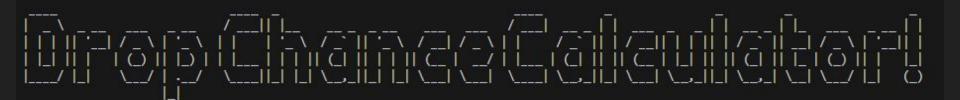
Drop Chance Calculator

T1A3 - Terminal Application



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

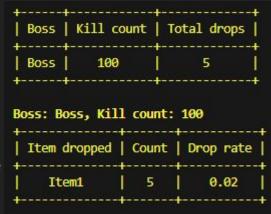
- Whats my chance of having a drop by now?
- Simulate drop attempts
- 3. Store my boss stats
- 4. Display my boss stats

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

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



TQDM - Progress bar for simulation

```
Simulated successful occurrences (1-100000000):
100000000
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

```
from art import aprint
from termcolor import cprint
def safe input(prompt, conversion func=None):
    readable error = {
        'int': 'numbers',
        'float': 'numbers'.
    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:
                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
                        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")
            return user input
```

Chance Calculator

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

```
from termcolor import corint
    from inpututil import safe input
6 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'])
                    if drop rate > 1:
                        drop_rate = 1 / drop_rate
           while True:
                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)
                if simulated successful occurrences <= 0:
                    cprint("Invalid input for simulated successful occurrences. Please enter a positive whole number.", "red", attrs=['bold'])
                   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, 2 seconds ago | 1 author (You)
import json
from boss import BossStore
from inpututil import safe input
from termcolor import cprint
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")
```

Display Boss Kill Stats

```
You, 15 hours ago | 1 author (You)
from prettytable import PrettyTable
from termcolor import cprint
You, 15 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):
    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:

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
```

```
from mainmenu import MainMenu
from termcolor import corint
def main():
        main menu = MainMenu()
        main menu.load boss records()
           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()
                cprint('Invalid choice. Please enter a number from 1 to 5.'. 'red', attrs=['bold'])
    except Exception as exception:
        print("An unexpected error occured:", 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!