CS-A1121

Strategy game project plan

Make a merry Cate great again

Huong Do
2-25-2022

Contents

1. Personal information 2
2. General description and difficulty level 2
3. Use case description and draft of the user interface 2
4. Program’s structure plan 3
5. Data structures 3
6. Files and file formats 4
7. Algorithms 4
8. Testing plan 5
9. Libraries and other tools 5
10. Schedule 5
11. Literature references and links 5
12. Attachments 5

1. Personal information

Title: Make a merry Cate great again

Creator's name: Huong Do

Student number: 780443

Study program: Quantum Technology

Study year: 2019-2024

Date: 24/02/2022

2. General description and difficulty level

Users will play the role of a common, regular cat and try to increase their stats, collect items, and recruit other feline fellows in order to defeat the dog troops and make Cate - the merry feline kingdom - great again.

Difficulty level: Hard (with possibility to adjust to medium)

3. Use case description and draft of the user interface

The program will have different buttons that users can click on, which send information to the program. Some inputs (e.g. name) will be typed by the users. Upon getting the input, the program will response with corresponding actions or with a message to users on screen. The program knows when users click a button and executes it. The program also automatically plays the enemies after users finish their turns.

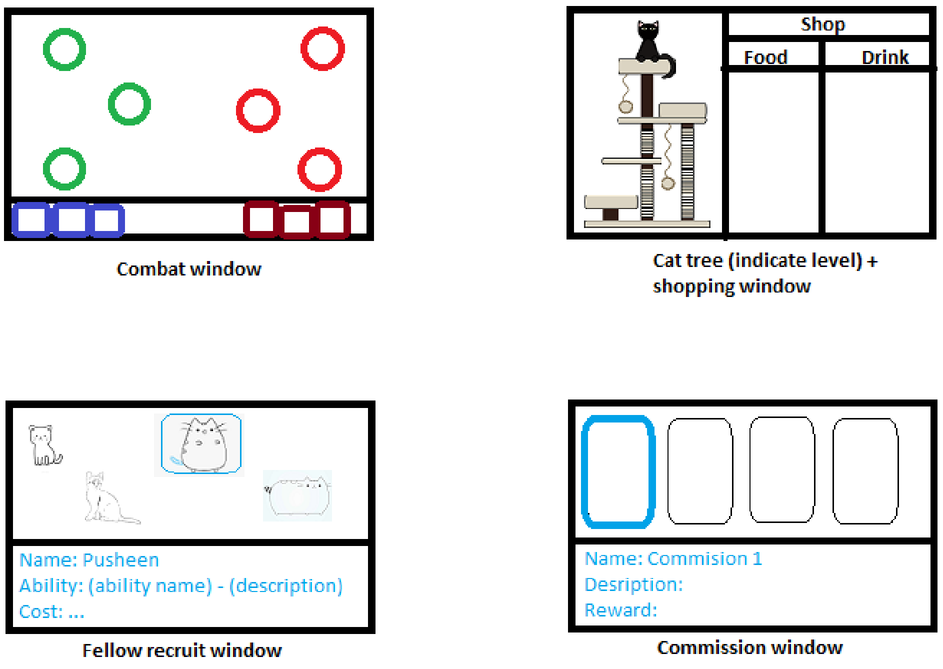

Figure 1: Draft of user interfaces

Real use cases

- Users equip toys (for normal attack) and gadgets (for special attack) for their mains, choose the preferred fellows and desired food and drink, and arrange suitable positions for their main and the 2 fellows. Then they can go to combat.

- At the beginning of the combat, users can choose a coin side and the program will flip the coin to decide who goes first. In one turn, users can execute 1 attack and use 1 item. After the users' turn ends, the program will execute enemies' actions automatically. The objective of the combat is to win, and the objective of the game is to defeat the final boss. In order to defeat the enemies, users need to strategically build their main as well as choosing the right fellows to join the combat. Bringing the right items and gadgets is also important as users can only equip 5 items and 1 gadget for each combat. To get items and increase their stats, users need to do different commissions to get rewards and CateCoin (which is used to buy food, drink, etc.)

4. Program's structure plan

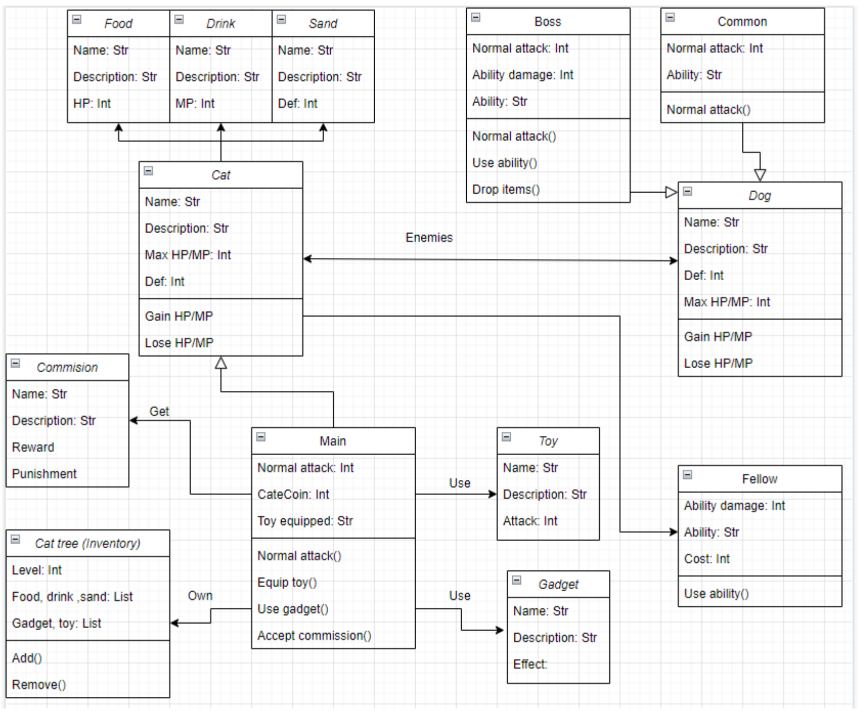

Figure 2: Draft UML class diagram

The two main classes are cat and dog, whose subclasses are *main* and *fellow*, and *boss* and *common*, respectively. The *cat* class has objects food, drink and sand whose local states are immutable. The *main* subclass of *cat* class has 4 objects: commission, toy, gadget, and cat tree. So far, each class only models the attributes of the classes.

5. Data structures

In my program, I used the following data structures:

- Dictionary: Dictionary can be used to store cats and dogs' info as they have a key-value structure without duplicate key. Moreover I can change value as I wish (like when their level increases)

- List: List will be used to store items, recruited fellows, etc. because it is completely mutable from the whole list to its elements.
- Tuple: Tuple will be used to store every item they boss can drop. Tuple is immutable so the bosses can drop the same items multiple times as the drop can be random.

In my program, I will use dynamic structure (e.g. list)

6. Files and file formats

Text files are used often as there are many features that include long description. Images are also needed for backgrounds, items, and different dog/cat breeds. In the text file, each input is a string containing different attributes of one feature. All attributes are separated by a dash and all inputs are separated by an enter ↵ so that file-reading can be done easily. An example of a text file is shown below:

```
Water/10/Restore a small amount of MP↵

Milk/20/Restore an adequate amount of MP↵
```

For images, they will be stored in their respective folder.

7. Algorithms

In a strategy game, math happens mostly when there's a combat. Below is an example of how the total damage dealt is calculated

$$\begin{aligned}\text{Total damage dealt} \\ &= ATK * (enhanced\ items) * debuff \\ &- def * (enhanced\ items) * debuff\end{aligned}$$

The AI in the program is implemented in boss-level enemies, who need to calculate how much damage they can deal to each opponent, as well as considering the impact of each opponent on the whole combat. By doing so, the boss enemies can figure out which cat possesses the highest level of threat. An example is given below:

A cat whose ability is healing the whole team should be killed first; however if that cat has some kind of protections, it will be much harder to kill it. In that case, the target should be changed to the cat that generates the protections. If such protections come from items or gadgets, bosses will change target to the cat with the next highest threat and come back to the original target once the items' effect wear off.

Other possible solutions are much less complicated:

- Enemies always aim cat with the lowest HP (with no regard to the target def)
- Enemies always aim cat that can deal the highest damage

8. Testing plan

The following tests can be executed to ensure the functionality of the game"

- Test security: The program should not allow adjustments to the pre-determined features, or when in combat, users execute more than 1 attack and 1 item/gadget in a single turn.
- Test damage done: This is done to ensure the right calculations.
- Test turn: This is to ensure that turns are in correct order, and that the enemies' turns are done automatically.
- Test skills: This is a complicated test which ensures the interaction among different skills, items, gadgets, etc.

9. Libraries and other tools

For the project, I will use PyQt and a random number generator.

10. Schedule

For the first checkpoint, all classes and text files for features should be completed. They need not be correct; however they provide a base to start the more complicated part.

For the second checkpoint, classes should function well, and all the calculations are done correctly. The AI of bosses is also implemented.

11. Literature references and links

I will use the previous lectures of this course as well as consulting further information from Google (which will be fully referenced).

12. Attachments

So far there is no attachment.