Overview of Application

The Basic RPG adventure is a command line application that was planned and developed in about 1 week for the Coder Academy Term 1 Assessment 3 in the Accelarated 2023 March Cohort. This application was used to test our skills in project management and development within a short time period. The coding style guide that was being followed is the pep8 guide which I have attempted to adhere to as well as fix code later on to follow it.

The application itself has 3 main features, allowing user to input commands, a weapon check and a random number password. With these features validation code is used in each to ensure graceful error handling for invalid inputs. I will go through the logic and code used as I go through the features provided.

1. User Input

The user will be able to input their name into the game at the start and it will be used as part of the intro question if they wish to play the game. The code here uses a while loop asking for player to enter a name. I allow the use of numbers and special characters as some people may wish to do that however as a name is part of the intro sequence of the game I used the strip() function to ensure it was not an empty space. Then the loop also continues if choice is invalid option for entering the game.

```
while True:
    PLAYER_NAME = input("Welcome to the game! Please enter your name: ")
    while PLAYER_NAME.strip() != "":
            # strip clearing end and start white space
            game_state = input(f'{PLAYER_NAME.strip()}, would you like to play? (y/n):\n')
            if game_state == 'y':
                    gaming.createRoom(gaming)
            elif game_state == 'n':
                print ("Game will now quit.")
                quit()
            else:
                print ("Invalid input please use 'y' or 'n'")
    else:
        print ("Invalid input please do not leave blank")
```

Through user input once the game is accepted the user will be able to move around the rooms using selected parameters given to the player, these parameters are validated to make sure that they are valid using a while loop to check if they have made a valid choice as of yet and if so to call another function to activate the next room. As with the image seen here I have used while loops to make the player choose a valid input which is specified in the input function, to get player options I used elifs to then call the next function in the code.

```
# MAIN ROOM
def createRoom(self, instance):
    while True:
        choice = input("You are in a room filled with torches with a few choices of direction. Do you go 'forward' 'back' 'left' or 'right'?
        \n")
        if choice == "forward":
            self.scaryRoom()
        elif choice == 'back':
            print("You attempt to use the back door of the room but it opens into a dead end")
        elif choice == 'right':
            self.skeletalRoom()
        elif choice == 'left':
            self.puzzleRoom()
        else:
            print ("Please enter a valid input")
```

In this snippet, I tried to use a different style of coding where I made a list and looped the input of the user until something in the list was chosen then went to an if elif state.

```
def skeletalRoom(self):
    allowed_choices = ['forward', 'left', 'back'] #list of possible choices
    while True:
        choice = input("You are in a room filled with skeletons do you go 'forward' or 'left' or 'back'?\n")
        if choice.lower() in allowed_choices:
            break
        print("Invalid input, please use a valid option")

    # once a valid choice is made
    if choice.lower() == 'forward':
        print("The door ahead opens into a dead end wall and go back to the previous room")
        self.skeletalRoom()
    elif choice.lower() == 'left':
        self.weaponRoom()
    elif choice.lower() == 'back':
        self.createRoom()
```

2. Weapon Check

Allow user to pick up a weapon and in a room and if the player tries to pick it up again the weapon is no longer available printing out new text that instead says the weapon is no longer where it was. The player will now have set the weapon to `True` which is then referenced by the code in a specific room of the game. This room checks if player has the weapon or not and depending on their choice to fight or flee they will either escape or die.

Firstly I created the weapon and made it part of the initialisation of class player and set as False because player does not initially have a weapon, I then instanced in the class rooms() so it was part of initialisation and accessible. I then created a function where the weapon could be set to True in later portions of the code.

```python
class player():
    def __init__(self, weapon=False):
        # initiates weapon and sets to False
        self.weapon = weapon

    def get_weapon(self):
        self.weapon = True          # sets weapon to True when called


class rooms:
    def __init__(self):
        # creates usable instance of player for weapon
        self.player = player()
```

This next portion of code is where the player will have the option to pick up the weapon and in doing so set weapon to True as well as change text because the code checks if player weapon is True or False.

```python
# Right side of the main room
def weaponRoom(self):
    # check if player weapon = True
    if not self.player.weapon:
        print("You see the hilt of a blade lodged into a wall")

        while True:             # loop until a valid choice is made
            pick_up = input("Do you pick up the weapon? (y/n)\n")
            if pick_up.lower() == 'y':
                self.player.get_weapon()
                print("You pick up the weapon and now have a weapon and return to the previous location")
                self.skeletalRoom()
            elif pick_up.lower() == 'n':
                print("You decide to leave the hilt and go back to the previous location")
                self.skeletalRoom()
                break
            else:
                print ("Invalid input please use 'y' or 'n'")
    else: # if player weapon is True
        print("You see the hole where the blade used to be. Other than that, it's just a dead end wall. You return to the previous room.")
        self.skeletalRoom()
```

This is later referenced in the monsterRoom() as it checks whether or not player has a weapon which affects their ability to escape.

```
def monsterRoom(self):
    print("As you enter the room you hear a grunt and eyes staring into your soul. ITS A MONSTER!")
    while True:
        fight_flee = input("Do you 'fight' or 'flee'?\n")
        if fight_flee == 'fight' and not self.player.weapon:
            print(death)
            quit()          # on death quit
        elif fight_flee == 'flee' and not self.player.weapon:
            print ("You escape the room and return to the previous location")
            self.scaryRoom()
        elif fight_flee == 'fight' and self.player.weapon:
            print ("YOU HAVE ESCAPED")
            print (complete)
            quit()
        elif fight_flee == 'flee' and self.player.weapon:
            print ("You escape the room and return to the previous location")
            self.scaryRoom()
        else:
            print ("Please enter a valid input")
```

3. Random Number

The game has a puzzle that generates a random number each time it is run and printed out to a CSV file. This file containing the number can be read by the player once they interact with the specific area required. This number can then be used to escape the dungeon by inputting it into another area of the maze and let the player through. The player only has 3 guesses otherwise they will die. However if the player decides to leave the guessing game, the number of guesses will reset back to 0.

Using the random package I below I wrote the number to a CSV file which is what the player will interact with.

```
import csv
import random
from art import *

death = text2art("YOU - HAVE - DIED!")
complete = text2art("COMPLETED!")

# Global available variable for reading through csv file
random_number = str(random.randint(10000000, 99999999))
# write to file
with open('random_number.csv', mode='w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow([random_number])
```

As seen here on this next image if player says they wish to read out number will be shown by code reading out the CSV.

```
def puzzleClue(self):
    choice = input('You see a book that looks like it relates to the door on the right. Read it? (y/n)\n')
    if choice == 'y':
    # Read the random number from the CSV file and provide the first digit as a clue
        with open('random_number.csv', mode='r') as file:
            reader = csv.reader(file)
            random_number = next(reader)[0]
        print(f'A series of numbers is written over and over again, the number is: {random_number} you return to the center of the room.' )
        self.puzzleRoom()
    else:
        print('You return to the center of the room')
```

If the player then uses that number in the puzzle they are able to escape. Otherwise the puzzle works by allowing the player to reset if they leave the puzzle and if they stay they have 3 attempts otherwise they die and the game quits using the quit() function.

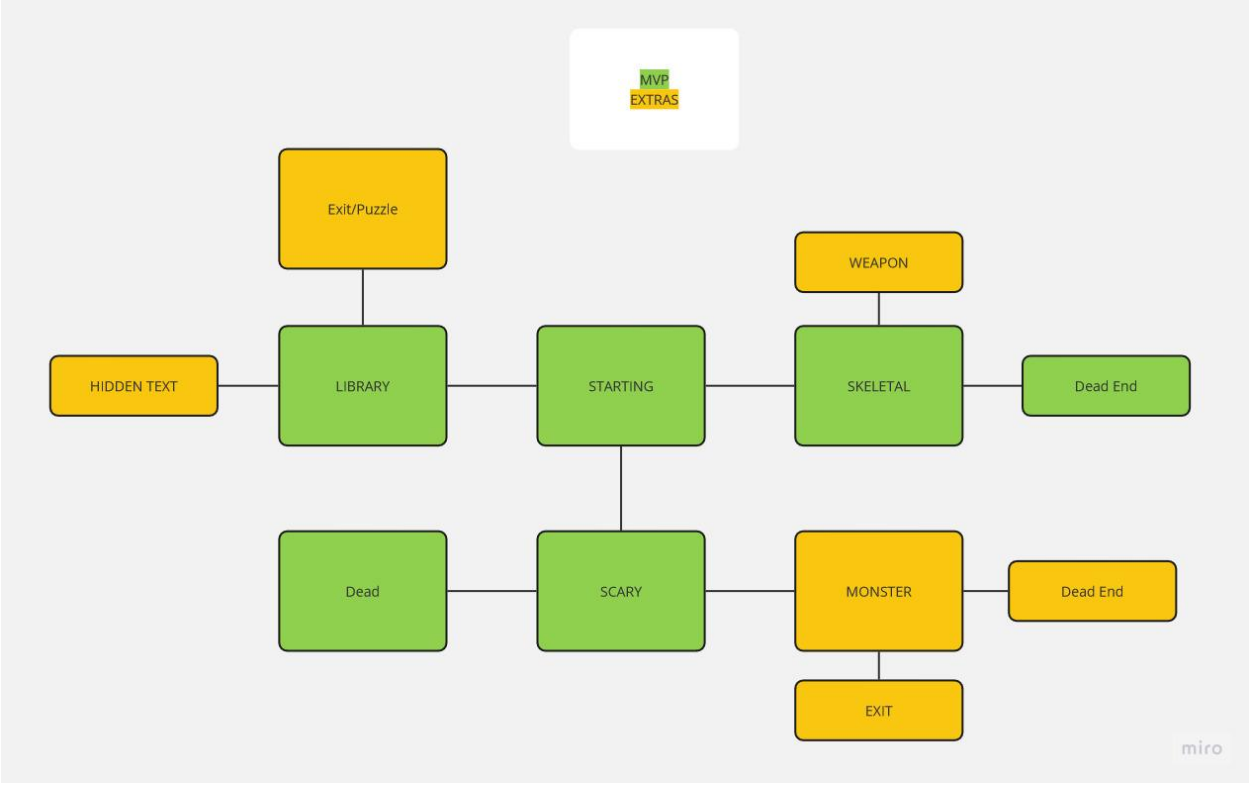
```
def puzzleEscape(self):
    attempts = 0
    guessing = True

    while guessing:

        print ("As you look closer at the door and you try to fill in a 8 digit code)")
        user_input = input ("Enter an 8 digit number\n")
        if len(user_input) == 8 and user_input.isdigit():
            # Check if player is correct
            if user_input == random_number:
                print('As soon as you enter the numbers doors begin to move and you escape the dungeon!')
                print (complete)
                quit()
            else:
                again = input('Sorry, your guess is incorrect. Try again? (y/n)\n')
                if again.lower() == "n":
                    print ('You hear something above you turning as you return to the center of the room')
                    guessing = False
                    self.puzzleRoom()
                elif again.lower() == "y":
                    attempts += 1
                    print(f"You have {3-attempts} attempts left.")

                    if attempts == 3:
                        break
                    else:
                        # reset guessing to True and prompt for input again
                        guessing = True
                else:
                    print("Invalid input. Please enter 'y' or 'n'.")
        else:
            print("Invalid input. Please enter an 8-digit number")
    else:
        print("Invalid input. Please enter 'y' or 'n'.")

    print ("The ceiling suddenly opens up and a spike trap comes hurtling down killing you.")
    print (death)
    quit()          # on death quit
```

4. Validation

With each of these features validation code is used in each area to ensure that the user is only inputting valid information, this information could be moving around the map or choosing different options, it can also be for inputting the password for the puzzle into the input to ensure that it is 8 digits and only contains numbers.

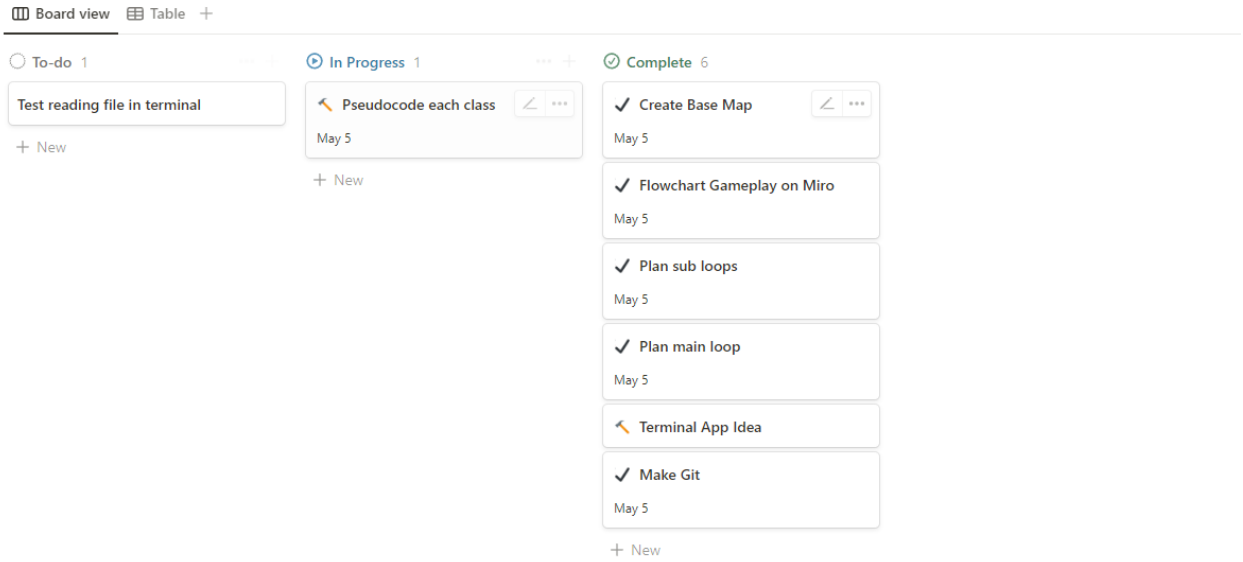
Development Plan

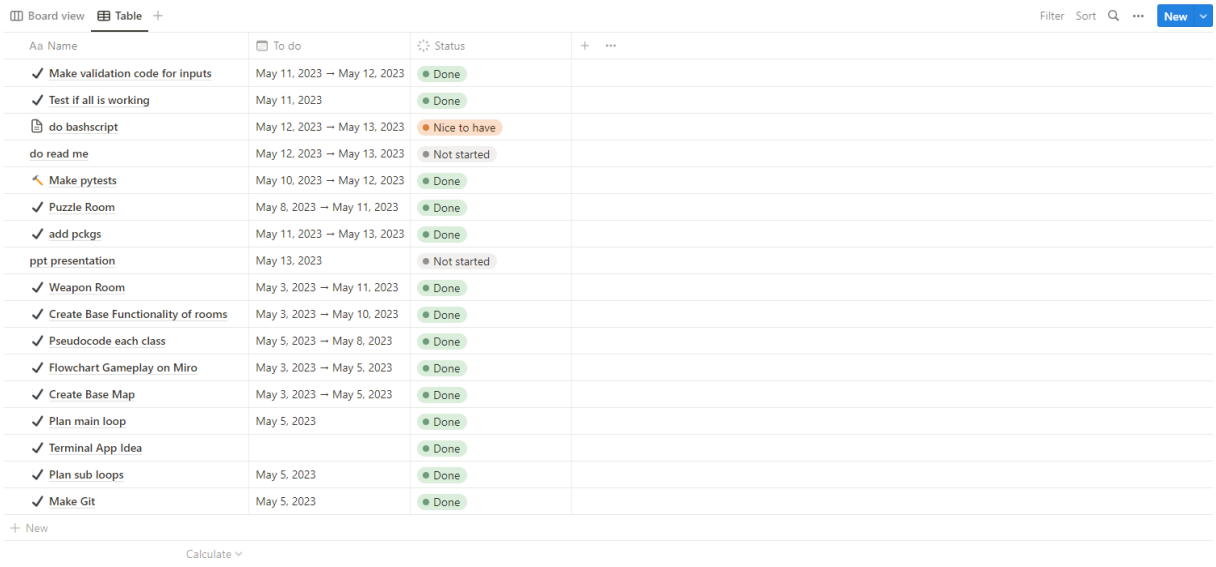
The development plan that I used was the agile plan so I would test and see if something was doable and then make adjustments accordingly. By doing this I adjusted my schedule and plan if I was able to complete things or add things by ensuring that my MVP was functional. This image is a map of the game in which green rooms were MVP and the yellow are nice to have if I am able to make it function.



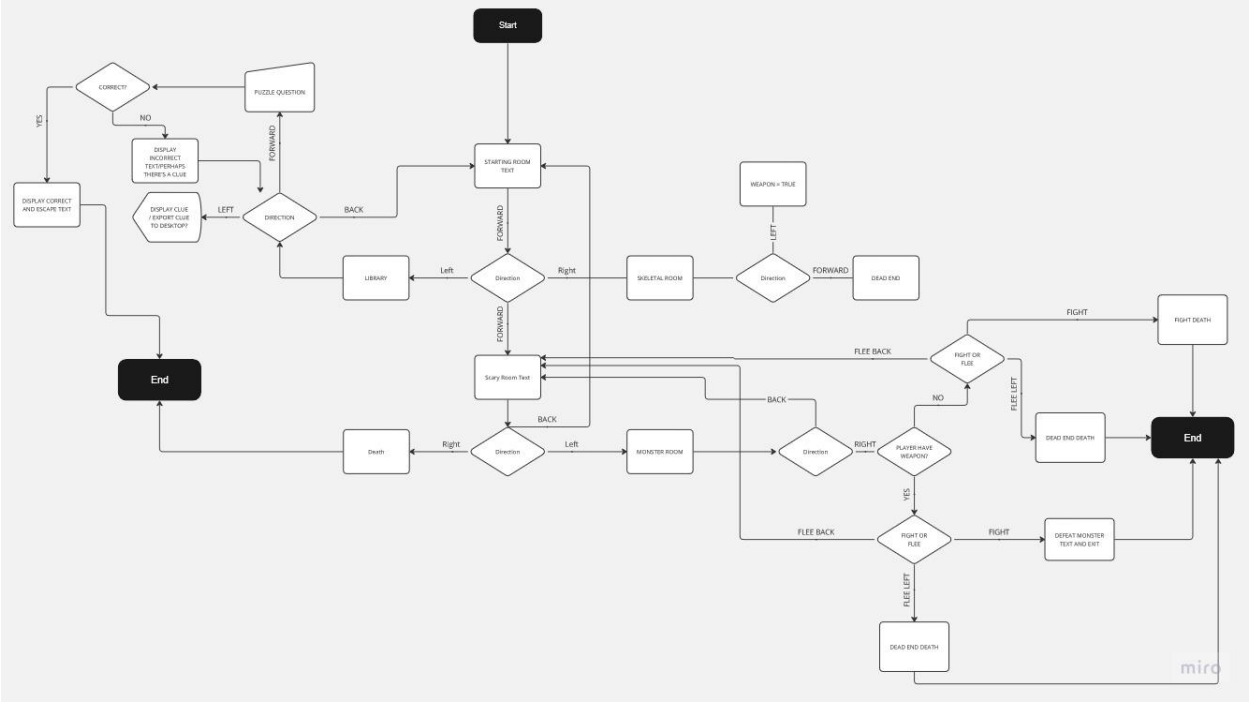
I used Notion for my project tracker, the reason for this is because I use Notion to track all of the other projects I am working on and I have gotten used to using it. However it does have less functionality for a kanban board as it only has To-do, In progress and Done functionality for the board. However it does have more options when changed to different formats such as table format.

Documentation for this project must be supplied as part of the following documents,
1. **README** document
2. Slide deck





The flowchart below is the game itself and how each choice should have gone if I was able to complete it in time. Some things may have changed however the functionality is still the same. For example the player originally had more options in the flowchart for fighting or fleeing however I did not have time to complete what I wanted to do. The flowchart however did help me understand what I wanted to do and how I could go about doing these things



Overall the parts I found most challenging was the testing portion as I have not yet got a good grasp of using pytest. The majority of the code was using things I had a good base of which was whiles, ifs, variable usage, functions, validation. I do think with more work I could make it cleaner and dryer however I would need to plan it out more in advance as well as understand how to refactor the code so that it is cleaner and more modular.

With the planning I am likely going to use an actual kanban board specific software in the future and need to make more time to do the coding portion instead of planning.