1]

The agent and foe were able to trade off their damage against once another, moving to a conclusion, and taking actions that maximized rewards.

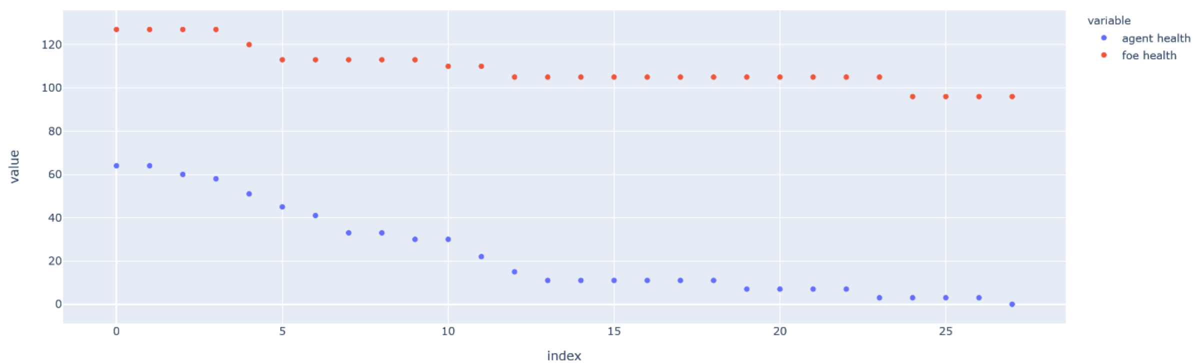

Fig. 2 Agent and Foe HP throughout Encounter [1]

And utility calculations were fully effective – the forwards search approach with estimated utility closely tracked actual utility.

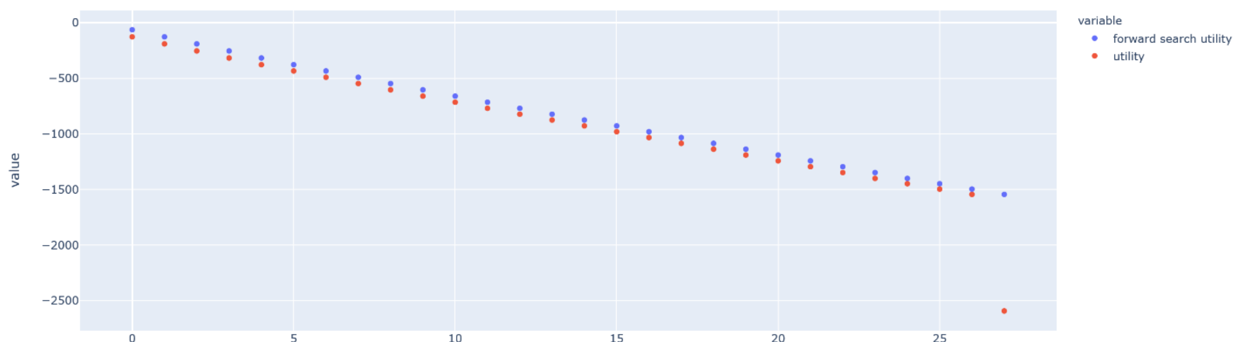

Fig. 3 Forwards Search Utility versus Actual Utility [1]

However, there was still a tendency by the agent to converge to a trivial solution, using almost entierly its basic hit attacks and not engaging in many of the more complex actions that could lead to more complex benefit tradeoffs. We investigated several paths to understand this behavior better.

The first is also the simplest: game balance. Part of the reason why the RAWR algorithm would be so useful if fully implemented is that game balance in DnD is a very complex and multi-dimensional problem. Often, the success or failure of DnD implementations relies entierly on designer intuition and experience to come up with problem framings that are challenging but not impossible to overcome. We found that in order to prove our algorithm, we needed to create an initial combat encounter that was well-framed *a priori*, to prove that trivial solution spaces or modeling failures were coming from the model and not from the bad character choices. Ultimately, we did a great deal of iteration across the parameter space, changing both party's stats, values, and even the mechanics behind their attacks. However, although we were able to make a lot of progress – ultimately concluding in a problem framing in which victory for both agent and foe was possible, depending on the random outcome, we were still unable to fully balance some of the agent's powers and still found trivial solutions dominating. As such, we proceeded to a second area of inquiry.

Our second area of investigation was in the fundamentals of the value function. We had initially selected the online planning approach because of its similarity to the actual process of DnD play and because of the high-level parameter space. However, we discovered that there is a crucial and subtle difference in between the DnD problem and those that have often been solved using online approaches – namely, the large, highly state-dependent and changing space of acceptable actions. Many of the algorithms in online planning depend on an effective transition function in between the state and the next state that is easily traceable in both directions, allowing them to iterate through a chain of actions both forwards and backwards[2]. This allows an approach like forwards search to proceed first to the maximum depth and, working backwards, determine the expected utility of pursuing a branch. In our case, however, this state transition function does not exactly work backwards, because many actions are causal. For example, at a given state, the actor's ability to cast a spell depends on if they have previously cast enough spells to use up their total spell allowance.

This is a crucial issue because in DnD every action besides default attacks is useful precisely because it opens up the state space for future actions – by giving the actor more things that they can do – or provides more reward in other ways, like preventing the foe from acting as strongly. Because of its lack of causality, the RAWR algorithm could not take these concerns into account, leading to a trivial solution.

As such, it is the conclusion of the research team that online planning approaches are not, in fact, very effective at solving the DnD problem. A Monte Carlo approach was investigated for improving the results, and initial results looked promising, with a mix of actions being taken, including spells.[1] Further work would be required to prove out that this approach is as robust as the original one, by iterating through the space of possible encounters.

VI. Future Work

The authors are very excited about the current outcome of the work. The current program simulates the realistic behavior of an agent and foe, and appears to do so approximately as well as an extremely mediocre human player. This is an important first step to a complete analysis of the DnD problem. However, the program currently does effectively factor in the future values of actions, especially how they affect the action space. Future work should therefore focus on this aspect in order to deliver on the promise of this algorithm.

Given that online planning methods seem fundamentally unsuited to this problem, the authors suspect that further investigation of offline planning methods could actually be most useful. Since the problem is discrete, actions to increase the speed of the solving, including using an integer space like the one implemented in this code, could help a complete, exact solution of the problem become possible.

Another step for future work would be to help handle situations not currently modeled for. Most crucial are two elements: movement and multiagent behavior. The first is structural. Movement is a key element of DnD – not to mention any semblance of real combat –and composes some of the most interesting strategy. By moving about a play field, both agents and foes can restrict the actions available to their antagonists, or boost their own options. Given the complexity of implementing geometrical movement of this type, it was scrapped in the current version of RAWR, but its inclusion in a future version will greatly increase the efficacy of the simulation.

Secondly, and even more crucially, DnD is a game that thrives on group, not solo play. The current implementation has only a single foe, while real DnD usually has several. But, even more crucially, it also only has a single agent, whereas much of the strategy of real DnD lies in the interactions between several agents, and how, working as a team,

they might be able to emphasize their relative strengths and make up for their weaknesses. A multi-agent model could serve as a crude model for this behavior, although obviously without the players having an ability to explicitly communicate about strategy, and the basic structure of our model of DnD is extensible to this possibility. However, like movement rules, it was judged ultimately too complex to implement in the current implementation.

Finally, future work could be useful in further exploring the space of optimization algorithms. More advanced forms of online learning and optimization could provide an even more utility-maximizing result, potentially in less time. Thankfully, the structure is robust enough that other approaches could be built atop it.

VII. Conclusion

Our program's first implantation, RAWR, is currently capable of simulating a single simulated encounter with a single agent and foe. It provides the fundamental proof of concept and building blocks for much better future implementations. The authors are excited for the forthcoming field of DnD research that they hope will develop from their work, and can't wait to experience future researcher's findings.

VIII. Notes on Work Completed

All of architectural decisions and most of the coding work was done synchronously between the two team members. Relatively speaking, Valarie did more work on the state transfer function and Thomas more on the agent and foe state definition. In documentation, Valarie did most of the work on the figures and code outputs, and Thomas did most of the documentation and writeup.

Acknowledgments

Many thanks to Mykel Kochenderfer, the teaching team, and everybody else who helped the authors understand decision making under uncertainty. The authors would also like to thank the creators of Stack Overflow and some of the contributors, who helped the authors understand everything else. And many thanks to Gary Gygax and David Arneson for developing the system that has entranced so many of the authors' weekends.

References

[1] Available Code: https://github.com/thwhite/aa228-group98. Monte Carlo implementation is on a branch and incomplete.

[2] Kochenderfer, M, Wheeler, T. and Wray, K. "Algorithms for Decision Making," [online textbook], Version 1. URL: https://github.com/sisl/algorithmsbook/ [retrieved 9 November 2020].

[3] "DnD Wiki" [online database]: https://dnd-wiki.org/wiki/Young_Yellow_Dragon_(5e_Monster)