

Drop Chance Calculator

T1A3 - Terminal Application

Drop Chance Calculator

Glossary of Terms

- Kill count - This refers to the number of times a player has defeated a specific boss. This number is important to track as its relation to the odds of receiving certain items helps users gain insight into how lucky they are.
- Drop - A drop is a broad term used for items that are received from a boss. Similar words like loot or rewards could also be used.
- Drop rate - This is the probability of a boss dropping a specific item when defeated. While many games have many differing mechanics surrounding this certain assumptions have been made for drop rate calculations in this program. The fancier term being a series of Bernoulli trials. Often these rates can be given as a percentage or out 1/drop rate, e.g 5%, 0.05 or 1/20.

Features

- Calculate Drop Chance
- Simulate Drop Occurrences
- Store Boss Stats
- Display Boss Stats

```
1. Whats my chance of having a drop by now?  
2. Simulate drop attempts  
3. Store my boss stats  
4. Display my boss stats
```

Python Packages Used

- Termcolor - Allows text to display with colour and have bold styling! Even works for the tables from Pretty Table. Used in every feature

- Art - ASCII art to help with the titles

Drop Chance Calculator!

- Pretty Table - Allows lists/dicts to be shown in a table

Boss	Kill count	Total drops
Boss	100	5

Boss: Boss, Kill count: 100

Item dropped	Count	Drop rate
Item1	5	0.02

- TQDM - Progress bar for simulation

Simulated successful occurrences (1-10000000):

10000000

11%|



How to use

- Text based terminal application. User interacts by typing letters/numbers and pressing enter to navigate through the application
- Most features prompt the user to enter names of bosses or numbers in relation to their killcount, drop rate and drop occurrences.
- Typing exit at any point closes the application

Utility Features - safe_input

You, 1 second ago | 1 author (You)

```
from art import aprint
from termcolor import cprint
```

```
def safe_input(prompt, conversion_func=None):
    readable_error = {
        'int': 'numbers',
        'float': 'numbers',
        'str': 'text'
    }

    while True:
        user_input = input(prompt)
        if user_input.lower() == "exit":
            cprint("Thank you for using Drop Chance Calculator!", "cyan")
            aprint("sad and confused")
            exit()
        elif user_input.strip() == '':
            cprint("Invalid input! - Please enter something.", "yellow")
        elif conversion_func:
            try:
                converted = conversion_func(user_input)
                if isinstance(converted, int) or isinstance(converted, float):
                    if user_input.replace('.', '', 1).isdigit():
                        if converted <= 0:
                            raise ValueError(f"Invalid input! - Please enter positive {readable_error[conversion_func.__name__]}.")
                        return converted
                    else:
                        cprint(f"Invalid input! - Please enter positive {readable_error[conversion_func.__name__]}.", "yellow")
            except ValueError as exception:
                cprint(f"Invalid input! - Please enter positive {readable_error[conversion_func.__name__]}.", "yellow")
        else:
            return user_input
```

Chance Calculator

```
from termcolor import cprint
from inpututil import safe_input
```

You, 1 second ago | 1 author (You)

class DropCalculator:

```
    def __init__(self, main_menu):
        self.main_menu = main_menu
        self.boss_name = None
        self.drop_rate = None
        self.attempts = None
```

self.success_probability = None You, 1 second ago • Uncommitted changes

```
    def calculate(self):
```

```
        self.boss_name = safe_input("Enter boss name:\n")
        if self.boss_name is None:
            return
```

```
        self.drop_rate = safe_input("Enter drop rate (e.g. 0.01 or 100 for a 1/100 chance):\n", float)
        if self.drop_rate is None:
            return
        if float(self.drop_rate) > 1:
            self.drop_rate = 1 / float(self.drop_rate)
```

```
        self.attempts = safe_input("Enter number of attempts:\n", int)
        if self.attempts is None:
            return
```

```
        self.success_probability = 1 - (1 - self.drop_rate) ** self.attempts
        cprint(f'There is a {self.drop_rate * 100:.3f}% chance per kill to receive the drop you want from {self.boss_name}', 'light_green')
        cprint(f'After {self.attempts} attempts, you had a {self.success_probability * 100:.3f}% chance of being successful at least once.', 'light_green')
        safe_input("Press any key to return to main menu\n")
```

Drop Simulator

```
3  from termcolor import cprint
4  from inpututil import safe_input
5
6  You, 1 minute ago | 1 author (You)
7  class AttemptSimulator:
8      def __init__(self, main_menu):
9          self.menu = main_menu
10
11  def calculate(self):
12      boss_name = safe_input("Boss name:\n")
13
14      while True:
15          drop_rate = safe_input("Drop rate (e.g. 0.01 for 1/100 or 100 for 1/100):\n", float)
16          if drop_rate == 1:
17              cprint("Drop rate cannot be 1. Please enter a decimal or a whole number.", "red", attrs=['bold'])
18              continue
19          else:
20              if drop_rate > 1:
21                  drop_rate = 1 / drop_rate
22              break
23
24      while True:
25          simulated_successful_occurrences = safe_input("Simulating successful occurrences (1-100000000) - large numbers can cause the program to take some time to process(sub 1,000,000 is advised):\n", int)
26          if simulated_successful_occurrences <= 0:
27              cprint("Invalid input for simulated successful occurrences. Please enter a positive whole number.", "red", attrs=['bold'])
28          else:
29              break
30
31  attempts_per_success = []
32  You, 19 hours ago • reformatted folders for assessment requirements
33  for simulation in tqdm(range(simulated_successful_occurrences)):
34      attempts = 0
35      while random.random() >= drop_rate:
36          attempts += 1
37      attempts_per_success.append(attempts)
38
39  min_attempts = min(attempts_per_success)
40  max_attempts = max(attempts_per_success)
41  avg_attempts = sum(attempts_per_success) / len(attempts_per_success)
42
43  cprint(f"Program successfully got the drop {simulated_successful_occurrences} times from {boss_name}.", 'light_cyan')
44  cprint(f"The fewest attempts between drops was {min_attempts} and the most was {max_attempts}.", 'light_cyan')
45  cprint(f"Average attempts to be successful was: {avg_attempts:.3f}", 'light_cyan')
46  input("Enter any key to return to main menu\n")
47
```


Store Boss Kills

You, 2 seconds ago | 1 author (You)

```
import json
from boss import BossStore
from inpututil import safe_input
from termcolor import cprint

You, 2 seconds ago | 1 author (You)
class StoreBossStats:
    def __init__(self, MainMenu):
        self.main = MainMenu

    def execute(self):
        boss_name = safe_input("Boss name:\n")
        kill_attempts = int(safe_input("How many kill attempts do you have?\n"))
        items_dropped = {}
        while True:
            item_name = safe_input("Enter the name of the item dropped, or 'done' to finish:\n")
            if item_name.lower() == 'done':
                break
            item_drop_rate = float(safe_input("Enter the drop rate of the item:\n"))
            item_count = int(safe_input(f"How many times have you received {item_name}?\n"))
            items_dropped[item_name] = {'count': item_count, 'drop_rate': item_drop_rate}
        save = safe_input("Save? Y/N\n")
        if save.lower() == 'y':
            boss_instance = BossStore(boss_name, kill_attempts, items_dropped)
            self.main.boss_records[boss_name] = boss_instance
            with open('boss_stats.json', 'w') as file:
                json.dump({key: value.to_dict() for key, value in self.main.boss_records.items()}, file)
            cprint(f"Boss {boss_name} saved successfully.", "light_green")
        input("Enter any key to return to main menu\n")
```

You, 19 hours ago • reformatted folders for assessment requirements ...

Display Boss Kill Stats

```
You, 15 hours ago | 1 author (You)
1  from prettytable import PrettyTable
2  from termcolor import cprint
3
You, 15 hours ago | 1 author (You)
4  class BossStatDisplayer:
5      def __init__(self, boss_records):
6          self.boss_records = boss_records
7
8      def execute(self):
9          table = PrettyTable(["Boss", "Kill count", "Total drops"])
10         for boss in self.boss_records.values():
11             total_drops = sum(item['count'] for item in boss.items_dropped.values())
12             table.add_row([boss.name, boss.kill_attempts, total_drops])
13         cprint(table, "light_yellow")
14
15         for boss in self.boss_records.values():
16             cprint(f"\nBoss: {boss.name}, Kill count: {boss.kill_attempts}", "light_yellow", attrs=["bold"])
17             item_table = PrettyTable(["Item dropped", "Count", "Drop rate"])
18             for item, details in boss.items_dropped.items():
19                 item_table.add_row([item, details['count'], details['drop_rate']])
20             cprint(item_table, "light_yellow")
21
22         input("\nPress any key to return to the main menu\n")
23
```

Exception and Error Handling

```
def load_boss_records(self):
    try:
        You, 19 hours ago • reformatted folders for assessment requirments
        with open('boss_stats.json', 'r') as file:
            try:
                self.boss_records.update({key: BossStore.from_dict(value) for key, value in json.load(file).items()})
            except json.decoder.JSONDecodeError:
                cprint("Warning: boss_stats.json is empty or not properly formatted. Will create a new one now.", "red", attrs=['bold'])

    except FileNotFoundError:
        cprint("boss_stats.json not found. Creating a new file now.", "red", attrs=['bold'])
        with open('boss_stats.json', 'w') as file:
            pass

elif conversion_func:
    try:
        You, 16 hours ago • added better exception handling for negative numb...
        converted = conversion_func(user_input)
        if isinstance(converted, int) or isinstance(converted, float):
            if user_input.replace('.', '', 1).isdigit():
                if converted <= 0:
                    raise ValueError(f"Invalid input! - Please enter positive {readable_error[conversion_func.__name__]}.")
                return converted
            else:
                cprint(f"Invalid input! - Please enter positive {readable_error[conversion_func.__name__]}.", "yellow")
        except ValueError as exception:
            cprint(f"Invalid input! - Please enter positive {readable_error[conversion_func.__name__]}.", "yellow")
    else:
        return user_input

You, 3 hours ago | 1 author (You)
from mainmenu import MainMenu
from termcolor import cprint

def main():
    try:
        main_menu = MainMenu()
        main_menu.load_boss_records()

        while True:
            choice = main_menu.display()
            if choice == 1:
                main_menu.drop_calculator()
            elif choice == 2:
                main_menu.simulate_attempts()
            elif choice == 3:
                main_menu.store_boss_stats()
            elif choice == 4:
                main_menu.display_boss_stats()
            elif choice == 5:
                main_menu.exit_program()
            else:
                cprint('Invalid choice. Please enter a number from 1 to 5.', 'red', attrs=['bold'])

    except Exception as exception:
        print("An unexpected error occurred:", str(exception))

if __name__ == "__main__":
    main()
```

Challenges

- Conceptualising how to make the entire project. Was difficult to visualise how I wanted it to be set up and that became a process
- Classes: While I thought I understood classes, it was more difficult applying it to a new scenario
- Time management: Some things took substantially longer than I had planned due to errors or problems appearing that I couldn't seem to nail down easily

Ethical Issues

- Wasn't too many ethical issues as far as I could think of for this
- The program is very simple in what it does and I don't believe it could cause any harm or discomfort to people.
- Luck can invoke some strong emotions in people and displaying to people how 'unlucky' they are due to a dry spell may increase individuals frustration.

Favourite Moments of Development

- Getting my `safe_input` to work was really satisfying as it let me condense a lot of my error handling into one area
- Some problems took hours to solve but the feeling of overcoming them and finding a solution was really motivating!