A Policy Optimization of Greed

Chance Addis

Abstract—**The game of Greed is a stochastic, zero-sum game that tests a player's risk-taking, similarly to blackjack. We analyze Greed through the lens of game theory by modeling it as a Markov Decision Process (MDP). This framework allows us to compute an optimal policy that maps each game state to the action that maximizes a player's expected chance of winning. In addition to presenting the theoretical formulation, we explore efficient algorithms for computing this policy. By leveraging dynamic programming, state pruning, and compact policy representations, we substantially reduce computational overhead and enable fast, scalable analysis across large state spaces.**

1 Introduction

Games of chance and choice have long fascinated both players and theorists. From the strategic depth of poker to the probabilistic tension of blackjack, such games offer fertile ground for mathematical and algorithmic analysis. In this work, we turn our attention to Greed —a deceptively simple dice game with rich strategic structure.

Greed blends elements of push-your-luck decision-making and competitive scoring. At every step, players must weigh their chances, choosing how much they are willing to risk a bust, or worse, being overtaken in the last round.

In this paper, we formalize Greed as a Markov Decision Process (MDP) and compute its optimal policy (strategy) using dynamic programming. We begin by precisely stating the rules of the game, then model it as an MDP by defining its states, actions, and transitions. Next, we develop an efficient function for computing the probability mass function (PMF) of dice sums, a core component of the game. With this foundation, we compute the optimal policy and introduce performance optimizations to handle the game's large state space. Finally, we analyze the resulting strategy and discuss implications and extensions.

2 Rules

In order to play Greed, first the players must agree on a ruleset.

Definition 2.1 (Ruleset) A *ruleset* of Greed is a pair (M, s) where:
- $M \in \mathbb{N}^+$ is the *maximum score*, and
- $s \in \mathbb{N}^+$ is the *number of sides per die.*

With the ruleset defined, the game can begin.

Definition 2.2 (Greed) Let the ruleset for this game be (M, s).

The game is played between two players, **Alice** and **Blair**. Each player begins with an initial score of 0. Let $a, b \in \mathbb{N}_0$ denote the scores of Alice and Blair, respectively.

A starting player is chosen, typically by having both players roll a die and selecting the player with the higher result (re-rolling in case of a tie).

On a player's turn (e.g., Alice's), they choose a number $n \in \mathbb{N}_0$ of s-sided dice to roll:
- If $n > 0$, then n dice are rolled, and the resulting sum is added to the player's score. If the score exceeds M, the player *busts* and immediately loses.
- If $n = 0$, the player *stands*, triggering the *final round*: the opponent (Blair) takes exactly one final turn under the same rules (choosing any $n \in \mathbb{N}_0$).

The game ends when a player busts or when both players have stood. The result is determined as follows:
- If one player busts, the other wins.
- If both players stand, the winner is the player with the higher score.
- If both players have equal scores, the game is a tie.

The outcome can be represented as a zero-sum outcome function $O : \{\text{Alice}, \text{Blair}\} \to \{-1, 0, 1\}$, given by:

	Alice's Payoff	Blair's Payoff
Alice wins	$+1$	-1
Blair wins	-1	$+1$
Tie	0	0

3 Mathematical Framework

Definition 3.1 (State) A *state* S of Greed is defined by the 4-tuple $(a,\ b,\ t,\ l)$, where $a \leq M$ is Alice's score, $b \leq M$ is Blair's score, $t \in \{\text{Alice}, \text{Blair}\}$ indicates whose turn it is, and $l \in \{\text{true}, \text{false}\}$ is a boolean that indicates whether the game is in the final round.

For the purposes of computing optimal moves, the 3-tuple $(a,\ q,\ l)$ is equivalent to the 4-tuple $(a,\ b,\ t,\ l)$. The difference between the two is that the 3-tuple does not include the turn indicator t, which is not relevant for computing optimal moves. a is just the score of the player whose turn it is, and q is just the score of the player whose turn is up next. Therefore, from now on, we will use this 3-tuple to represent the state, indicating whose turn it is.

Example. Suppose Alice is up, and has a score of 85. Blair is up next, with a score of 70. It is not the final round. Then the state is $(85, 70, \text{false})$. This is equivalent to the 4-tuple $(85, 70, \text{Alice}, \text{false})$.

In addition to a state, it's also necessary to model the action space A of Greed.

Definition 3.2 (Action) An *action* is a move that a player can make in a given state. This action is simply some n number of dice to roll. $n = 0$ corresponds to *standing* and $n \geq 1$ corresponds to *rolling*.

Lastly, we define a transition function, which models the probability of transitioning from one state to another.

Definition 3.3 (Transition Function) The *transition function* for greed is defined as $f : S \times A \to [0, 1]$, mapping a state $s \in S$ and action $a \in A$ to the probability of transitioning from one state to another, given s, a.

For states $s = (a, q, l)$ and $s' = (a', q', l')$ in S, and action $a \in A$, the transition probability $f(s, s', a)$ is

$$\begin{cases} 1.0 & \text{if } s' = (q, a, \mathrm{T}) \wedge a = 0 \\ \boldsymbol{p}_n(a' - a) & \text{if } s' = (q, a', l) \wedge a \geq 0 \\ 0.0 & \text{otherwise} \end{cases}$$

where $\boldsymbol{p}_n(k)$ denotes the probability that the sum of n dice equals k.

Notice that many possible invalid states exist but have probability 0.0 either because $\boldsymbol{p}_n = 0.0$ (e.g., because $k < 0$), or because the transition violates some other rule like $l = \mathrm{T}, l' = \mathrm{F}$ or $a' \neq q$.

A game of greed is defined by its transition function $f : S \times A \to [0, 1]$. This is a result of the nature of greed as a Markov Decision Process (MDP), which is also defined by the transition function. Because Greed is a MDP, it inherits all the properties of an MDP, including its memoryless property, and therefore its ability to be solved via dynamic programming, which we'll prove and apply in Section 5.

4 PMF

Computing the probability mass function (pmf) of dice sums is essential for modeling Greed, as the game's scoring depends on the distribution of outcomes from rolling multiple dice. Specifically, we require the pmf of the sum of n independent and identically distributed (i.i.d.) discrete uniform random variables on the set $\{1, 2, ..., s\}$—that is, the total when rolling n fair s -sided dice. A closed-form expression for this distribution is known [1]:

Theorem 4.1 (PMF of dice sum) Let $\boldsymbol{p}(t \mid n, s)$ denote the probability that the sum of n i.i.d. discrete uniform random variables on the set $\{1, 2, ..., s\}$ equals t. Then:

$$\boldsymbol{p}(t \mid n, s) = \frac{1}{s^n} \sum_{k=0}^{\lfloor \frac{t-n}{s} \rfloor} (-1)^k \binom{n}{k} \binom{t - sk - 1}{n - 1}$$

This formula is exact, but computationally expensive for large n and s due to the growth of binomial coefficients. In practice, we use the Fast Fourier Transform (FFT) to compute the pmf more efficiently via convolution.

For a single die:

$$\boldsymbol{p}(t \mid 1, s) = \begin{cases} \frac{1}{s} & \text{if } 1 \leq t \leq s \\ 0 & \text{otherwise} \end{cases}$$

For $n > 1$ dice, we compute recursively via convolution:

$$\boldsymbol{p}(t \mid n, s) = \sum_{k=1}^{s} \boldsymbol{p}(t - k \mid n - 1, s) \cdot \boldsymbol{p}(k \mid 1, s).$$

5 Policy Solver

It's worth expanding on the concept of a payoff. In the rules, the payoff function occurs only at the conclusion of the game. We generalize the payoff function to include intermediate states, allowing us to optimize the expected payoff at each step.

Remark

Importantly, the payoff is not a probability—it does not sum to 1 (it sums to 0, as this is a zero-sum game).

Definition 5.1 (Payoff) The *payoff* is a mapping $V : S \to \mathbb{R}$, where S is the set of all game states. For terminal states, this function is specified directly by the rules of the game (denoted O).

This value reflects how favorable a state is for the active player:

- $V(s) = 1$: the active player has (or is guaranteed to) win;
- $V(s) = -1$: the active player has (or is guaranteed to) lose;
- $V(s) = 0$: the game is balanced or results in a tie.

To optimize Greed, we must consider not only the value of a state but also the value of actions taken from that state.

Definition 5.2 (Action-Value) The *action-value* function is a mapping $Q : S \times A \to \mathbb{R}$, where $Q(s, a)$ represents the expected payoff of taking action a in state s, and then following some policy π thereafter.

In particular: $Q_\pi(s, a)$ is the expected payoff under a specific policy π.

Definition 5.3 (Policy) A *policy* is a function $\pi : S \to A$, which selects an action for every state. It defines the strategy followed by the player.

Definition 5.4 (Optimal Policy) The *optimal policy* $\pi_\star$ is one that maximizes the expected payoff for the active player. Formally, for all states $s \in S$,

$$\pi_\star(s) := \arg\max_{a \in A} Q_{\pi_\star}(s, a)$$

To find this optimal policy, we use **minimax via dynamic programming**. This approach calculates values from the end of the game backward to the initial state, ensuring optimal decisions at each step.

Practically, this means we memoize the results of previously computed states to avoid redundant calculations. On an implementation level, instead we simply evaluate states in reverse order, using the structure of the game to build up the full value and policy functions.

5.A Terminal States

For terminal states, the problem is simple: find some n that maximizes the probability that your sum t will yield $a + t \in [q, M]$. More precisely, for a game state (a, q, T) we optimize the expected payoff

$$n_\star := \max_{n \in [0,\infty)} \sum_{t=n}^{sn} \begin{cases} 1 & \text{if } q < a + t \leq M \\ 0 & \text{if } a + t = q \\ -1 & \text{otherwise} \end{cases}$$

Of course, it's impossible to test every possible n in practice. Luckily, there is an upper bound on the optimal n that can be derived from the game's parameters. Specifically, we can show that

Proposition 5.1.1 (Upper Bound on Actions) In a state (a, q, l), the optimal number of dice to roll satisfies

$$n_\star \leq \left\lceil \frac{2(M - a)}{s + 1} \right\rceil := n_{\max}$$

Proof. Note that the expected value of n die with s sides is $\frac{n(s+1)}{2}$. Thus $n_{\max}$ is the point where the mean of n exceeds the max score.

The pmf of the sum of the dice is unimodal. Consider any following state where $a + t \leq M$. For any two n such that $n_{\max} \leq n_1 < n_2$, the probability of being at that score is strictly decreasing between n_1 and n_2. This occurs at every score that yields a positive payoff. Therefore, the sum is strictly decreasing between n_1 and n_2.

This means that for any $n \geq n_{\max}$, the payoff will monotonically decrease. Therefore $n_\star$ must be less than or equal to $n_{\max}$. □

This is a reasonable upper bound. But we can actually do better. Running simulations on a wide set of n, we find that the payoff function is unimodal with respect to n. This means that once the payoff starts decreasing, we can stop testing further n and take the maximum.

———

It's possible to leverage previous knowledge to narrow the score of n to test even more.

Let $s = (a, q, T), s' = (a, q + t, T)$ where $t > 0$. Let $n_\star$ be the optimal number of dice to roll for state s, and $n'_\star$ be the optimal number of dice to roll for state s'. It's guaranteed that $n'_\star \leq n_\star$. Similarly, for $s = (a, q, T), s' = (a - t, q, T)$, it's also true that $n'_\star \leq n_\star$.

Exploiting this property, we can strategically start from $(M, 0, T)$ and evaluate each row $(M, 0, T), (M, 1, T), ..., (M, M, T)$, using the previous $n_\star$ as the starting point. This then repeats for each column as well.

5.B Normal States

For normal states, optimization is more complex, as it becomes necessary to consider future states when optimizing the policy. As already mentioned, the way to accomplish this task is to use dynamic programming, starting from high scores and going backwards.

Consider the max score (M, M, F). In this state, the active player is forced to roll 0 dice, or otherwise lose. Thus the *only* potentially non-negative payoff is to roll 0 dice; this is the optimal policy. The payoff for the opponent is whatever the payoff of the terminal state (M, M, T). Since this is a zero-sum game, our score is the negative of the opponent's score.

Now consider two states $(M - 1, M, F), (M, M - 1, F)$. In these states, the only possible next states are busts, (M, M, F), or terminal states. Therefore, the payoffs of the next states are all previously computed. Thus the optimal policy for a state $s = (a, q, F)$ can be computed by

$$n_\star := \max_{n \in \mathbb{N}} \sum_{t=n}^{s} \boldsymbol{p}(t \mid n, s) \cdot Q(s, a)$$

This continues until $(0, 0, F)$, at which point all normal states have been computed.

Like with terminal states, it's not possible to try all possible n, but luckily Proposition 5.1.1 works for normal states as well. There is no massive tricks here *as of yet*, but it is possible to compute certain states (e.g., $(M - 1, M, F), (M, M - 1, F)$) in parallel since they will never overlap.

6 Optimal Policy Analysis

With the theoretical framework established, we implemented the dynamic programming algorithm to compute optimal policies for Greed across various rulesets. Our analysis reveals fascinating strategic patterns that emerge from the interplay between risk and reward in this deceptively simple game.

The following visualizations present the computed optimal policies and their associated payoffs for both terminal and normal game states. These visualizations provide many insights into the strategic principles of Greed and what those principles say about the game itself.

6.A A Game of Chicken

Looking at Figure 1, it's clear that you should absolutely not stop rolling dice if you are not already very close

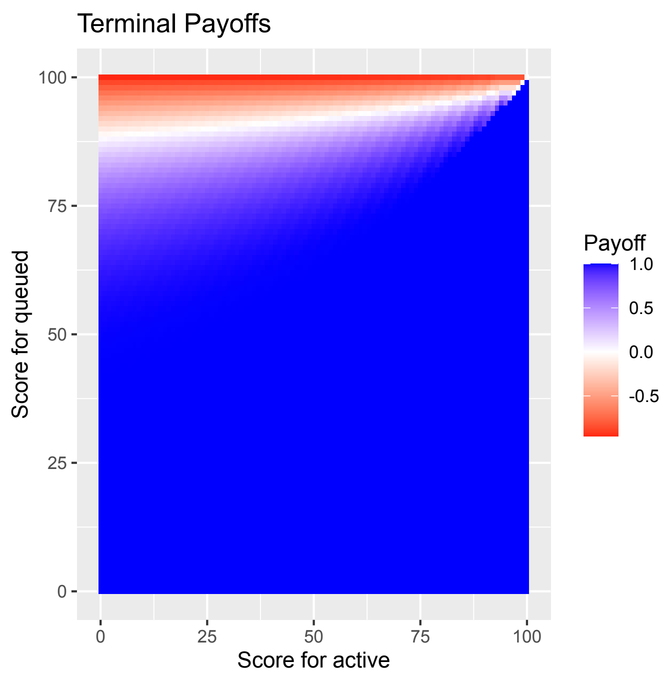

Figure 1: Optimal payoffs for terminal states.

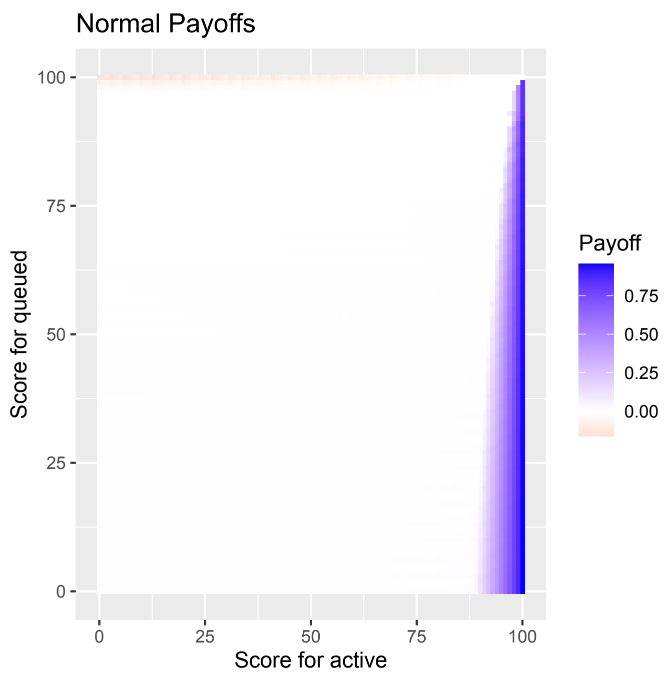

Figure 2: Optimal payoffs for normal states.

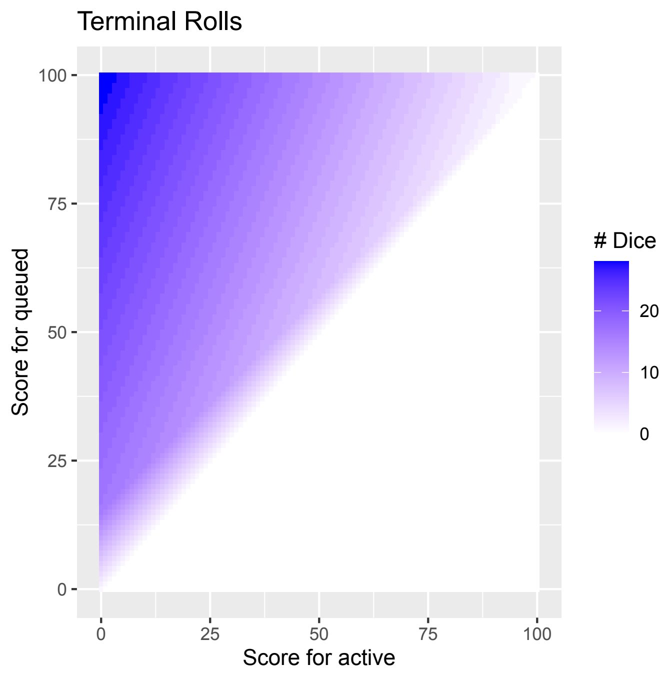

Figure 3: Optimal die to throw for terminal states.

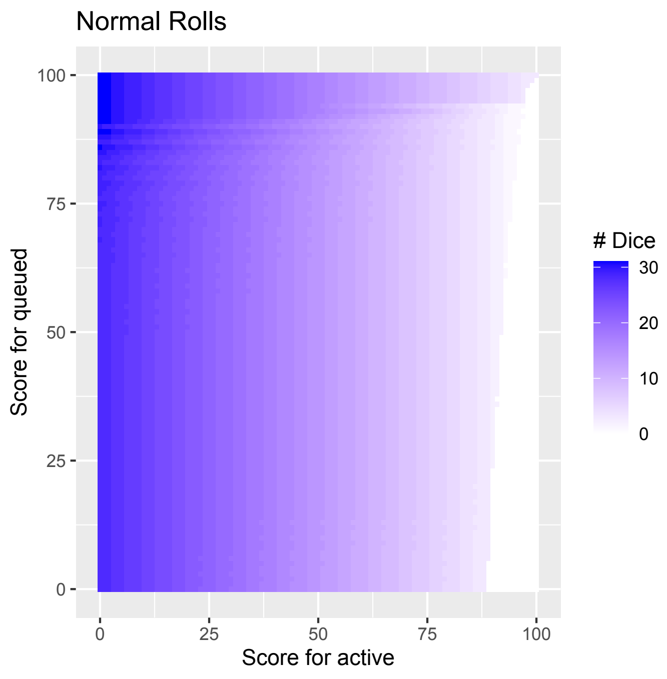

Figure 4: Optimal die to throw for normal states.

to the max score, because otherwise the opponent can easily catch up and win. The most notable feature is the band of balanced endgames (the white band), which follows a square-root function. Relating the terminal payoffs to the normal states, notice that the (bid) red area over the white band is the inverse of the (good) blue area for the normal payoffs. This is expected, since normal states in the positive region will lead the opponent to the corresponding negative region in the terminal states.

Another detail worth mentioning is the 4 white tiles at $(100, 100, F), (99, 99, F), \ldots, (96, 96, F)$ These states are ties, because rolling even one die will result in a bust at least half the time.

6.B A Balanced Game

Looking instead at the normal states, notice that the payoffs for normal states are very balanced. Unless you are in an extreme position, the game is a deadlock. Adjacent to all this, notice that the blue band for normal states corresponds to the white band on the normal rolls, signifying that when you have even a little advantage, you should aim to end the game on that turn. This implies that the game is very endgame focused, and the aim is to gain an advantage and end the game is fast as possible.

6.C Optimal Die Oddities

From an optimal die perspective, we see linear gradients, which is intuitive. After all, the goal is to either get as close to the maximum (for normal states) or between the opponent and the max (for terminal states). What's interesting are the edges, which tend to have odd values. In the terminal states with high n, we get some bizarre artifacts, likely a result of compounding floating-point precision errors.

There are also some odd n values, jumping back and forth between neighboring n. This is in part caused by the interaction of the sides and fixed maximum. However, it is likely also in part floating-point error or even (maybe) implementation oversights.

7 Conclusion

Our analysis revealed that Greed is fundamentally a game of careful endgame positioning—while midgame states tend to be balanced, securing even a small advantage can be decisive if leveraged to end the game immediately. The visualizations of optimal policies highlighted interesting strategic patterns, including the critical square-root boundary between winning and losing terminal positions. These insights not only deepen our theoretical understanding of Greed but also provide practical guidance for optimal play.

Future work could explore extensions to multiplayer variants or investigate the impact of different scoring rules on optimal strategies. There is also, I suspect, more room for performance improvements. However, the current improvements are enough for the goals of this project.

Greed is ultimately a simple game, but one that illuminates the patterns and complexity hidden within its mechanics. Like Conway's Game of Life, it's a game of simple rules that lead to complex behavior.

References

[1] Analytics Check, "Deriving the Probability Distribution for the Sum of Many Dice Rolls." [Online]. Available: https://www.analyticscheck.net/posts/sums-dice-rolls

[2] "Source code for *A Policy Optimization of Greed*." [Online]. Available: https://github.com/approximately-equal/greed

[3] D. F. Anderson, T. Seppäläinen, and B. Valkó, *Introduction to Probability*. Cambridge: Cambridge University Press, 2018.

[4] A. K. Singh, R. J. Dalpatadu, and A. F. Lucas, "The Probability Distribution of the Sum of Several Dice: Slot Applications," *Gaming Research & Review Journal*.