



**Assignment Cover Letter
(Individual Work)**

Student Information :

| Surname | Given Names | Student ID Number |
|----------------|--------------------|--------------------------|
| 1. Mikael | Jayson | 2440032442 |

| | | |
|----------------------------|---|-----------------------------|
| Course Name | : | Program Design Methods |
| Name of Lecturer(s) | : | Jude Joseph Lamug Martinez |
| Course Code | : | COMP6056 |
| Class | : | L1AC |
| Major | : | Computer Science |
| Title of Assignment | : | Face Recognition Attendance |
| Type of Assignment | : | Final Project |

Submission Pattern :

| | | | | | |
|-----------------|---|-----------|------------------------|---|-----------|
| Due Date | : | 13-1-2021 | Submission Date | : | 13-1-2021 |
|-----------------|---|-----------|------------------------|---|-----------|

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer's instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
3. The above information is complete and legible.
4. Compiled pages are firmly stapled.
5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

Plagiarism/Cheating

BiNus International seriously regards all forms of plagiarism, cheating and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

Declaration of Originality

By signing this assignment, I understand, accept and consent to BiNus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signature of Student:

Jayson Mikael

Text-Based Python RPG

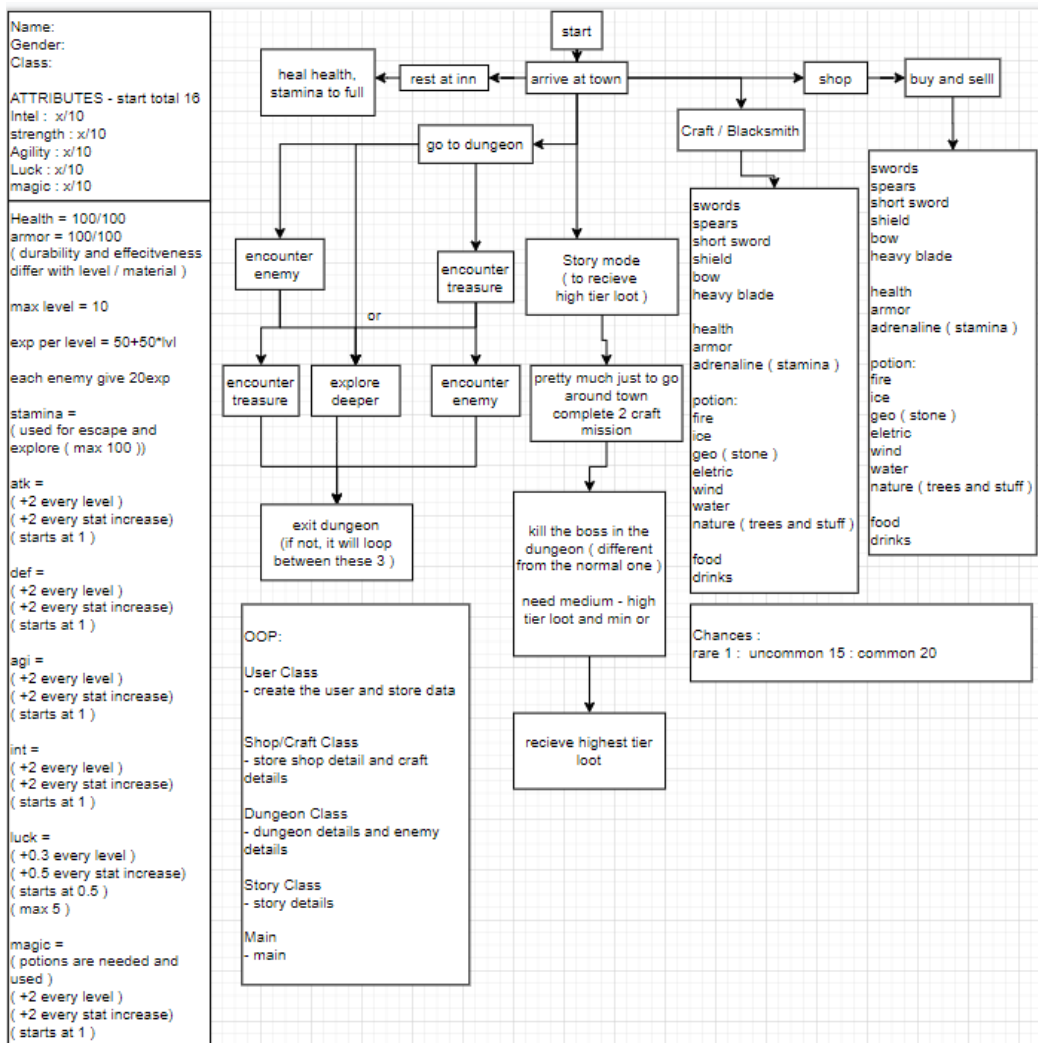
Name : Jayson Mikael

ID : 2440032442

I. Program Description

The program that I have created is using python as a base to create a Text-based RPG game. This is a simple game but requires specific input due to the constraints of python and/or general programming. The program is created using python only and libraries created by other users. This game is created to fill people's free time.

II. ERD



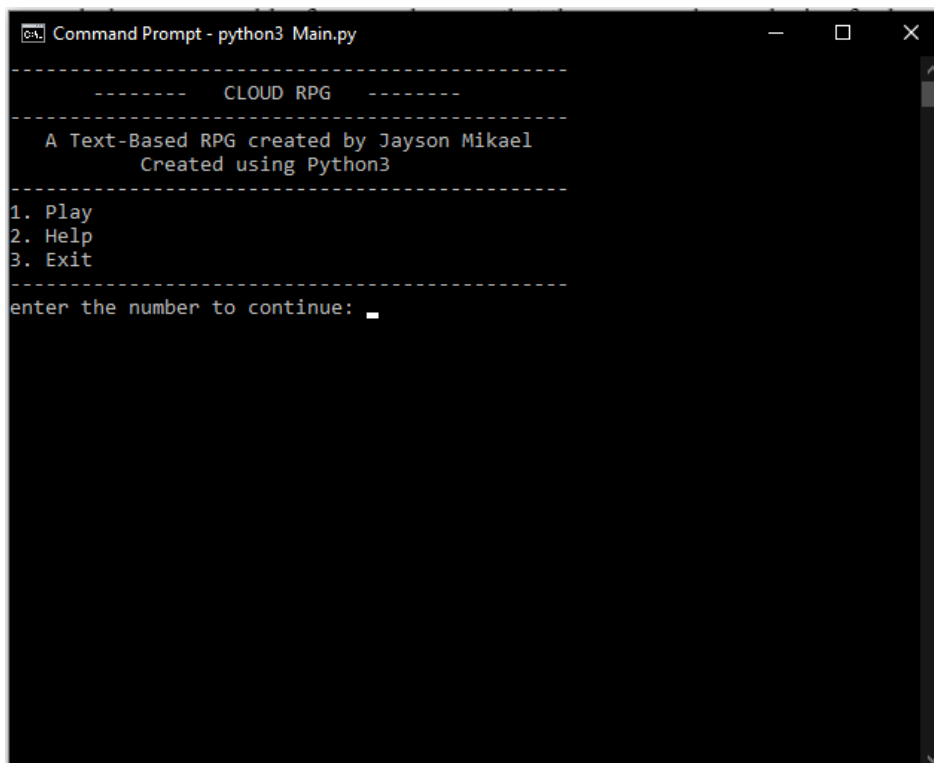
As can be seen above, the flowchart shows the flow of the game that can be taken by the user. However due to the complexity of the program and the limited time of alpha and beta testing for the program some parts of it had to be changed. I used OOP to simplify the program and to encapsulate different scenarios of the game and to make finding bugs easier. This allowed me to create a lot of things that can be done in the program.

Lots of them uses similar coding such as crafting and buying items. There is also some variables that are used a reference so that I don't have to call the item from a different class. The whole game would, of course, loop so that the user can keep playing for longer periods of time.

III. Game Interface

The game interface is rather simple and easy to read. Since it is a Text-based RPG, all of the visuals would be in the form of a text.

a. Main Menu



```
Command Prompt - python3 Main.py
-----
          CLOUD RPG
-----
A Text-Based RPG created by Jayson Mikael
Created using Python3
-----
1. Play
2. Help
3. Exit
-----
enter the number to continue: _
```

b. Help Menu

```
Command Prompt - python3 Main.py

----- CLOUD RPG -----
A Text-Based RPG created by Jayson Mikael
Created using Python3
-----
Follow each command in order to continue
usually by either inputing numbers or specific
words
This is created as i am unable to draw for the
sake of my life
-----
press 'enter' to continue
```

c. User Creation

```
Command Prompt - python3 Main.py

-----
User Created !
Name : Dex
Gender : Male
Attributes : {'INT': 2, 'STR': 4, 'AGI': 3, 'LCK': 3}
-----
Welcome to Cloud RPG !
-----
Currently Loading
-----
_

Command Prompt - python3 Main.py

Please Create a name = Dex
Pick a gender. ( Female or Male )
( write exactly as shown ): Male
-----
You currently have 12 points to spend on attributes
{'INT': 0, 'STR': 0, 'AGI': 0, 'LCK': 0}
Pick an Attribute to spend Points on: _
```

d. Starting Story Part 1

```
Command Prompt - python3 Main.py
-----
current location: Forest
you are currently running away from what it
seems to be a bear.
you're tired and your sword and armor has taken
some damage.
In front, you see a town with guards on the entrance.
-----
your current choices
1. run towards the town and ask help from the guard
2. fight the bear
```

e. Starting Story Part 2.1 (end)

```
Command Prompt
-----
current location: Forest
you are currently running away from what it
seems to be a bear.
you're tired and your sword and armor has taken
some damage.
In front, you see a town with guards on the entrance.
-----
your current choices
1. run towards the town and ask help from the guard
2. fight the bear
2
-----
you decided to fight the bear. the bear was very strong
but you still had some fight left inside. you fought the bear,
yet the bear was overwhelming you. in the end, you werent able to
defeat the bear, and you died.
-----
Game Over
```

f. Starting Story Part 2.2 (continue)

```
Command Prompt - python3 Main.py
-----
You ran straight to the entrance and the guards
sees the bear and helps you.
The bear decided to ran away after seeing the guards
You are now safe
Guard1 = 'What is your name, adventurer ? '
Dex = my name is Dex.
guard2 = 'You might want to head inside and take a break.'
you head inside...
Inside you saw a shop, the blacksmith and the inn.
-----
Where will you go ?
1. Shop
2. Blacksmith ( craft )
3. Inn
-----
Input the number :
```

g. Inn

```
You decided to take a sleep at the nearby inn.  
Health is healed to full  
Stamina and Mana are restored  
-----  
Cost : 2 gold coins  
Balance : 48  
-----
```

h. Inventory management

```
Command Prompt - python3 Main.py  
-----  
Name : Dex  
level : 1  
Exp : 0  
Currently equipped weapon = Sword  
Currently equipped armor = Leather Armor  
-----  
-----Inventory-----  
1. Sword - 1  
2. Shield - 1  
3. Leather Armor - 1  
4. Health Potion - 1  
5. Iron Ingots - 5  
6. Wood - 10  
7. Gold Coins - 48  
8. Food - 2  
9. Drink - 1  
-----  
What would you like to do ?  
1. Change weapon  
2. Change Armor  
3. Exit inn  
-----  
input the NUMBER to continue : _
```

i. Town

```
Command Prompt - python3 Main.py  
You are currently in the town.  
-----  
Where will you go ?  
0. Exit to Main Menu  
1. Shop  
2. Blacksmith ( craft )  
3. Inn  
4. Dungeon  
5. Recycle  
-----  
Input the number :
```

j. Shop

```
Command Prompt - python3 Main.py
-----Ethereal Fowl Shop-----
your current gold amount : 48
The shop is currently selling
item : price : availability
1. Sword : 7 : 10
2. Bow : 5 : 10
3. Shield : 4 : 10
4. Health Potion : 5 : 10
5. Iron Armor : 7 : 10
6. Food : 2 : 50
7. Drink : 2 : 50
-----
What would you like to purchase ?
input 'exit' to exit the shop
input the EXACT NAME in the list : _
```

k. Blacksmith

```
Command Prompt - python3 Main.py
-----Curious Forge-----
Current inventory:
Gold Coins : 48
Iron Ingots : 5
Wood : 10
-----
-----Part 1-----
Item : Cost : Iron : Wood
1. Sword : 3 : 2 : 1
2. Bow : 2 : 0 : 2
3. Shield : 2 : 3 : 2
4. Iron Armor : 2 : 3 : 0
-----Part 2-----
Item : Cost : Red Liquids : Yellow Liquid
5. Health Potion : 5 : 4 : 0
-----
What will you Craft ?
type 'exit' to exit the blacksmith
input the EXACT name of the item : _
```

l. Recycle Station

```
-----The Recycle Station-----
-----Inventory-----
1. Sword - 1
2. Shield - 1
3. Leather Armor - 1
4. Health Potion - 1
5. Iron Ingots - 5
6. Wood - 10
7. Gold Coins - 48
8. Food - 2
9. Drink - 1
-----
Current shop inventory:
{'Iron Ingots': 10, 'Wood': 10, 'Red Liquids': 5}
-----
Able to Recycle:
Sword | Bow | Iron Armor | Health Potion
What do you wish to recycle ?
type the EXACT name:
```

m. Dungeon

```
Command Prompt - python3 Main.py
You enter the dungeon
-----The Dungeon-----
would you like to search deeper ?
1. Yes
2. No
-----
Input number to continue : _
```

n. Battle

```
-----
Name : Dex
level : 1
Exp : 0
Currently equipped weapon = Sword
Currently equipped armor = Leather Armor
-----
Enemy Health = 100
Player Health = 100
-----
1. Attack
2. Defend
3. Heal
4. Flee
-----
input the number to continue :
```

IV. Game Functions / How to play

When starting the game, you will be greeted with the main menu allowing you to go to the help menu, play the game or the exit the program. Going to the help menu would show you how to play the game and who is the program created by. if the user starts the game, the user will be then instructed to create a character for the game. After inserting the name, gender and preferred attributes, the character would immediately start with a short story that will either end the game due to death or continue to go straight to the town. To continue the story, go to town. By doing this the story would continue and you would be in the safe hands of the town. Inside the town, there are several options that can be done such as going to the shop or blacksmith or go to the inn

to take a rest. If the user goes to the inn, they are also able to change inventory setting, such as change equipped weapon or armor.

After this initial storyline, the user would be given options to buy items, craft items, sleep, recycle items, or go to the guild. By going to the guild you would start a short yet long quest that would reward you in a weapon that is better than anything currently available. The game would then be done after the quest is done. The user can continue to go to the dungeons, buy items, and other things that is available.

In the dungeon, the battles are rather simple. When searching deeper, there would 3 different scenarios that would be possible, either you found a treasure chest, a rare treasure chest, or, more commonly, an enemy. When encountering an enemy, the battle is also made simple. You could attack, defend, heal, or flee. If the user were to attack the enemy, the program would use the “str” attribute of the user and the current equipped weapon to calculate the damage that would be applied to the enemy, however the enemy would also attack but deal a fixed amount of damage. If the user were to defend, then both enemy and user would not deal any damage. If the user were to heal, then 10hp would be restored. If the user were to flee, the “agi” and “lck” attribute of the user would be used to calculate whether the user is able to flee or not, but doing this would not reward you with resources and experience points.

In the shop, you are able to buy items in exchange of gold coins. Different items would cost differently. This part is rather finicky as the program demands specific inputs in order to continue. By inputting the exact words (capital letters included), the user is able to purchase different items. This would also be an alternative if the user would not like to craft. If the user would rather craft an item, the choices are, however, limited as the blacksmith could make so much. Recycling also another alternative to get materials for crafting. All 3 options require specific input, so be sure to read carefully.

V. Libraries used

There are some libraries used in order to simplify the code and to make the game work the way it is intended to, such as:

1. Time

Time is used mainly for `time.sleep` which allows the program to stop / to not read the next line of code for a specified amount of time.

2. Os

Os is used to clear the terminal in order to have a clean interface / clean look when changing scenarios.

3. Random

Random is mainly used in the `dungeon` class to create a sense of randomness especially when exploring the dungeon. It is also used for the “flee” mechanic so that if the players tries to flee there would a chance to fail.

VI. Code Explanation

The programs are created using Python 3 and using the OOP method. This means separating different parts of the code into 6 major parts. This allows easier debugging and for the reader to have an easier time to learn about the code.

a. Main.py

Main.py is the program to run if you wish to play the game. It combines all of the other classes to one so that the game could run.

```
def main():
    while True:
        draw.mainmenu()
        mainmenuchoice = int(input("enter the number to continue: "))
        if mainmenuchoice == 1:
            gameuser = user.createUser() #temporarily create a user to be used
            print("-----")
            print("Currently Loading")
            print("-----")
            time.sleep(4)
            draw.startstory()
            startstorychoice = int(input())
```

As can be seen above, the code starts by using “while True” so that the game would repeat if the user were to go back to the same scene. Draw.mainmenu() would draw the main menu visuals. The user would the input to the variable named mainmenuchoice. This would direct the flow of the game and the story. The user would then create the character in user.createUser(). time.sleep is used to stop the program for 4 seconds and allow the reader to read the created character. Draw.startstory would draw the visuals for the first part of the intro.

```
if startstorychoice == 1:
    draw.introstory1(gameuser.name)
    choice2 = int(input("Input the number : "))
    if choice2 == 1:
        town.shop(gameuser) #buy and sell items
    elif choice2 == 2:
        town.blacksmith(gameuser) #craft and recycle items
    elif choice2 == 3:
        town.sleep(gameuser) #heal and save the game to a file.

    else:
        print("ERROR: Unknown Input")
```

The next part is to direct the flow again to either the shop, blacksmith and the inn. This is constantly reused as it is main menu in the game after the intro.

```

while True:
    count += 1
    if count >= 2:
        draw.insidetown(True) #mainmenu when in the game
    else:
        draw.insidetown(False) #mainmenu when in the game
    choice3 = int(input("Input the number : "))
    if choice3 == 0:
        break
    elif choice3 == 1:
        town.shop(gameuser)
    elif choice3 == 2:
        town.blacksmith(gameuser)
    elif choice3 == 3:
        town.sleep(gameuser)
    elif choice3 == 4:
        dugn.dungeon(gameuser)
    elif choice3 == 5:
        town.recycle_sell(gameuser)
    elif choice3 == 6 and count >= 2:
        if gameuser.story == 0: str.part1(gameuser) # once executed, cant go back to here
        elif gameuser.story == 1: str.part2(gameuser)
        elif gameuser.story == 2: str.part3(gameuser)
    else:
        print("ERROR: Unknown Input")

```

This is the main menu of the game after the intro. It directs the program to the selected scene of the user.

b. Draw.py

Draw.py is mainly used to draw the visuals of the game that is outside of the dungeon, town, and story. This class mainly comprises of draw functions filled with print() function to draw the visuals of the menus and the game.

```

def mainmenu():
    os.system('cls')
    print("-----")
    print("          CLOUD RPG          ")
    print("-----")
    print("  A Text-Based RPG created by Jayson Mikael  ")
    print("          Created using Python3 ")
    print("-----")
    print("1. Play")
    print("2. Help")
    print("3. Exit")
    print("-----")

```

As can be seen above, this is the draw function to make the visuals for the main menu.

```
def insidetown(introdone):
    os.system('cls')
    print("You are currently in the town. ")
    print("-----")
    print("Where will you go ?")
    print("0. Exit to Main Menu")
    print("1. Shop")
    print("2. Blacksmith ( craft )")
    print("3. Inn")
    print("4. Dungeon")
    print("5. Recycle")
    if introdone == True:
        print("6. Adeventure's Guild ( story ) ")
    print("-----")
```

The code above shows the main menu after the intro. The variable intro done is a simple variable to show the story path or not. If its true then, the code would print the extra line and proceed.

c. User.py

User.py is used for user creation and to store the user's data. The user's data is saved using an object so that the data is collected in one place. The user object is then used for data retrieval and changes.

```
class User: #User class and account when playing.
    def __init__(self, name, gender, attr):
        self.name = name
        self.gender = gender
        self.attr = attr #attributes : Intelligence, Strength, Agility, and Luck
        self.Health = 50.0 #start health is half
        self.Armor = 0.25 #damage reduction ( change with armor )
        self.Level = 1
        self.EXP = 0 #for every 10 EXP = 1 level
        self.UserInventory = {"Sword":1, "Shield":1, "Leather Armor": 1, "Health Potion": 1, "Iron Ingots": 5, "Wood": 10, "Gold Coins": 50, "Food":2, "Drink":1}
        self.Equipped = "Sword"
        self.EqArmor = "Leather Armor"
        self.story = 0
        self.objective = 20
        self.strComplt = False
```

There is a lot of information stored in the object, such as name, gender, attributes, health, inventory and others. When the object is instantiated, several information is missing. The missing information is created by the user, these being name, gender and attributes. Several others are created already as they are stock configurations.

```

def createUser(): #UserCreation
    os.system('cls')
    name = str(input("Please Create a name = "))
    gender = str(input("Pick a gender. ( Female or Male )\n( write exactly as shown ): "))
    currentpoints = 12
    Attr = {"INT":0, "STR":0, "AGI":0, "LCK":0} #attributes helps in dungeon
    while True:
        if currentpoints <= 0:
            print("are you sure ? (Y/N)")
            makesure = str(input("input either Y or N: "))
            if makesure.lower() == "y":
                break
            elif makesure.lower() == "n":
                pass
        print("-----")
        print("You currently have", currentpoints, "points to spend on attributes")
        print(Attr)
        pick = str(input("Pick an Attribute to spend Points on: ")) #attribute_maker
        if pick.upper() not in Attr.keys():
            print("error input")
        else:
            spend = int(input("Change / Add to: "))
            if spend <= currentpoints:
                if Attr[pick.upper()] >= spend:
                    currentpoints = currentpoints + Attr[pick.upper()]
                    Attr[pick.upper()] = spend
                else:
                    Attr[pick.upper()] = Attr[pick.upper()] + spend
            else:
                print("not enough points")
            currentpoints = currentpoints - spend

```

To create the user, the code above is used. `Os.system('cls')` is used to clear the terminal so that previous draws would be deleted and new draws would not continue to draw below the old ones. Name and gender are rather simple, an `input()` function is used to gain user input for the name and gender. It is then inserted to a temporary variable with similar names. In order to have fluid attributes, the program would first give the user 12 points to allocate. A while loop is used so that the user is able to allocate until the available points (`currentpoints`) reaches 0. The inputs from the user are divided into 2. The first being “pick”. This is to pick the attribute to be changed. Then the program asks the user to input the amount, named “spend” to allocate into the chosen attribute. If the spend points are more than the available points then the program would error check and print out an error message. If it does have enough points, it would add it into the chosen attribute. If the spend is less than the allocated points in the attribute, it would change to the spend points inputted by the user. After allocating all of the points, the program would ask if the user

agrees or would like to change again. If they agree then it would proceed. If not then the loop continues.

```
user1 = User(name, gender, Attr)
os.system('cls')
print("-----")
print("User Created !")
print("Name : ", name)
print("Gender : ", gender)
print("Attributes : ", Attr)
print("-----")
print("Welcome to Cloud RPG !")

return user1
```

after the allocation of the points, the user is created in a temporary variable named user1. The input from the user is then printed again for the user to see. The final part returns the variable containing the object for other part of the program to use.

```
def checklevel(userobj): # checks the level of the user and add 1 attr point when level up
    if userobj.EXP >= 10:
        userobj.EXP = userobj.EXP - 10
        userobj.Level += 1
        print("-----")
        print("You currently have 1 points to spend on attributes")
        print(userobj.Attr)
        pick = str(input("Pick an Attribute to spend Points on: "))
        if pick.upper() not in userobj.Attr.keys():
            print("error input")
        else:
            userobj.Attr[pick.upper()] += 1
        print("-----")
    else: pass
```

This part of the program checks the EXP of the user, this would allow the user to level up and gain an attribute point to which they can allocate again. If the user has 10 exp or more, the level would increase by 1 and the exp reduced by 10. It uses similar coding to when the user is initially created.

d. Town.py

Town.py handles the interactions made inside the town in the game, hence the name “Town”. This class handles the shop, blacksmith, inn, and inventory management.

```
shopinventory = {"Sword":10, "Bow":10, "Shield":10, "Health Potion":10, "Iron Armor":10, "Food":50, "Drink":50}
shoplist = list(shopinventory.keys())
shopPrice = {"Sword":7, "Bow":5, "Shield":4, "Health Potion":5, "Iron Armor":7, "Food":2, "Drink":2}
craftPrice = {"Sword":3, "Bow":2, "Shield":2, "Health Potion":5, "Iron Armor":2}
recycleavail = {"Iron Ingots":10, "Wood":10, "Red Liquids": 5, }
materials = ["Iron", "Wood", "Red Liquids"]
```

A number of variables are used in order to have a deposit of the stock configurations of the store prices, available stocks / supplies and needed materials for crafting.

```
def shop(userobj): # buy items
    while True:
        os.system('cls')
        print("-----Ethereal Fowl Shop-----")
        print(f"your current gold amount : {userobj.UserInventory['Gold Coins']}")
        print("The shop is currently selling ")
        print("item : price : availability")
        count = 0
        for item in shopinventory.keys(): # display the items
            count += 1
            print(f"{count}. {item} : {shopPrice[item]} : {shopinventory[item]}")
        print("-----")
        print("What would you like to purchase ?")
        print("input 'exit' to exit the shop")
        buyChoice = str(input("input the EXACT NAME in the list : "))
        if buyChoice == "exit": break
        amt = int(input("input purchase amount : "))
        if shopinventory[buyChoice] != 0: #buy the selected item
            if userobj.UserInventory["Gold Coins"] >= (amt*shopPrice[buyChoice]):
                userobj.UserInventory["Gold Coins"] = userobj.UserInventory["Gold Coins"] - shopPrice[buyChoice]*amt
                userobj.UserInventory.setdefault(buyChoice,0)
                userobj.UserInventory[buyChoice] += amt
                shopinventory[buyChoice] = shopinventory[buyChoice] - amt
            else:
                print("Not enough Gold Coins")
        else:
            print("No longer available ")
```

Creating the shop system was rather easy as most of the data is stored within variables in the Town class. This meant that all I need to do is to display them and to create a program to exchange the gold coins in the inventory and to give the player / user the item that is purchased. To do this, I, firstly, ask the player/ user to input the exact name of the selected item that is to be purchased and firstly check whether the user didn't input “exit” or the user will exit the shop. Then the user will input the amount to purchase using numbers. After checking the user input whether it is available in the shop, and it is available, the program will check if the user's gold amount is more then or equal to the price of the item multiplied by the amount that the user wishes to buy.


```

while True:
    os.system('cls')
    print("-----Curious Forge-----")
    print("Current inventory:")
    print(f"Gold Coins : {userobj.UserInventory['Gold Coins']}")
    print(f"Iron Ingots : {userobj.UserInventory['Iron Ingots']}")
    print(f"Wood : {userobj.UserInventory['Wood']}")
    print("-----")
    print("-----Part 1-----")
    print("Item : Cost : Iron : Wood")
    print("1. Sword : 3 : 2 : 1")
    print("2. Bow : 2 : 0 : 2")
    print("3. Shield : 2 : 3 : 2")
    print("4. Iron Armor : 2 : 3 : 0")
    print("-----")
    print("Item : Cost : Red Liquids : Yellow Liquid")
    print("5. Health Potion : 5 : 4 : 0")
    print("-----")
    print("What will you Craft ? ")
    print("type 'exit' to exit the blacksmith")
    craftChoice = str(input("input the EXACT name of the item : "))

    if craftChoice == "exit": break # exit the blacksmith
    amount = int(input("amount to craft : "))
    # crafting program below
    if craftChoice == "Sword" and userobj.UserInventory["Iron Ingots"] >= amount*2 and userobj.UserInventory["Wood"] >= amount*1 and userobj.UserInventory["Gold Coins"] >= amount*3:
        userobj.UserInventory["Gold Coins"] = userobj.UserInventory["Gold Coins"] - amount*3
        userobj.UserInventory["Iron Ingots"] = userobj.UserInventory["Iron Ingots"] - amount*2
        userobj.UserInventory["Wood"] = userobj.UserInventory["Wood"] - amount*1
        userobj.UserInventory.setdefault(craftChoice, 0)
        userobj.UserInventory[craftChoice] += amount

```

the code above is responsible for crafting. First, when the code is called in Main.py, the visuals are drawn first after cleaning the terminal / command prompt. The user's gold coins, iron ingots, and wood are displayed so that the user is able to know the number of items that they can craft. To craft, it is similar to buying. The user needs to input the exact name of the item they wish to make and the amount to make, and then if the user has enough materials and gold, the crafted item would be sent immediately to the inventory. To send the item to the inventory a .setdefault() is used to make a key-value pair in the inventory dictionary with the "key" being the name of the item and the "value" being the amount of items you have in your inventory. Doing this will allow us to have the item ready and to only adjust for the amounts. When crafting, required gold, iron and wood or gold and red liquids are subtracted, then the item is added into your inventory. This would repeat for other types of items.

```

while True:
    print("-----The Recycle Station-----")
    inventory(userobj)
    print("-----")
    print("Current shop inventory:")
    print(recycleavail)
    print("-----")
    print("Able to Recycle: ")
    print("Sword | Bow | Iron Armor | Health Potion")
    print("what do you wish to recycle ?")
    print("type 'exit' to exit the station")
    cycleChoice = input("type the EXACT name: ")
    if cycleChoice in userobj.UserInventory.keys(): #recycle the selected item
        if cycleChoice == "Sword":
            if recycleavail["Iron Ingots"] >= 1:
                userobj.UserInventory.setdefault("Iron Ingots", 0)
                userobj.UserInventory["Iron Ingots"] += 1
                userobj.UserInventory["Sword"] = userobj.UserInventory["Sword"] - 1
            else:
                print("not enough materials from the Station")
                print("-----")
        elif cycleChoice == "exit": break
    else:
        print("Incorrect Input !")
        print("-----")

```

For the recycle station, it is very similar to the crafting station. The main difference is that the program would first check whether the item the user wants to recycle is available in the inventory.

All of the code will have some form of error checking like the one on the left.

e. Dungeon.py

Dungeon.py is responsible for the battles and receiving loot such as treasures or rare treasures

```
def dungeon(userobj):
    os.system('cls')
    print("You enter the dungeon")
    while True:
        print("-----The Dungeon-----")
        print("would you like to search deeper ?")
        print("1. Yes")
        print("2. No")
        print("-----")
        searchChoice = int(input("Input number to continue : "))
        if searchChoice == 1:
            path = [1,2,3]
            randopath = random.choices(path, weights=[50,5,1])
            if randopath == [1]:
                battle(userobj)
            elif randopath == [2]:
                treasure(userobj)
            elif randopath == [3]:
                rare(userobj)
        elif searchChoice == 2: break
        else: print("Error input")
        time.sleep(1)
        os.system('cls')
```

The code above is responsible for the flow of the search in the dungeon. I used random.choices as a way to guarantee higher chances of the user to go to battle and lower chances to immediately receive treasures. This helps as the user is intended to continuously fight in a battle rather than getting treasures all the time. This part also helps if the user wishes to exit the dungeon.

```
def battle(userobj): # battle the enemy ( this will happen most of the time )
    os.system('cls')
    enemyhp = 100
    enemyAGI = 4
    print("-----")
    print("-----you encountered an enemy !-----")
    print("-----")
    time.sleep(3)
    while True:
        os.system('cls')
        if enemyhp <= 0:
            userobj.objective = userobj.objective - 1
            userobj.EXP += 2
            treasure(userobj)
            if userobj.story >= 1 and userobj.objective <= 0:
                userobj.strComplt = True
            break
```

The battle system is also rather difficult. A while True is used to keep the user in a loop until the enemy is defeated. But first a message would appear signalling a battle that will be occurring. Before the battle starts, the enemy's health would first be set to 100 and the enemy's agility

attribute to be 4. When the battle starts, the program first checks whether the enemy health is 0 or

below. If not, then the program continues in the loop. If it is, then the battle would end and 2 EXP and a treasure chest would be rewarded. The program also checks for the completion of the story mode.

```
draw.statusbar(userobj)
if userobj.story >= 1:
    print(f"current objective : kill {userobj.objective} enemies")
print("-----")
print(f"Enemy Health = {enemyhp}")
print(f"Player Health = {userobj.Health}")
print("-----")
print("1. Attack")
print("2. Defend")
print("3. Heal")
print("4. Flee")
print("-----")
battleChoice = int(input("input the number to continue : "))
```

The visuals are then drawn with the status bar first showing hp, level, exp, equipped weapon and armour. And if the user is in story mode, the objective is displayed at the very top.

```
if battleChoice == 1: #attack
    if userobj.Equipped == "Sword": #currently equipped is sword
        print("you dealt damage yet you also received some")
        enemyhp = enemyhp - 50*(userobj.attr["STR"]*0.1)
        userobj.Health = userobj.Health - 4*(1-userobj.Armor)
        time.sleep(1)
```

The attack mechanics are rather simple when using the sword. You would deal damage in calculation of the alpha damage from the sword (50 dmg) multiplied by the strength attribute which is multiplied by 0.1. But using the sword would mean that the user would take damage. The damage taken is also in calculation of 4 multiplied by (1 minus the user's armour protection value). A time.sleep is used so that the reader is able to read the message.

```

elif userobj.Equipped == "Bow":#currently equipped is bow, there will be 3 scenarios / possibility
    hitProb = random.randint(1, 3)
    if hitProb == 1:
        print("you dealt damage")
        enemyhp = enemyhp - 30 * (userobj.attr["STR"] * 0.2)
        time.sleep(1)
    elif hitProb == 2:
        print("you dealt damage yet you also received some")
        enemyhp = enemyhp - 30 * (userobj.attr["STR"] * 0.2)
        userobj.Health = userobj.Health - 8 * (1 - userobj.Armor)
        time.sleep(1)
    else:
        print("You missed and you got hit!")
        userobj.Health = userobj.Health - 8 * (1 - userobj.Armor)
        time.sleep(1)

```

The next part is when using a bow. When using a bow, the user is then randomized into 3 different scenarios, where the user would inflict damage only, take damage only or both. This is equally randomized. The damage inflicted is different too. The alpha is now 30 and the strength multiplier is now 0.2. However, if you take damage the multiplier is now twice of that of the sword (8).

```

elif battleChoice == 2:#defend
    print("you defended the enemy's attack")
    time.sleep(1)

```

When using the sword received from the story mode, the “Excalibur”, the alpha damage of the sword is tripled to 150, instead of 50.

```

elif battleChoice == 3:#heal
    if userobj.Health >= 100:
        print("max health !")
        time.sleep(1)
    elif userobj.UserInventory["Health Potion"] >= 1:
        print("you used a health potion")
        userobj.Health += 10
        userobj.UserInventory["Health Potion"] = userobj.UserInventory["Health Potion"] - 1
        if userobj.Health >= 100: userobj.Health = 100 #cant go above 100
        time.sleep(1)
    else:
        print("no health potion left")
        time.sleep(1)

```

when defending, it is just a time.sleep for the message.

in order for the player to heal, there are some error checking that is made. Firstly, checking whether the health is already 100 or above, and then if the user have available potions. If the user already have 100 or more health or have no more potions, an error message would appear. If the

user have, then the health would be increased by 10 with the cost of 1 potion. However if the user would heal above 100, then the health would be changed to 100 as it is the max.

```
elif battleChoice == 4: #flee. success depends on the lck attribute
    if userobj.attr["AGI"] >= enemyAGI:
        if userobj.attr["LCK"] >= 6:
            chance = random.random(1,100)
            if chance >= 49:
                print("You've escaped !")
                time.sleep(1)
                break
            else:
                print("Unable to escape !")
                time.sleep(1)
```

To flee, the user would first have a higher agility attribute than the enemy which is 4. This would enable the user to flee. When fleeing, the luck attribute would be used and different levels would result in different percentages. the program would check the user's luck attribute and direct it to the different chances. random.random(1,100) picks a number from 1 to 100. Then the number is then checked if it is below the requirements or not. If it is, the user is able to escape. If not, then the user is not able to escape.

```
def treasure(userobj): #recieve normal treasure
    os.system('cls')
    userobj.UserInventory.setdefault("Iron Ingots", 0)
    userobj.UserInventory["Iron Ingots"] += 5
    userobj.UserInventory.setdefault("Gold Coins", 0)
    userobj.UserInventory["Gold Coins"] += 10
    userobj.UserInventory.setdefault("Wood", 0)
    userobj.UserInventory["Wood"] += 5
    userobj.UserInventory.setdefault("Red Liquids", 0)
    userobj.UserInventory["Red Liquids"] += 2
    print("-----")
    print("You found a treasure chest !")
    print("You recieved :)")
    print("1. 5x Iron")
    print("2. 5x Wood")
    print("3. 2x Red Liquids")
    print("4. 10x Gold coins")
    print("-----")
    town.inventory(userobj)
    print("-----")
```

```
def rare(userobj): # recieve rare treasure
    os.system('cls')
    recieve1 = random.choice(availloot)
    userobj.UserInventory.setdefault(recieve1, 0)
    userobj.UserInventory[recieve1] += 1
    userobj.UserInventory.setdefault("Gold Coins", 0)
    userobj.UserInventory["Gold Coins"] += 30
    userobj.UserInventory.setdefault("Iron Ingots", 0)
    userobj.UserInventory["Iron Ingots"] += 15
    userobj.UserInventory.setdefault("Wood", 0)
    userobj.UserInventory["Wood"] += 10
    userobj.UserInventory.setdefault("Red Liquids", 0)
    userobj.UserInventory["Red Liquids"] += 10
    print("-----")
    print("You found rare loot")
    print("You found :)")
    print(f"1. 1x {recieve1}")
    print("2. 15x Iron")
    print("3. 10x Wood")
    print("4. 10x Red Liquids")
    print("5. 30x Gold coins")
    print("-----")
    town.inventory(userobj)
    print("-----")
```

as mentioned above, if the user did not go to battle, then the user immediately receives the treasure. The treasure is brought to the inventory similarly like the shop by using .setdefault() and adding the inventory amount. Then the visuals would be drawn. EXP however would not be

rewarded. A rare treasure may be found with a chance of receiving a weapon and higher amount of gold and material.

f. Story.py

Story.py handles the story mode. It is mainly for the visuals but enables other part of the code by tweaking a part of the user's object.

```
def part1(userobj):
    os.system('cls')
    userobj.strComplt = False
    print("-----The Quest-----")
    print("you decided to meet with the adventurer guild")
    print("to see whether there are any quests that can be")
    print("done. you stumbled upon one quests that asks you ")
    print("to clear a certain part of the dungeon. But you need")
    print("meet with the person who will give you the area location")
    print("-----")
    input("press enter to continue")
    print("-----")
    print("you meet with the person and talked about the location")
    print(f"{userobj.name}: you must be the person who put up the quest")
    print("Nate: yes, this is the quest that i put up. its easy to go to")
    print("      location. just go to the left after you enter. ")
    print(f"{userobj.name}: any ideas on how many there will be ?")
    print("Nate: probably around 20 enemies in there.")
    print(f"{userobj.name}: OK, great. shouldn't be to hard ")
    print("-----")
    print("Current Obejctive : kill 20 enemies")
    input("press enter to continue")
    userobj.story = 1 #checker for the program to know that the user is in story mode
```

The code above is part one of the 3-part story, it shows what the user is doing and the conversation with the quest giver and shows the objective of that part. The part of the user's object that is tweaked is userobj.story. userobj.story is what signals the program that the user is in story mode and is currently in part one.

```
def part2(userobj):
    if userobj.strComplt == True:
        os.system('cls')
        userobj.strComplt = False
        print("-----The Quest-----")
        print("you've completed the quest and went to see Nate, the")
        print("person who hired you on this quest.")
        print("you go to his house only to find that it is empty...")
        print("a note was left behind saying that he have left to see")
        print("the part of the dungeon that you've cleared. you go back")
        print("only to find that he has died due to a trap.")
        print("-----")
        input("press enter to continue")
        print("-----")
        print("you found another note saying that he found another part")
        print("of the dungeon that ends with a valued treasure, called")
        print("----- EXCALIBUR -----")
        print("and thus you decided to go back to town first to prepare")
        print("-----")
        print("Current Objective : kill 35 enemies")
        input("press enter to continue")
        userobj.story = 2 #checker for the program to know that the user is in story mode (this means its at part 2)
        userobj.objective = 35 #changes the target / no of enemy to kill

    else: #means that you havent completed the quest / objective
        os.system('cls')
        print("-----You have yet to complete the mission-----")
        time.sleep(5)
        pass
```

part 2 is similar with changes such as an error checker for whether the objective is completed or not. If the user has completed it, the user will be brought to the 2nd part of the story and a new objective. Userobj.strComplt is a variable to verify whether the objective is complete or not. Every time the user goes to the next part of the story is returned to false, signalling that it hasn't been completed

```
def part3(userobj):
    if userobj.strComplt == True and userobj.story == 2:
        os.system('cls')
        userobj.strComplt = False
        print("-----The Quest-----")
        print("You've completed the dungeon and found the valued treasure!")
        print("you recieved : ")
        print("----- EXCALIBUR -----")
        print("with this the story comes to an end but this does not mean")
        print("it ends as a new chapter starts a new !")
        print("-----")
        input("press enter to continue")
        print("-----")
        print("Congratulations for making it this far ! ")
        userobj.story = 0 #elims
        userobj.UserInventory.setdefault("Excalibur", 0) # gives the user the sword
        userobj.UserInventory["Excalibur"] += 1

    else: #means that you havent completed the quest / objective
        os.system('cls')
        print("-----You have yet to complete the mission-----")
        time.sleep(5)
        pass
```

In part 3, the user is then given the “Excalibur” Sword using similar coding to the shop. And the story is ended. Userobj.story, which signals the program the user's current story part, is returned to 0.

VII. Lessons that have been learnt

There are a lot of lessons that I have learnt just by making certain parts of the program. I had before made some parts of the RPG such as the shop and menus, but the rest are new and I have yet to fully grasp on how to make such mechanic efficiently. Logically, the program is rather simple and it is ways to understand, but there are a lot of little aspects to the mechanic that made it hard and annoying to debug. Mainly with how every action that the user make have to change something. I mainly did a RPG game so that I would be able to learn better about OOP and how it would make coding easier and I can say it did made it easier for me. The hardest part for me to make was the town and dungeon class. The problem was mainly checking on the interactions of the player / user to his/her inventory. If the user was to buy an item, the item needs to be inserted to the inventory and exchanged to money or, if crafted, for materials. That was an aspect that was rather annoying because I had forgot to code one part and it made the program not work as intended.

There are plenty of error checking that needs to be done but for the majority of the code has worked well and as intended to be.

VIII. Project link

The following link will bring you to the github repository that contains the the RPG game:

https://github.com/JaysM32/Sem1_Python_FinalProject

IX. References

- Negi, M., 2018. *How To Clear Screen In Python?* - *Geeksforgeeks*. [online] GeeksforGeeks. Available at: <<https://www.geeksforgeeks.org/clear-screen-python/>> [Accessed 28 December 2020].