Problem statement

Our project provides a demo of a single player, visual roleplaying game. The turn-based combat consists of healing and attacking opponents on a stage which is selected from a world map. Our project is accessible through web-browser.
The demo aims to provide entertainment and a solid base for future expansion.

Flow Diagram

New Classes.py

class Unit():

attributes:

- name - string, the name of the unit
- max_hp - int, maximum health allowed
- hp - int, current health
- strength - int, affects attack power
- defense - int, dulls opponent's attack power
- isAlive - bool, keeps track of whether unit is alive or has been defeated
- lvl - int, scales unit stats
- stats - dictionary, holding all stats

methods:

attack(self, target):

calculate attack power (int) depending on strength and target's defense

if the target's hp drops below 0 or reaches 0:

update target's isAlive to False,

update combat outcome (dictionary) to include keys:

target.name - holding target.hp, self.name - holding self.hp, and finally "message" - holding "you win"

if unit's hp drops below 0 or reaches 0:

update unit isAlive to False,

update combat outcome to include keys:

target.name - holding target.hp, self.name - holding self.hp, and finally "message" - holding "you lose"

otherwise:

update combat outcome to include keys:

target.name - holding target.hp, self.name - holding self.hp, and finally "message" - holding "combat"

update all stats, and **return combat outcome**

heal(self, target):

calculates a random number to heal unit by, if unit's hp is not max_hp, if the number is bigger than max_hp then updates hp to equal max_hp.

update combat outcome to include keys:

target.name - holding target.hp, self.name - holding self.hp, and finally "message" - holding "combat"

return combat outcome

Flow Diagram

subfuncs.py

functions:

update_enemy(target, enemy list):

if the list is not empty:

update the target to be the last object of the list

return the last object of the list

otherwise, if the list is empty:

return “stage complete”

combat(choice, unit, target, enemy list):

if choice is attack:

call newClasses.unit attacking target,

call newClasses.target attacking user, store in result (dictionary)

return result

otherwise if choice is heal:

return call newClasses.unit heal

Flow Diagram

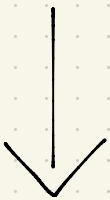
App.py
Flask app:
global variable count keeps track of whether the page has been reloaded.

@app.route('/'):
count is incremented by one.
if count is equal to one:
global variables, Carlos, enemy list, stages, and target declared.
Carlos is our main character, and we define him as a new unit object.
enemy list is initialized with a for loop using a range, inside the loop new enemies are defined as new unit objects.
target is set to the last object in the list
stages is a dictionary that holds whether each stage is in focus or not.
count is set back to zero, ready for next reload.

returns render_template('index.html' and passes unit objects Carlos and target)

@app.route('/getMap/', methods=['POST', 'GET']):
call global variable stages
if this route is called via GET method, **return json file containing stages**

@app.route('/getHP/', methods=['GET']):
if route is called via GET method, **return json file containing a dictionary holding Carlos' name and hp, "target" and target hp**



Flow Diagram

Flask app continued:

@app.route('/attackTurn/', methods=['GET', 'POST']):

call global variables Carlos, enemy list, target, and create global variable choice.

if route called via POST:

create variable message holding json file requesting information from JavaScript. (this is a dictionary)

call global variable choice,

set choice equal to message['choice']

return 'Success', 200

if route called via GET:

if choice is attack:

set variable outcome to hold the result of calling subfuncs.combat with global variables choice, Carlos, target, and enemy list

loop through the dictionary, looking for messages "you lose" and "you win":

if "you lose" set Carlos' hp back to max and set target's hp back to max as well to prepare to loop back around for round 2.

if "you win" check that enemy list is not empty to remove the last element from the list.

set target equal to the result of calling subfuncs.update_enemy

if target is "stage complete"

return a json file containing the message "stage complete"

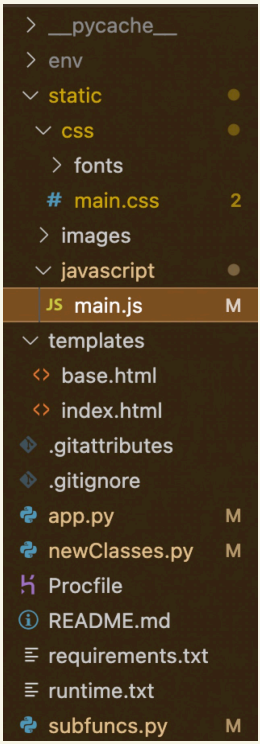
return a json file containing outcome if message is not "you win" or "you lose" to continue the flow of combat.

if choice is heal:

set outcome equal to the result of calling subfuncs.combat with choice, Carlos, target, and enemy list

return json file containing outcome.

Codebase Organization



Firstly, free floating files such as .gitattributes, .gitignore, profile, README.md, requirements.txt, and runtime.txt are mainly there for deployment with Heroku.

The env directory is for holding the Flask virtual environment which is activated using the terminal when doing test runs.

The static directory is for holding the css and JavaScript of the website - mainly for displaying graphics rather than interactivity. Inside the static directory is also images and fonts.

The templates directory is for holding the HTML of the website. Inside of the HTML, flask is implemented with Jinja2.

Finally, app.py is for adding python interactivity to the HTML, and talking to the JavaScript file through JSON files sent and received through jQuery's Ajax requests.

newClasses.py is meant to hold the classes required for the demo while subfuncs.py is meant to hold functions used for the logic behind the demo.

Module Breakdown

Flask:

serves as a middle man between JavaScript and the files containing Python logic.

example:

```python
@app.route('/attackTurn/', methods=['GET', 'POST'])
def attackTurn():
    global carlos
    global elist
    global target
    global choice

    if request.method == 'POST':
        msg = request.get_json()
        global choice
        choice = msg["choice"]

        return 'Success', 200
    elif request.method == 'GET':
        if choice == "attack":
            outcome = subfuncs.combat(choice, carlos, target, elist)
            for key in outcome:
                if key == "message":
                    if outcome[key] == "you lose":
                        print(outcome[key])
                        carlos.hp = carlos.max_hp
                        target.hp = target.max_hp
                    elif outcome[key] == "you win":
                        if len(elist) >= 0 :
                            elist.pop()
                            carlos.money += random.randint(15, 35)
                            target = subfuncs.update_enemy(target, elist)
                            if target == "stage complete":
                                return jsonify({"message": "stage complete"})
                        else:
                            return jsonify({"message": "stage complete"})
            return jsonify(outcome)
        elif choice == "heal":
            outcome = subfuncs.combat(choice, carlos, target, elist)
            return jsonify(outcome)
```

Flask will receive information from JavaScript when a button is pressed

Flask will send this information to JavaScript which will use ajax to update the html without reloading the page

```python
@app.route('/getMap', methods=['GET', 'POST'])
def returnMap():
    global stages
    if request.method == 'GET':
        return jsonify(stages)
    elif request.method == 'POST':
        currentStage = request.get_json()
        stages = subfuncs.update_stage(currentStage, stages)
        return jsonify(stages)
```

Flask app route endpoint listens for javascript call to app route.

```javascript
fetch('/getMap').then(function (response) {
  return response.json();
}).then(function (text) {
  stg1 = text.stage1;
  stg2 = text.stage2;
  stg3 = text.stage3;
  stg4 = text.stage4;
  currentStage = updateMap(stg1, stg2, stg3, stg4);
});
```

Javascript ajax using jQuery, receives the json file, calls a function to update the HTML.

html jinja2 example

```
<meter id="hpbarMU"
min = "0" max= {{ unit.max_hp }}
low = {{ unit.max_hp / 2 }} high = {{ unit.max_hp - 10 }} optimum = {{ unit.max_hp }}
value = {{ unit.hp }} ></meter>
<canvas id="mainchar" height="55" width="66"></canvas>
```

This sets the initial display of health bars through passing through a unit class to the

Contribution Statement

Blue (Leslie): HTML, Flask, Python, Art implementation

Kofi: presentation material

Oscar: combat

Braulio: art development