# Advanced Dice Parser using Regular Expression

## Discord Bot Dice Commands used in Dungeons & Dragons 5th Edition

Shaffira Alya Mevia 13519083
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: alyameviia@gmail.com

***Abstract*—In the times of the COVID-19 pandemic, many Tabletop Roleplaying Game (TTRPG), specifically Dungeons & Dragons, players have moved into a popular chatting platform Discord to continue playing. One of the main aspects of any TTRPG is the dice. Discord's user-supported chatbots are able to parse specific dice commands to roll a dice virtually. This can be done using regular expressions.**

***Keywords—dice parser; string matching; regex; discord; chatbot; Dungeons & Dragons 5e;***

## I. Introduction

Written in the times of the COVID-19 pandemic, in order to keep ourselves and others safe, we must maintain a physical distance. Because of that, it is hard for people to meet each other at this time. Including players from one of the most popular Tabletop Top Roleplaying Game (TTRPG) Dungeons & Dragons. They also suffered the consequences because they are unable to meet each other to play when the entire premise of the game is to meet on a table and play on top of the table. Players started to get creative and continued playing Dungeons & Dragons or D&D online. One of the popular media to do so is via Discord because one of its prominent features is the support of user-built chatbots.

The main aspect of a Dungeons & Dragons session or any TTRPG is the dice, ranging from the common 6 sided dice to a 4 or 20 sided dice. In some cases, we also need to roll an "imaginative" 100 sided dice typically for a roll table or a way for the Dungeon Master to measure the success rate of the player's action. Commonly this is done with the user typing a message with the 'XdY' format or something similar to it. On the dice format before, X means how many dice are present while Y means how many sides does the dice have. For example, a '1d20', a common dice used in Dungeons & Dragons sessions, means one dice with 20 sides. From here the chatbot then parses the message and executes the rolls. One of the ways user input can be parsed is with Regular Expressions.

## II. Theoretical Framework

### A. Pattern Matching

The pattern matching algorithm is a collection of algorithms used to test an expression to determine if it has certains characteristics. Pattern matching is one of the fundamental steps in processing texts and/or images. Pattern matching has many applications such as search features in a text editor, search engines, or web applications, fingerprint analysis, bioinformatics, and many more.

Pattern matching algorithms can be divided into a couple of categories, single-pattern matching algorithms, infinite-set pattern matching algorithms, and many more. The Knuth-Morris-Pratt algorithm and Boyer-Moore algorithm can fall into either a single-pattern algorithm or infinite-set algorithm. Another pattern matching algorithm example is the naive-search algorithm by implementing the brute force algorithm.

There is also a finite-state-automaton search algorithm that is a search based on the deterministic finite automaton machine. Deterministic finite automaton is a machine that accepts or rejects an input by iterating all of the characters with each of the accepted characters will change the machine's state depending on the transition arrow that corresponds to the accepted character before. Deterministic means that there is only at maximum one transition arrow that the character can take to each state. One example of pattern matching or search using the finite-search-automaton is regular expression, where the regular expression's syntax each explains the transition arrow of the finite-search-automaton machine.

### B. Regular Expression (Regex)

Regular expression or sometimes referred to as regex or RegExp is a sequence of characters that specifies a search pattern. It can help you to match, locate, and manage a text which will now be referred to as strings. Formally, regex is a way for algebra to express languages that can be defined recursively.

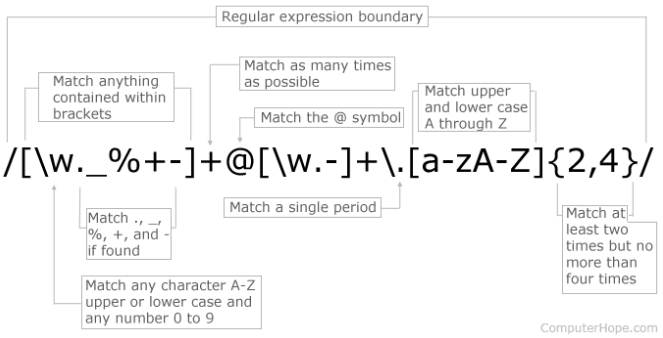


**Image 1 Regular Expression Structure (Source: [4])**

The regular expression algorithm uses regular language or a collection of strings that can be expressed with regular expression or can be processed by a deterministic-finite-automaton machine. Formally, regular language is from the alphabet Σ following a set of rules as follows recursively.

a. An empty language is notated by the Ø symbol. A language that can have an empty string that is notated by {ε} is regular language.
b. For every x, where x ∈ Σ or x is a part of Σ, a singleton language that is notated by {x} where this feature can only belong to a regular language.
c. If A and B is a regular language, then a union between A and B or can be notated with A ∪ B, a concatenation between A and B or can be notated with A • , and a Kleene star from A or A* are also defined as a regular language.A Kleene star from a string A is a combination of all iterations from the string A.
d. There are no other languages of Σ that are considered as a regular language.

Other than regular language, a regular expression has to also follow the regular grammar. A regular grammar is a mathematical object G with four main components N, Σ, P, S that can be notated as G = (N, Σ, P, S) according to the following.

- N is a non-terminal set symbol, where all other symbols that can be derived from this symbol are not empty.
- Σ is a terminal set symbol, where there are no other symbols that can be derived from this symbol. Another name for this set symbol is an alphabet.
- P is a set from derived rules and grammars, where all rules have a form such as below.
    - A → aB, where aB is derived from A and aB is a concatenation from a and B
    - A → a
    - A → ε, for A, B ∈ N
- S is the starting symbol or character

Regular expression uses special characters to match strings. The special characters can be classified as common tokens, general tokens, anchors, meta sequences, group constructs, character classes, flags or modifiers, and substitution. Below are some of the basic syntax used in regular expressions.

| Character | Matches |
|---|---|
| ^ | Matches beginning of string |
| $ | Matches end of string |
| . | Match any character |
| (...) | Capture anything matched |
| (?:...) | Non-capturing group |
| [...] | Matches anything contained |
| [^...] | Matches anything not contained |
| \ | Escape character |
| \d | Any digit character or [0-9] |
| \w | Any alphanumeric character or [A-Za-z0-9_] |
| \s | Any whitespace character |
| * | Zero or more occurrences |
| + | One or more occurrences |
| ? | Zero or one occurrence |
| {x} | x occurrences |
| {x,} | x or more occurrences |
| {x,y} | x to y occurrences |

Regex works by comparing the regex pattern to a targeted string. For example, we want to match ‘eggs’ from a string ‘I bought a dozen of eggs today’. We can use a regex ‘/eggs/’ to get the eggs.

Not only from the patterns, but a regex engine can also use flags to modify the search. For example, in ECMAScript (JavaScript) we can use the /g flag to instruct the engine to not stop after the first match has been found or the /i flag to match with case insensitivity. Some flags can be used in a regex engine.

### *C. Discord API with discord.js*

API or Application Programming Interface is a collection of routines, protocols, and tools used to build software applications. An API specifies how software components interact and are used when programming GUI components. For example, inside the operating system lies an API that allows us to copy and paste information in the form of text or images from one application to another that is not really flexible to one another. API’s initial usage was designed as a method to interact between one application to another

application. As the time passes, the use case for an API has slightly changed because of how programs and applications are now in different types of environments. A popular form of an API is RESTful API that can make all kinds of API flexible in all sorts of environments.

Discord provides developers APIs that are based around an HTTPS/REST API for general operations and a persistent secure WebSocket-based connection for sending and subscribing to real-time events. The most common use case is providing service or access to a platform through the OAuth2 API. While developers can directly access the Discord API, sometimes developers also use modules for easy interaction with the Discord API. Some popular API wrappers and modules include the discord.py for Python, discord.js for Node.js, JDA or DiscordPHP for Java. Inside the Discord Developer Portal, developers can create many applications that are not limited to just creating Discord bots. Another feature is for server moderators to check their server's insightful analytics about the community to check member engagement and server growth rate.

Discord.js module uses an object-oriented approach with predictable abstractions. Currently there are around 229 million downloads or users using this module with around 100 developers contributing on this active and growing project. At the time of writing, the latest version of discord.js is 12.5.3 that is updated around April 2021.

## III. Implementation

The dice parser is implemented inside an ongoing Discord bot project called SapicowBot.

### *A. Regex Search Pattern*

There are several regular expressions used in SapicowBot's dice parser. To understand the pattern, first, we must address the common dice format used in text-based dice. The most common ones are the 'XdY' and either 'XdYkhZ' or 'XdYklZ'. The first one means there are X amounts of Y-sided dices. While the second one means, from X amounts of Y-sided dices keep Z highest (kh) or lowest (kl) results. Example usage of the format is '1d20', meaning it is a 20 sided dice and '2d20kh1', meaning it will take 1 highest result from 2 dices with 20 sides. In addition to that, we can add modifiers to the dice roll's result. For example in an ongoing Dungeons & Dragons session, the player is asked by the Dungeon Master to roll a Perception check. The player has a +5 modifier for the Perception skill. So the player will roll a '1d20+5' dice. This can result in for example the '1d20' gives a 5 then the player's Perception check result is 5+5 or 10.

SapicowBot's dice parser is inspired by another Discord bot called Avrae. Below is a list of operators SapicowBot's dice parser can read.

- k, refers to keep
- p, refers to drop
- ro, refers to reroll once
- rr, refers to reroll infinitely
- mi/ma, refers to the minimum or maximum result
- e, refers to explode dice of value
- ra, refers to reroll and add

SapicowBot's dice parser consists of several main regular expressions that will be explained below.

1. Main Parser

```
/((adv|dis|advantage|disadvantage|disadv)|(
?!\d+)(?=#)\s*(.*$)|\s*(\d+\s*d\s*\d+)\s*((
(k|p|ro|rr|mi|ma|e|ra)(h|l|))\s*(\d+|)|)|(?
=([\+\-\*\/<>]*))\s*(\d+)|([\+\-\*\/<>]*))/
gi
```

Below is an explanation of the regex pattern above. Each alternative is placed in front of each other based on the pattern priority.

a. First Alternative

```
(adv|dis|advantage|disadvantage|disadv)
```

This pattern is used to handle advantage and disadvantage options when rolling the dice. For every alternative inside the capturing group will do an exact matching for the targeted string.

b. Second Alternative

```
(?!\d+)(?=#)\s*(.*$)
```

This pattern is used to handle comments that the user provides. The first part of the pattern or `(?!\d+)` is a negative lookahead. It will assert that the insides of the negative lookahead or `\d+`, matches a repeating digit with a minimum of one occurrence, does not match. This makes sure that the comment section is not part of the dice format.

The second part of the pattern or `(?=#)` is a positive lookahead, which means that it will assert the string with the occurrence `#`.

The third part of the pattern or `\s*` will match every whitespace character ranging between zero and unlimited. This is used to handle if the user types space between the `#` and the comment itself.

Lastly is the pattern `(.*$)` that will capture every character with the occurrences of zero to unlimited until the end of the string. An example string that can match is '# Insight Check'.

c. Third Alternative

```
\s*(\d+\s*d\s*\d+)\s*(((k|p|ro|rr|mi|ma|e|r
a)(h|l|))\s*(\d+|)|)
```

This pattern is used to handle the dice formats itself. Any `\s*` that appears on the regex pattern is used to handle any whitespace leaks that the bot might encounter. The first main part of the pattern or `(\d+\s*d\s*\d+)` is a capturing group that will match one or more occurrences of a digit character between a d character. For example, this pattern will handle strings such as '1d20', '2d20', or '4d4'.

The next main part of the pattern or `(((k|p|ro|rr|mi|ma|e|ra)(h|l|))\s*(\d+|)|)` is a capturing group consisting of several alternatives that are used to handle the operators that can be used. These operators are optional to be used. After the operators are present, a

number detailing the operator must be present for several operators hence the `(\d+|)` pattern.

d. Fourth Alternative

```
(?=([\+\-\*\/<>]*))\s*(\d+)
```

This pattern is used to handle modifiers that are used on the dice. The first part of the pattern or `(?=([\+\-\*\/<>]*))` is a positive lookahead which will assert that the string will have `[\+\-\*\/<>]*` before the modifier or `\d+` which is captured by a capturing group.

e. Fifth Alternative

```
([\+\-\*\/<>]*))
```

This pattern is used to capture any mathematical operators that appear on the string.

The main parser regex pattern uses a global modifier flag (/g) which matches all or does not return until the first match is found and a case insensitive flag (/i).

2. Operators and Selectors

```
/(?<=\d+)(d|(k|p|ro|rr|mi|ma|e|ra)(h|l|))(?=\d+)/gi
```

This regex will search for operators and selectors only. This is done by checking if there are numbers in front and behind the operators and selectors using a positive lookbehind `(?<=\d+)` and lookahead `(?=\d+)`.

The operators and selectors regex pattern uses a global modifier flag (/g) which matches all or does not return until the first match is found and a case insensitive flag (/i).

3. Mathematical Operators

```
/[\+\-\*\/<>]/g
```

This regex will handle any mathematical operators that appear on the string. The mathematical operators' regex pattern uses a global modifier flag (/g) which matches all or does not return until the first match is found

*B. Dice Parser Algorithm*

The dice parser works by splitting the string using the regex. Below is the pseudocode for the dice parser algorithm.

```
Function diceParser(str)
  // Main Parsing
  separated = match str with mainParserRegex
  filter separated from an undefined and empty string

  // Whitespace Checker
  whitespace = []
  If separated[0] has white space(s)
    For i = 0 to separated.length
      If separated[i] does not have '#'
        replace all white space with empty string
      Endif
    Endfor
  Endif
  If whitespace.length > 0
    separated = whitespace
  Endif

  // Splitting
  result = []
  For i = 0 to separated.length
    If separated[i] does not match mathOpsRegex and
matches operatorSelectorRegex
      char = []
      newArr = []
      arr = split separated[i] with operatorSelectorRegex
      filter arr from undefined and empty string
      For j = 0 to arr.length
        If arr[j] matches alphabetRegex
          push arr[j] to char
        Else
          push arr[j] to newArr
        Endif
      Endfor
      If char.length > 0
        push joined char with empty string delimiter to
newArr
      Endif
      push newArr to result
    Else
      If separated[0].length > 0
        push separated[0] to result
      Endif
    Endif
  Endfor
  filter result from empty array or empty string

  return result
Endfunction
```

**Fig. 1 Dice Parser Pseudocode**

Based on the pseudocode, the regexes are used both to match and split the string. The result will be an array already separated by operators and dice formats. For example, we will use the string below to demonstrate.

```
2d20+1d4+2d20kh1+4d20kl2+5+4d6mi2+2d6e6+10d6ra6+
4d6ro<3-1d20k1 adv # Insight check
```

The dice parser will start to parse based on the pseudocode and returns an array as follows.

```
[
   ["2", "20", "d"], "+",
   ["1", "4", "d"], "+",
   ["2", "20", "1", "dkhkh"], "+",
```

```
    ["4", "20", "2", "dklkl"], "+",
    "5", "+",
    ["4", "6", "2", "dmimi"], "+",
    ["2", "6", "6", "dee"], "+",
    ["10", "6", "6", "drara"], "+",
    ["4", "6", "droro"], "<",
    "3", "-",
    ["1", "20", "1", "dkk"],
    "adv", "# Insight check"
]
```

This result array will then be sent to the dice implementation algorithm. The dice implementation will read the result and starts to generate the random numbers according to the dice formats. After the numbers are generated, it will be sent back to the user via the Discord API.

### *C. Pattern Matching Results*

Below are some test cases used to test the regular expressions.

1. Normal roll

The first test is rolling a common dice used in a Dungeons & Dragons session, a 20 sided dice. Rolling a 20 sided dice can be caused by various actions but in this case the player is asked to roll a perception check. Based on the player's character's statistic, for a perception check it does not come with a modifier or a +0 modifier.

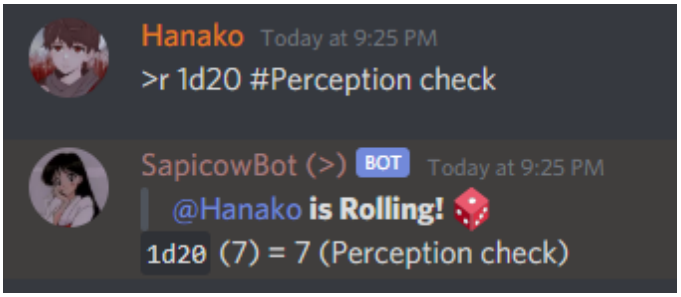


**Image 2 Normal perception check without a modifier**

The second test is rolling a high number of n-sided dice. Sometimes a 1d100 is used to roll a roll table. In Dungeons & Dragons 5th Edition's Wild Magic Sorcerer's subclass, players sometimes have to roll a Wild Magic surge to determine what effects take place when a Wild Magic surge occurs to the player's character.

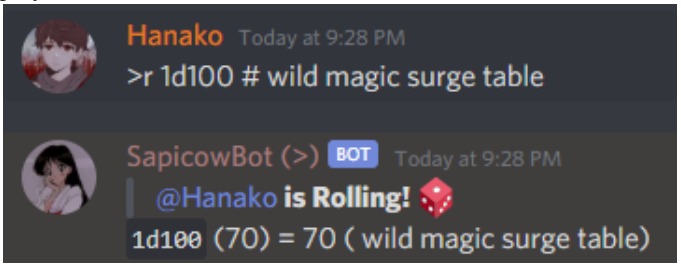


**Image 3 Wild Magic Surge Table**

2. Keep roll with modifier

Based on the player's character's statistics, some skills will have a modifier depending on the character's ability score. In this condition, the character encounters something that makes their character have an advantage on this skill check.

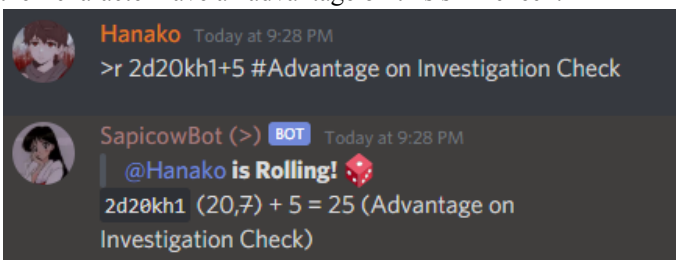


**Image 4 Keep high roll with modifier**

3. Random advanced roll

Although this is not a common use case for the typical Dungeons & Dragons session or any TTRPG sessions in general, the dice parser can still parse and print results from these random dice rolls with several mathematical operators included.

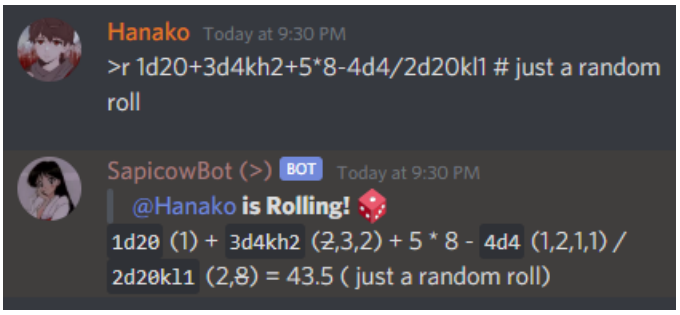


**Image 5 Random advanced roll**

## IV. Conclusion

Reasons why some Dungeons & Dragons players come to Discord to play is because of the support user-built chatbots. Some specific Dungeons & Dragons supported chatbots have the ability to roll dice which is something that is hard to do so when playing an online session if players were to use real dice. While there are multiple ways a dice can be parsed, SapicowBot's dice parser uses regular expressions. Based on the test cases, regular expression is one of the most powerful and efficient technologies for pattern matching, specifically in the case of parsing a dice format. Although the regular expressions used in this paper could be better improved, this is already enough to successfully parse an advanced dice format.

## Video Link at Youtube

A video demonstrating and explaining the dice parser can be accessed via this link (https://youtu.be/lcJkG-kuNI8).

## Acknowledgment

First, the author would like to thank God Almighty for giving the strength, knowledge, ability, and opportunity to finish this paper. The author would also like to thank the IF211 Strategi Algoritma team, specifically Dr. Nur Ulfa Maulidevi, ST, M.S, as the author's class lecturer. In addition, the author also thanks the Flumphs & Friends D&D 5e community on Discord for letting the permission to survey the server and Wonderers of Waterdeep community for bug testing and giving feedback to the dice parser feature.

## References

[1] Munir, Rinaldi. (2020). *Pencocokan String (String/ Pattern Matching)*. IF2211 Strategi Algoritma - Semester II Tahun 2020/2021. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

[2] BillWagner et al. (2021, April 26). *Pattern matching overview*. Microsoft Docs. https://docs.microsoft.com/en-us/dotnet/csharp/pattern-matching

[3] Computer Hope. (2020, December 31). *Regex*. Computer Hope. https://www.computerhope.com/jargon/r/regex.htm

[4] Beal, Vangie. (2021, April 9). *API (Application Program Interface) Meaning & Definition*. Webopedia. https://www.webopedia.com/definitions/api/

[5] Discord. (2021). *API Reference*. Discord Developer Portal. https://discord.com/developers/docs/reference

[6] DiscordJS. (2021). *Welcome | discord.js*. DiscordJS. https://discord.js.org/#/docs/main/stable/general/welcome

[7] d20. (2021). *Dice Syntax*. d20. https://d20.readthedocs.io/en/latest/start.html#dice-syntax

[8] Regex101. (2021). *Regular Expression*. Regex101. https://regex101.com/

[9] Team, W. R. (2014). *Player's Handbook (Dungeons & Dragons)* [E-book]. Wizards of the Coast.

[10] Sipser, Michael (1998). "Chapter 1: Regular Languages". Introduction to the Theory of Computation. PWS Publishing. pp. 31–90. ISBN 978-0-534-94728-6.

## PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 26 April 2021

Shaffira Alya Mevia
13519083