Optimal Combat Scenario for Dungeons & Dragons

Alyssa Vellucci and Jonas Wagner

***Abstract*— This article examines the multi-step decision-making process involved in combat encounters in Dungeons and Dragons 5th edition. In D&D combat, players fight creatures controlled by the Dungeon Master. The goal is to maximize the hit points of player characters while minimizing those of the enemies. The system includes the set of PCs and enemies, and the states of the system include the HP values of PCs and enemies, the condition of the PC and enemy, and spell slots remaining.**

Inputs include the actions the enemy performs and the state of the creature, while outputs include the actions the PCs perform on the enemy creature. he uncertainties in this system come from the dice rolls the players make when performing an action and not knowing the specific stats of the enemy they face. Combat is based on initiative order, and each round is a full completion of the initiative list.

This system uses a Markov Decision Process to create the dynamic program. This dynamic programming algorithm has been shown to be up to 70% effective in winning these combat scenarios, and can be shown to achieve higher win-rates than human inputs alone.

I. Introduction

The goal of this project is to create a dynamic program that can optimize a combat scenario for the game Dungeons & Dragons. While the scenario the dynamic program will optimize is not a 1:1 representation of the D&D 5e Ruleset, it is similar enough that it should be recognizable as a D&D game [1]–[3]. Due to the sheer number of decisions that could be made in a typical D&D combat, many limitations had to be implemented so that the program could fit within the scope of the project. These limiting assumptions will be outlined in detail in Section II.

In a typical D&D combat, characters controlled by the players (PCs) face off against enemy creatures (”monsters”) controlled by the Dungeon Master (DM). The actions of PCs and creatures (collectively, referred to as actors) are determined by a dice roll. The roll of the die plus some modifier needs to meet or exceed the so-called Armor Class (AC) to be able to count as a ”hit” and deal damage. The PC’s AC is known, while the AC of the monster is hidden from the players. Only when the players roll some number and hit do they ascertain the AC of the monster.

Players can perform many actions in their turn, including: melee or ranged weapon attacks, spell casts, movement on the battlefield, interaction with their environment, helping other PCs, ingest a potion (or poison), and more. For the simulation outlined in this report, the only actions a PC can take are: move, weapon attack, drink a potion, or do nothing. Monsters will have the same set of actions, sans drinking a potion. The battlefield is flat with no obstacles or borders to limit movement. Both PC and monster have an AC of 15. Their strength and dexterity modifiers (used for the sword and the bow) are +5, but this modifier is only used to determine the success of the attack, not as a modifier on the damage done.

A combat is considered successful if the PCs live while killing the monster. Therefore, the dynamic program will be optimized to maximize the HP value of the PC, while minimizing the HP value of the monster.

The authors are with the Mechanical Engineering Department at the University of Texas at Dallas, Richardson, TX, USA vellucci@utdallas.edu and jonas.wagner@utdallas.edu

II. Problem Definition

A. Limiting Assumptions

TABLE I
Assumptions

Movement	Single deterministic movement of 1 square per turn, followed by an action
Actions	Single action per time step: - Melee (hit check)(d6) - Short range - Ranged (hit check)(d8) - Longer range - Health Potion (d4 + 1) - Nothing
Characters	1 PC and 1 Monster with identical specs and modifiers, using the same weapon (+2)
Monster	Moves in a standard pattern and cannot heal
Time Horizon	Infinite Time Horizon
Battlefield	Infinite with no obstacles

The assumptions for this problem are outlined above. It is important to point out that this is not a true infinite time horizon problem: the scenario does not end when converging to some value. The simulation ends when one of the actors reaches a HP value of zero. Because it is unknown how many time steps this will take, the program is modeled like an infinite time horizon.

B. Environment Definition

1) States: Let each character be associated with position and HP states. For position, let

$$x_{pc,p}, x_{mn,p} \in \mathcal{X}_p \subseteq \mathbb{Z}^2$$

describe the position on an infinite 2-d grid. For HP, let

$$x_{pc,hp}, x_{mn,hp} \in \mathcal{X}_{hp} \subseteq \mathbb{Z}_+ = \{0, 1, 2, \dots\}$$

describe the HP for each character.

2) Inputs: The inputs to the system consist of movement and actions impacting the position and hp states respectively. For movement, a deterministic input of

$$u_{pc,m}, u_{mn,m} \in \mathcal{U}_m$$
$$= \{\text{Stay, N, E, S, W, NE, NW, SE, SW}\}$$
$$= \left\{ \begin{matrix} (0,0),(-1,0),(+1,0),(0,-1),(0,+1), \\ (-1,-1),(-1,+1),(+1,-1),(+1,+1) \end{matrix} \right\}$$

Where 0 is the actor not performing a movement at all.

For actions, the input for the PC and monster are respectively

$$u_{pc,a} \in \mathcal{U}_a = \{\text{Melee, Ranged, Heal, Nothing}\}$$

$$u_{mn,a} \in \mathcal{U}_a = \{\text{Melee, Ranged, Nothing}\}$$

where each of the actions (except nothing) are stochastic.

For Melee and Ranged attacks, the character acts upon another character's HP where the impact on HP is as follows:

1) Ensure in range for either melee or ranged attack - otherwise can't attack.
2) "Roll" for success/fail - if fail then self-loop on opponent HP
3) "Roll" for effectiveness - opponent HP decreased by Weapon/self Modifiers (2) + d6/d8

The PC is also allowed to use a health potion which has a stochastic effect upon the player's health:

1) Ensure potion is available - otherwise can't heal
2) "Roll" for effectiveness - player's HP increased by health modifier (1) + d4

3) Stochastic Elements: The stochastic aspects of the system can be described as a noise signal consisting of the PC and monster dice rolls. The actions themselves can be modeled as either a function of the noise signal or as a Markov chain acting directly upon the PC or Monster HP states.

The noise introduced by dice rolls are defined as for {d2, d4, d6, d8, d10, d20, d100} as

$$w_{pc,dn}, w_{mn,dn} \in \mathcal{W}_{dn} = \{1, 2, \ldots, n\}$$

with $n = 2, 4, 6, 8, 10, 20, 100$ respectively where each outcome is equally likely. The PDF can be seen in shorthand with $P(w_{i,dn} = \begin{bmatrix} 1 & \cdots & n \end{bmatrix}^T) = \frac{1}{n} \begin{bmatrix} 1 & \cdots & 1 \end{bmatrix}^T$.

C. Simplistic System

1) States, Inputs, and Noise: For the simplistic case, let states at time-step k, be

$$x_k = \begin{bmatrix} x_{pc,p} \\ x_{mn,p} \\ x_{pc,hp} \\ x_{mn,hp} \\ x_{pc,potion} \end{bmatrix} \in \mathcal{X} = \mathcal{X}_p^2 \times \mathcal{X}_{hp}^2 \times \mathcal{X}_{potion} \subseteq \mathbb{Z}^4 \times \mathbb{Z}_+^2 \times \mathbb{Z}_+$$

where states and sets are defined as before and $x_{pc,potion} \in \mathcal{X}_{potion}$ is the number of potions available to the PC.

Let the inputs to the system be only the players inputs:

$$u_k = \begin{bmatrix} u_{pc,m} \\ u_{pc,a} \end{bmatrix} \in \mathcal{U} = \mathcal{U}_p \times \mathcal{U}_a$$

The monster's inputs to the system will be incorporated as a deterministic and stochastic input that are closed-loop within the system and treated as part of the nonlinear aspects of the update function.

Let the noise signal w_k for each time-step be defined as a collection of the PC and monster dice rolls,

$$w_k = \begin{bmatrix} w_{pc} \\ w_{mn} \end{bmatrix}, \quad w_i = \begin{bmatrix} w_{i,4} \\ w_{i,6} \\ w_{i,8} \\ w_{i,20} \end{bmatrix} \forall_{i=pc,mn}$$

The associated success or fail noise $\forall_{i=pc,mn} \forall_{j=mn,pc} \forall_{l=dex,str}$ is derived as

$$w_{i,sf} = \begin{cases} 0 & (a_{i,l} + w_{i,d20} \leq a_{j,ac}) \\ 1 & (a_{i,l} + w_{i,d20} > a_{j,ac}) \end{cases}$$

where the modifiers are held constant as $a_{i,l} = a_{i,dex} = a_{i,str} = 5$ and $a_{j,ac} = 15$.

D. System Update Equation

The evolution of the system can be described as Markov chains or by a nonlinear update function.

The nonlinear update function is described as in (1) where the associated functions update states as follows:

- The players deterministic movement input:

$$f_{pc,m}(u_k) = u_{pc,m}$$

- The PC actions dependent on action selection and dice rolls as in (2).
- The monsters state-dependent movement:

$$f_{mn,m}(x_k) = \text{direction}(x_{pc,p} - x_{mn,p})$$

  where $\text{direction}()$ is calculated as the closest cardinal direction that heads towards the player.
- The monsters state-dependent action as in (3).

Weapon and potion modifiers are designed as $a_{pc,wpn,m} = a_{mn,wpn,m} = 0$, $a_{pc,wpn,r} = a_{mn,wpn,r} = 0$, and $a_{pc,potion} = 1$. Ranges are set as $a_{pc,rng,m} = a_{mn,rng,m} = 2$ and $a_{pc,rng,r} = a_{mn,rng,r} = 5$.

E. Markov Chain Realization

Equivalent Markov chains can be defined for each of the steps of a turn instead of the nonlinear mixed deterministic and stochastic functions.

The relative and deterministic nature of these makes updating the system with the update function much easier; however, the stochastic nature of actions lends themselves to a Markov implementation.

$$x_{k+1} = f(x_k, u_k, w_k) = \begin{bmatrix} x_{pc,p} + f_{pc,m}(u_k) \\ x_{mn,p} + f_{mn,m}(x_k) \\ \begin{bmatrix} x_{pc,hp} \\ x_{mn,hp} \\ x_{pc,potion} \end{bmatrix} + f_{pc,a}(x_k, u_k, w_k) + f_{mn,a}(x_k, w_k) \end{bmatrix} \tag{1}$$

$$f_{pc,a} = \begin{cases} \begin{bmatrix} 0 & -w_{pc,sf}(a_{pc,wpn,m} + w_{pc,d6}) & 0 \end{bmatrix}^T & u_{pc,a} = \text{Melee AND } \|x_{pc,p} - x_{mn,p}\|_2 \leq a_{pc,rng,m} \\ \begin{bmatrix} 0 & -w_{pc,sf}(a_{pc,wpn,r} + w_{pc,d8}) & 0 \end{bmatrix}^T & u_{pc,a} = \text{Ranged AND } a_{pc,rng,m} < \|x_{pc,p} - x_{mn,p}\|_2 \leq a_{pc,rng,r} \\ \begin{bmatrix} a_{pc,potion} + w_{pc,d4} & 0 & -1 \end{bmatrix}^T & u_{pc,a} = \text{Heal AND } x_{pc,potion} \geq 1 \\ \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} & u_{pc,a} = \text{Nothing OR Otherwise} \end{cases} \tag{2}$$

$$f_{mn,a} = \begin{cases} \begin{bmatrix} -w_{mn,sf}(a_{mn,wpn,m} + w_{mn,d6}) & 0 & 0 \end{bmatrix}^T & \|x_{pc,p} - x_{mn,p}\|_2 \leq a_{mn,rng,m} \\ \begin{bmatrix} -w_{mn,sf}(a_{mn,wpn,r} + w_{mn,d8}) & 0 & 0 \end{bmatrix}^T & a_{mn,rng,m} < \|x_{pc,p} - x_{mn,p}\|_2 \leq a_{mn,rng,r} \\ \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T & a_{mn,rng,r} < \|x_{pc,p} - x_{mn,p}\|_2 \end{cases} \tag{3}$$

$$\begin{bmatrix} x_{k+1,pc,hp} \\ x_{k+1,mn,hp} \end{bmatrix} = \begin{cases} \begin{bmatrix} x_{k,pc,hp} \\ x_{k,mn,hp}^T P_{pc,melee} \end{bmatrix} & u_k = \text{melee} \\ \begin{bmatrix} x_{k,pc,hp} \\ x_{k,mn,hp}^T P_{pc,ranged} \end{bmatrix} & u_k = \text{ranged} \\ \begin{bmatrix} x_{k,pc,hp}^T P_{pc,heal} \\ x_{k,mn,hp} \end{bmatrix} & u_k = \text{heal} \\ \begin{bmatrix} x_{k,pc,hp} \\ x_{k,mn,hp} \end{bmatrix} & u_k = \text{nothing} \end{cases} \tag{4}$$

$$x_{k+1,pc,hp} = \begin{cases} x_{k,pc,hp}^T P_{mn,melee} & \|x_{k+1,pc,p} - x_{k+1,mn,p}\|_1 \leq \text{Melee Range} \\ x_{k,pc,hp}^T P_{mn,ranged} & \text{Melee Range} < \|x_{k+1,pc,p} - x_{k+1,mn,p}\|_1 \leq \text{Ranged Range} \\ x_{k,pc,hp} & \text{Ranged Range} < \|x_{k+1,pc,p} - x_{k+1,mn,p}\|_1 \end{cases} \tag{5}$$

1) Additional Assumptions: A few additional assumptions must be made to restrict the problem to a markov chain implementation:

- The monster and PC HP will be restricted to a finite range, $x_{pc,hp}, x_{mn,hp} \in \mathcal{X}_{hp} = \{0, 1, \ldots, n_{hp,max}\}$
- The potion action is limited to the number of potions in the PC inventory. If no potion exists, the PC does nothing instead.
- The update of $x_{pc,hp}$ and $x_{mn,hp}$ in closed-loop will be assumed to be independent of each other.

2) Transition Martrices: Transition matrices are derived directly from each of the cases in the nonlinear functions $f_{pc,a}$ (2) and $f_{mn,a}$ (3) applied for a finite number of HP states, $x_{pc,k}, x_{mn,k} = P(x_{pc,hp}),_(x_{mn,hp}) \in [0,1]^{n_{hp,max}+1}$.

The transition matrices will be defined as $[p_{i,j}] = P(x_{k+1} = i, x_k = j)$. This can be thought of as each column being the probabilistic function results for $f(x_k, w_k), x_k = j, w_k = P[w_k]$.

The resultant Markov chains (stored as the struct M in section) are denoted by $P_{i,j} \forall_{i \in \{pc,mn\}} \forall_{j \in \mathcal{U}_a}$.

3) Closed-loop Markov Chain: The actual Markov chains associated with closed-loop will be selected directly based on state and player input by selecting the appropriate markov chains.

Since the assumption that u_k will be limited to those feasible by the PC, the following described as follows:

1) PC movement:

$$x_{k+1,pc,p} = x_{k,pc,p} + u_{pc,m}$$

2) PC action: (4)
3) Monster movement:

$$x_{k+1,mn,p} = x_{k,mn,p} + \text{direction}(x_{k+1,pc,p} - x_{k,mn,p})$$

4) Monster action: (5)

III. Dynamic Programming Implementation

A Markov Decision Process (MDP) implimentation of the dynamic programming algorithm is used to determine the optimal policy which can then be tested against the nonlinear update equation version for simulation.

A. MDP Implementation

1) Additional Simplifications: Due to the nature of the problem being of such high dimension, the following additional simplifications will be done:

- The policy will be only based on relative distances between the PC and Monster. (i.e.)

$$x_{k,pos} = x_{pc,p} - x_{mn,p} = \begin{bmatrix} x_{pc,p,x} - x_{mn,p,x} \\ x_{pc,p,y} - x_{mn,p,y} \end{bmatrix}$$

  This is mainly to reduce the number of position dimensions from 4 to 2 and ensure the finite position is useful.
- The objective function will be only based on the HP of PC and monster.
- A value iteration method will be used with only a psudo-infinite-horizon as computing till convergence is not practical.
- The player potion dimension will be reduced from $x_{pc,potion} \in \{0, 1, \dots\}$ to $x_{pc,potion} \in \{0, 1\}$ representing potion/no potion.

2) Layered Markov Chain: The MDP will be applied upon a finite 5D space $\mathcal{X} = \mathcal{X}_{pos,x} \times \mathcal{X}_{pos,y} \times \mathcal{X}_{pc,hp} \times \mathcal{X}_{mn,hp} \times \mathcal{X}_{pc,potion}$ where the probability of being each location is stored within an array. For each of the potential inputs, a Markov chain is then created in this 5D space using the transition matrices from subsubsection II-E.2. Within MATLAB this is computed ahead of time (and stored in P.mat) and then flattened for matrix multiplication for computational efficiency.

B. Optimal Policy

1) Objective Function: The objective function is defined to minimize the monsters HP and maximize the PC HP while incentivizing monster death and strongly disincentivising PC death. Since it is only dependent on the PC and Monster HP, the stage cost static across all positions (visualization included within results in Fig. 1).

2) Stage-cost Updates: The closed-loop markov chains will be used to determine the probabilities of future states $\mathbf{P}[x_{k+1}]$ that can then be used for the expected next time-step cost computation:

$$\mathbf{E}[J(f(x_k, u_k, w_k))] = \mathbf{P}[x_{k+1}]^T J_{k+1}(x_{k+1})$$

3) Value Iteration: A finite-time value iteration MDP implementation is then applied, with a large finite-horizon, to approach the optimal infinite-horizon result without waiting for convergence. The results of which are stored in an optimal policy $u_k = \pi^*(x_k)$ which has been computed (and stored in pi_star.mat).

IV. Results

A. Stage Cost Function

Fig. 1 displays the stage costs of the dynamic program, with the left and bottom red bars signifying the termination of the simulation when the PC or monster reaches 0 HP, respectively. The objective is to maximize the PC's health while minimizing the monster's, resulting in a significant cost when the monster's HP is at its maximum. The program is structured to minimize negative values, effectively maximizing the highest cost when the PC is at full health.

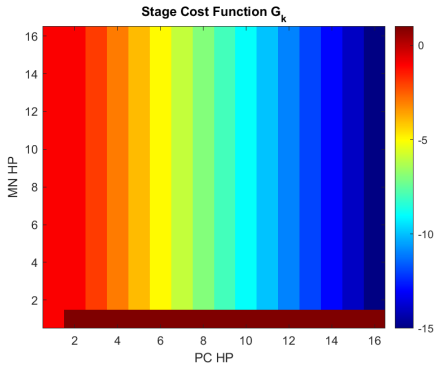

Fig. 1. Stage Cost of G_k.

B. Simulation Visualization

The optimal policy is tested using a simulation based on the nonlinear update equation (1). A visualizer was created to show the change of the PC and monster's states over the course of the combat. Fig. 2 below shows an example of the visualizer animation sequence. The HP values of the monster and PC are plotted in Fig. 3.

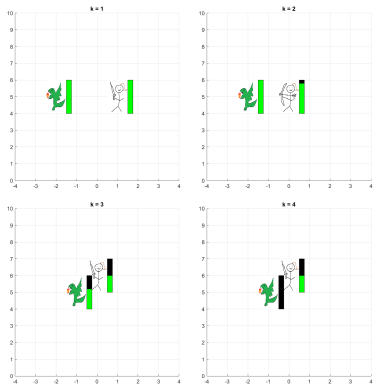

Fig. 2. A combat scenario in which the PC defeats the monster in 4 time steps

C. Monte Carlo Simulations

This process was repeated 1000 times in a Monte Carlo simulation in order to fully realize the results of the dynamic program. The Monte Carlo simulation results differed depending on what computer processor was used to calculate the Markov decision process.

1) Intel Results: Using an Intel i7-8750H (8th gen) processor, the win-rate of the PC was about 40%, as shown in Fig. 2. Using an AMD Ryzen 9 3950X, the win-rate of the PC was closer to 70%, as shown in Fig. 5. This might be due to how the processors differ in their methods of calculating random variables, as these simulations were performed using the exact same code and random number seed.

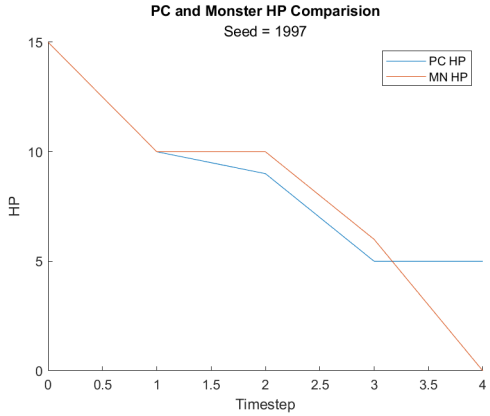

Fig. 3. A plot of the HP values of both actors from the combat scenario shown in Fig. 1

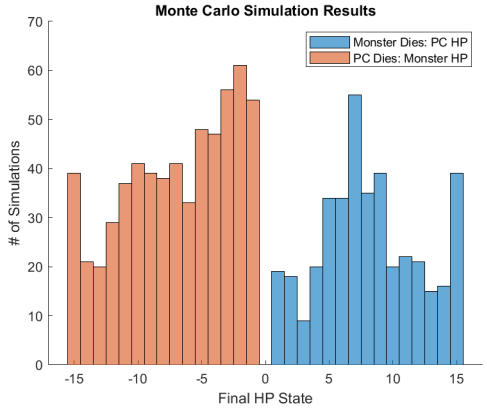

Fig. 4. Monte Carlo simulation performed on an Intel processor.

2) AMD Results: Figs. 4 and 5 depict scenarios where one of the actors dies. In some instances, both the PC and monster survive and the simulation concludes without a decisive outcome. This outcome could be attributed to the random positioning of the actors on the battlefield, wherein they may be too distant from each other, causing them to spend time running towards or away from each other instead of attacking. Alternatively, it could result from a sequence of unfavorable dice rolls, resulting in both parties failing to land successful hits.

D. Human Comparison

The Monte Carlo simulation results were compared to human decisions in the context of D&D gameplay, acknowledging that the optimal choice is not always made by human players. In the human simulation, the PCs' actions were manually inputted while playing against the monster, and the findings are presented in Fig. 6. The results indicate that the human performance was superior to that of the Intel processor, achieving a 60% win-rate; however, it fell short of the performance of the AMD processor.

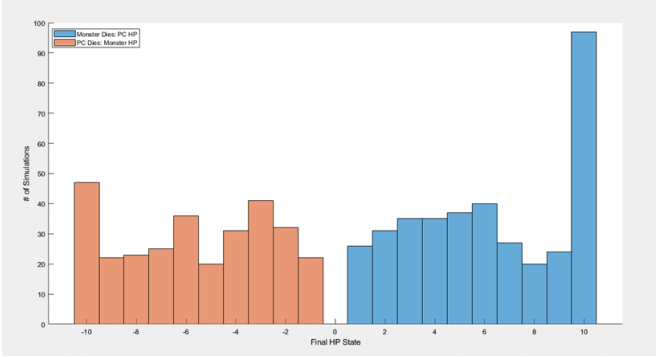

Fig. 5. Monte Carlo simulation performed on an AMD processor.

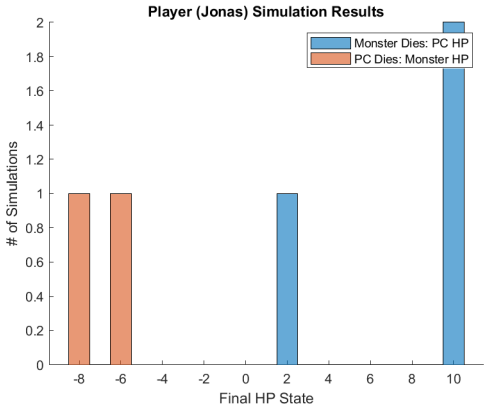

Fig. 6. Human combat results over 5 simulations.

REFERENCES

[1] W. R. Team, *Player's Handbook*, 5th ed. Wizards of the Coast, 2014. [Online]. Available: https://dnd.wizards.com/products/tabletop-games/rpg-products/dd-core-rules-gift-set

[2] "Wizards of the coast - dungeons & dragons." [Online]. Available: http://dnd.wizards.com/

[3] "Dnd 5e wiki." [Online]. Available: http://dnd5e.wikidot.com/

APPENDIX

The code for this project can be found at `https://github.com/jonaswagner2826/MECH6326_FinalProject/tree/master/SimulationProgramming`