

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

Main.py

main.py X

src > main.py > ...

You, 10 hours ago | 1 author (You)

```
1 from mainmenu import MainMenu
2 from termcolor import cprint
3
4 def main():
5     try:
6         main_menu = MainMenu()
7         main_menu.load_boss_records()
8
9         while True:
10             choice = main_menu.display()
11             if choice == 1:
12                 main_menu.drop_calculator()
13             elif choice == 2:
14                 main_menu.simulate_attempts()
15             elif choice == 3:
16                 main_menu.store_boss_stats()
17             elif choice == 4:
18                 main_menu.display_boss_stats()
19             elif choice == 5:
20                 main_menu.exit_program()
21             else:
22                 cprint('Invalid choice. Please enter a number from 1 to 5.', 'red', attrs=['bold'])
23
24         except Exception as exception:
25             print("An unexpected error occurred:", str(exception))
26
27 if __name__ == "__main__":
28     main()
```

MainMenu.py

```
class MainMenu:
    def __init__(self):
        self.boss_records = {}

    def display(self):
        tprint("Drop Chance Calculator!")
        cprint("1. Whats my chance of having a drop by now?", "light_green")
        cprint("2. Simulate drop attempts", "light_cyan")
        cprint("3. Store my boss stats", "light_blue")
        cprint("4. Display my boss stats", "light_yellow")
        cprint("5. Exit", "red")
        cprint("Type 'exit' at any time to exit the program.", 'dark_grey')
        return safe_input("What would you like to do? (enter number then press enter): \n", int)

    def drop_calculator(self):
        DropCalculator(self).calculate()

    def simulate_attempts(self):
        AttemptSimulator(self).calculate()

    def store_boss_stats(self):
        StoreBossStats(self).execute()

    def display_boss_stats(self):
        BossStatDisplayer(self.boss_records).execute()

    def exit_program(self):
        cprint("Thank you for using Drop Chance Calculator!" , "cyan")
        aprint("sad face")
        exit()

    def load_boss_records(self):
        try:
            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
```

Utility Features - safe_input

inpututil.py X

src > inpututil.py > ...

You, 36 minutes ago | 1 author (You)

```
1 from art import aprint
2 from termcolor import cprint
3
4 def safe_input(prompt, conversion_func=None):
5     readable_error = {
6         'int': 'numbers',
7         'float': 'numbers',
8         'str': 'text'
9     }
10
11     while True:
12         user_input = input(prompt)
13         if user_input.lower() == "exit":
14             cprint("Thank you for using Drop Chance Calculator!", "cyan")
15             aprint("sad and confused")
16             exit()
17         elif user_input.strip() == '':
18             cprint("Invalid input! - Please enter something.", "yellow")
19         elif conversion_func:
20             try:
21                 converted = conversion_func(user_input)
22                 if isinstance(converted, int) or isinstance(converted, float):
23                     if user_input.replace('.', '', 1).isdigit():
24                         if converted < 0:
25                             raise ValueError(f"Invalid input! - Please enter positive {readable_error[conversion_func.__name__]}.")
26                         return converted
27                     else:
28                         cprint(f"Invalid input! - Please enter positive {readable_error[conversion_func.__name__]}.", "yellow")
29             except ValueError as exception:
30                 cprint(f"Invalid input! - Please enter positive {readable_error[conversion_func.__name__]}.", "yellow")
31         else:
32             return user_input
33
```


Chance Calculator

dropcalculator.py

src > dropcalculator.py > ...

You, 6 hours ago | 1 author (You)

```
1 from termcolor import cprint
2 from inpututil import safe_input
3
4 class DropCalculator:
5     def __init__(self, main_menu):
6         self.main_menu = main_menu
7         self.boss_name = None
8         self.drop_rate = None
9         self.attempts = None
10        self.success_probability = None
11
12    def calculate(self):
13        self.boss_name = safe_input("Enter boss name:\n")
14        if self.boss_name is None:
15            return
16
17        self.drop_rate = safe_input("Enter drop rate (e.g. 0.01 or 100 for a 1/100 chance):\n", float)
18        if self.drop_rate is None:
19            return
20        if float(self.drop_rate) > 1:
21            self.drop_rate = 1 / float(self.drop_rate)
22
23        self.attempts = safe_input("Enter number of attempts:\n", int)
24        if self.attempts is None:
25            return
26
27        self.success_probability = 1 - (1 - self.drop_rate) ** self.attempts
28        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')
29        cprint(f'After {self.attempts} attempts, you had a {self.success_probability * 100:.3f}% chance of being successful at least once.', 'light_green')
30        safe_input("Press any key to return to main menu\n")
31
```

Drop Simulator

You, 6 hours ago | 1 author (You)

```
class AttemptSimulator:
    def __init__(self, main_menu):
        self.menu = main_menu

    def calculate(self):
        boss_name = safe_input("Boss name:\n")

        while True:
            drop_rate = safe_input("Drop rate (e.g. 0.01 for 1/100 or 100 for 1/100):\n", float)
            if drop_rate == 1:
                cprint("Drop rate cannot be 1. Please enter a decimal or a whole number.", "red", attrs=['bold'])
                continue
            else:
                if drop_rate > 1:
                    drop_rate = 1 / drop_rate
                break

        while True:
            simulated_successful_occurrences = safe_input("Simulating successful occurrences (1-100000000) - large numbers can cause the program to take some time\n")
            if simulated_successful_occurrences <= 0:
                cprint("Invalid input for simulated successful occurrences. Please enter a positive whole number.", "red", attrs=['bold'])
            else:
                break

        attempts_per_success = []
        for simulation in tqdm(range(simulated_successful_occurrences)):
            attempts = 0
            while random.random() >= drop_rate:
                attempts += 1
            attempts_per_success.append(attempts)

        min_attempts = min(attempts_per_success)
        max_attempts = max(attempts_per_success)
        avg_attempts = sum(attempts_per_success) / len(attempts_per_success)

        cprint(f"Program successfully got the drop {simulated_successful_occurrences} times from {boss_name}.", 'light_cyan')
        cprint(f"The fewest attempts between drops was {min_attempts} and the most was {max_attempts}.", 'light_cyan')
        cprint(f"Average attempts to be successful was: {avg_attempts:.3f}", 'light_cyan')
        input("Enter any key to return to main menu\n")
```

Store Boss Kills

```
You, 12 minutes ago | 1 author (You)
1 import json
2 from boss import BossStore
3 from inpututil import safe_input
4 from termcolor import cprint
You, 12 minutes ago | 1 author (You)
5 class StoreBossStats:
6     def __init__(self, MainMenu):
7         self.main = MainMenu
8
9     def execute(self):
10         boss_name = safe_input("Boss name:\n")
11         kill_attempts = (safe_input("How many kill attempts do you have?\n", int))
12         items_dropped = {}
13         while True:
14             item_name = safe_input("Enter the name of the item dropped, or 'done' to finish:\n")
15             if item_name.lower() == 'done':
16                 break
17             item_drop_rate = safe_input("Enter the drop rate of the item:\n", float)
18             item_count = safe_input(f"How many times have you received {item_name}?\n", int)
19             items_dropped[item_name] = {'count': item_count, 'drop_rate': item_drop_rate}
20         save = safe_input("Save? Y/N\n")
21         while save.lower() not in ['y', 'n']:
22             cprint("Invalid option. Please enter 'Y' to save or 'N' to cancel", "red")
23             save = safe_input("Save? Y/N\n")
24         if save.lower() == 'y':
25             boss_instance = BossStore(boss_name, kill_attempts, items_dropped)
26             self.main.boss_records[boss_name] = boss_instance
27             with open('boss_stats.json', 'w') as file:
28                 json.dump({key: value.to_dict() for key, value in self.main.boss_records.items()}, file)
29                 cprint(f"Boss {boss_name} saved successfully.", "light_green")
30             input("Enter any key to return to main menu\n")
31
```

You, yesterday • reformatted folders for assessment requirments ...

Display Boss Kill Stats

You, 22 hours ago | 1 author (You)

```
from prettytable import PrettyTable
from termcolor import cprint
```

You, 22 hours ago | 1 author (You)

```
class BossStatDisplayer:
    def __init__(self, boss_records):
        self.boss_records = boss_records

    def execute(self):
        table = PrettyTable(["Boss", "Kill count", "Total drops"])
        for boss in self.boss_records.values():
            total_drops = sum(item['count'] for item in boss.items_dropped.values())
            table.add_row([boss.name, boss.kill_attempts, total_drops])
        cprint(table, "light_yellow")

        for boss in self.boss_records.values():
            cprint(f"\nBoss: {boss.name}, Kill count: {boss.kill_attempts}", "light_yellow", attrs=["bold"])
            item_table = PrettyTable(["Item dropped", "Count", "Drop rate"])
            for item, details in boss.items_dropped.items():
                item_table.add_row([item, details['count'], details['drop_rate']])
            cprint(item_table, "light_yellow")

        input("\nPress any key to return to the main menu\n")
```

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!