| PIC 16 Final Project: Element of Surprise

| Julian M. Rice - 204793996

Welcome to Element of Surprise!

Michael, thanks for giving me the opportunity to finish this class despite having the airplane and final exam conflict. Actually, I'm writing this as I am on the plane, and am adding finishing touches to the program. This document will give a slight overview of what the project is about, including a brief description of each of the files and some of the functions and features that I found to be harder and more interesting to write. Hope you find the project interesting to some degree - I worked hard on it!

| PART 1: File Descriptions

`final.py`	This is the main project, which consists of class definitions, a few cool functions, and a lot of PyQt.
`final.ui`	I made part of the graphical user interface using Qt Designer, and this can be viewed through the final.ui file.
`song.wav`	I wrote this mini song that acts as background music for the game, and kept it short at under 3 minutes.
`fire.png, water.png, grass.png, heal.png, block.png, swap.png`	These are some images that are used in the game. They are the options that the players can choose to use for attacking.

Fire - Strong against grass types, weak to water types
Water - Strong against fire types, weak to grass types
Grass - Strong against water types, weak to fire types
Heal - Heal a certain amount of hp, random (10 to 30)
Block - Block any attack with a slight likelihood of failure
Swap - Swap your type to another type (will be different)

| PART 2: Features

Game Rules: There are a couple of different elements (no pun intended) at play for this game: The first one is the option to choose what element your player will be. The options are **Fire**, **Water**, and **Grass**. Similar to what is seen in Pokemon, Each element is strong against another and also weak to the remaining element. In this case, **Fire** would be strong against **Grass** and weak against **Water** (and so on). This is determined by a **typeMultiplier** function at the end of the file.

Another component is the class choice - in this game, players are able to choose between four nice and unique classes: **Swordsmen**, **Fortress**, **Assassin**, and **Mage**. Each class has its own different block and critical rates. I will define those two parameters below:

➢ **Critical Rate** (1-100): Every turn, a critical number is randomly calculated. If the number falls within the range of the character's critical hit range, then they do double the damage!

➢ **Block Fail** (1-100): Every time the player decides to **BLOCK**, a block fail number is randomly calculated. If the number falls within the range of the character's block fail range, then the block will fail and the player will take damage from the enemy if the enemy used an attack.

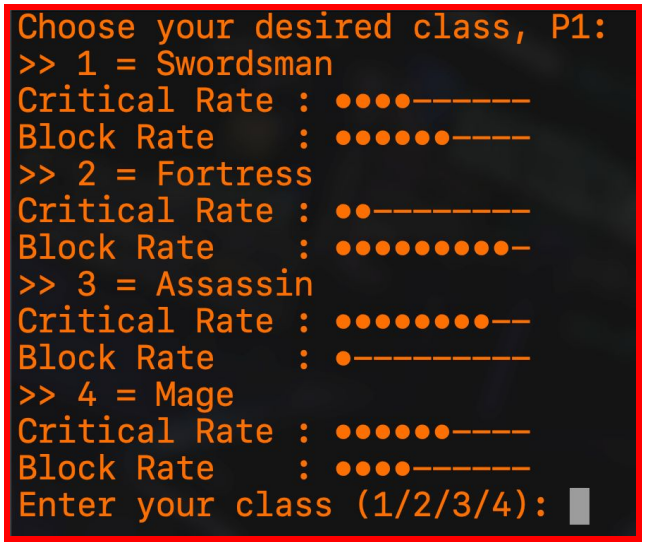
I defined constants at the top of the file to make my life easier when it came to deciding what moves did what. The constants include the following things:

- Move choices (1-6) - Covered above.
- Classes (1-4)
  - ✓ **Swordsman**: Good with the sword and is a great all around class.
  - ✓ **Fortress**: High blocking ability but pays for it with a low critical hit rate.
  - ✓ **Assassin**: Can deal serious amounts of damage but cannot block very well.
  - ✓ **Mage**: Not very good defensively, but is formidable with critical hits.
- Statuses (1-5) (More on this soon)
- Maximum Health (for healing) - Set to **100** for now.

I also made three lists of all total options / types / classes to enable while loop input. This means that the computer will continue to prompt you to type in a proper number if that number is not found anywhere in the potential lists.

- **potentialTypes** → Contains all the potential elements players can be
- **potentialOptions** → Contains all the potential moves a player can make
- **potentialClasses** → Contains all the potential classes a player can be

A Look into the **Gameplay (PyQt 5)**



A Look into the **Gameplay (Command Line)**

Statuses: I made a boolean array (list of boolean values) initialized to all **False** when calculating the effects of both players' moves in **`typeEffect()`**. If a player attempted to block, for example, then the block fail random number is generated and either a **successful BLOCK** or **failed BLOCK** boolean is set to **True**. This is then sent to the **`printOutput()`**, which creates a message string containing the data, or the results of the two moves being made.

```
            if enemyMove != BLOCK and enemyMove != SWAPPED:
                lostDamage = int(damageRate * random.randint(10
                if criticalHit:
                    lostDamage *= 2
            elif enemyMove == BLOCK:
                result = random.randint(0, 100)
                if (result <= self.blockFail):
                    lostDamage = 0
                    statuses[FAIL_BLOCK] = True
                    lostDamage = int(damageRate * random.randin
                    if criticalHit:
                        lostDamage *= 2
            elif enemyMove == SWAPPED:
                statuses[SWAPPEDBY] = True
            self.myHp -= lostDamage

        message = self.printOutput(lostDamage, gainedHealth, st
        return message

    def checkCritical(self):
        "Check if a critical hit was made. If so, return true"
        number = random.randint(1, 100)
        print("Critical Rate: " + str(self.critical))
        print("Number: " + str(number))
        if number <= self.critical:
            return True
        else:
            return False

#---------------------------------------------------------------
# SUBCLASSES OBJECT (Player Types)
#---------------------------------------------------------------
class Swordsman(Person):
    def __init__(self, name, myType):
        super(Swordsman, self).__init__(name, myType)
        self.critical = 5 #5 percent
        self.blockFail = 25
        self.className = "SWORDSMAN"
```

A Look into **Person** and **Swordsman**

```
    #Setting the turn style (P1 and P2)
    turnStyle = "background-color: white; color: red; border:
    self.playerTurn.setStyleSheet(turnStyle)

def profile(self):
    "Takes a cropped photo of the player and sets them as the
    cam = cv2.VideoCapture(0)
    ret, frame = cam.read()

    #Image name will be set to profile.jpg
    imgName = "profile.jpg"
    cv2.imwrite(imgName, frame)

    #This is the cropped version using a numpy array, I think
    final = cv2.imread("profile.jpg")
    final = final[50:650, 300:900]

    #Setting the background image to the background
    self.bg.setStyleSheet("background-image: url('profile.jpg'
    print("Photo done!")
    cam.release()

    #This is a good idea
    cv2.destroyAllWindows()

def gameStart(self, player1, player2):
    #PLAYER 1 TURN
    if self.mode:
        #Button Styling
        self.playerTurn.setText("Player 1's Turn")
        self.startGame.setStyleSheet('color: white; background
        self.startGame.setText("Move! (P2)")
        self.playerStats.setText("Player 2: " + player2.getCla
        self.playerStats.setStyleSheet("background-color: blac
        self.moveP.setText("  ")
```

A look into **OpenCV** and **gameStart**

Project Time & Comments

This was a tough project, and it took a lot of time relearning how to manipulate signals and slots with PyQt. What was more difficult for me, however, was learning how to implement a custom class (in this case, **Person**) into a PyQt interface. It took many hours of Youtube videos and Stack Overflow to get many of the cool features working in the game, especially because I was using Qt Designer for the first time. However, I am pretty satisfied with the final result. I think this is something that I can definitely spend more time expanding on given more time. The project time is split as follows:

# of Lines: 677 | # of Files: 10

Research: 12 hours
Development: 5 hours
Music: 2 hours
Documentation: 2 hours
Total: 21 hours

```
song = multiple_audio()
app = QApplication(sys.argv)
screenResolution = app.desktop().screenG
width, height = screenResolution.width()
window = Game(song)
window.setWindowTitle("Element of Surpri
window.setGeometry(width - 250, height -
song.play('song')
window.show()

sys.exit(app.exec_())
```

| **PART 3**: Conclusion

Expansion

Just reading about the descriptions of the sub classes makes me want to expand this into a bigger thing, which I might actually do during the winter break. Having different sub classes and saving a lot of coding time making them derived classes really speeds things up. A possibility could include there being a wide range of max health, attack, defense, and moves that might be unique per class. I would love to work an **OpenCV** feature that takes a snapshot of you and makes that your profile picture for the game (at least that match) for both players. Adding animations to the **PyQt** interface to spice it up sounds really cool, but I do not know how that would really work just yet either...! Perhaps a few different classes that represent different windows (menu, game, options, etc.) ...?

There's actually plenty more cool stuff in the game that I will not mention here, but is something that you could probably find in the source code if you were interested in some of the classes I wrote. After all, games are meant to be fun and surprising! Maybe you can play this with your **Pic 10A** class next quarter and give them a reward if they can beat you!

There might be some bugs in the program that I have not found. I did spend around an hour or two on the plane debugging the program, and I did manage to find 4 or 5 things, but nothing else has really popped up since. I am going to put this project up on **GitHub** later on!

Again, thank you for teaching this Pic 16 this quarter; it was more fun than I expected, and I learned a whole lot about a whole lot of modules that I did not even know existed! There is so much potential for every single one of the modules that there could honestly probably be a 10 week course that is simply dedicated to **OpenCV**, or **Numpy**, or etc.

Hopefully I'll see you again sometime!

Regards,
Julian