UC Software Engineering Project Report - Group 40

By

- David Liang (`dli83` , `69949439`)
- Vincent (`vno16` , `66346377`)

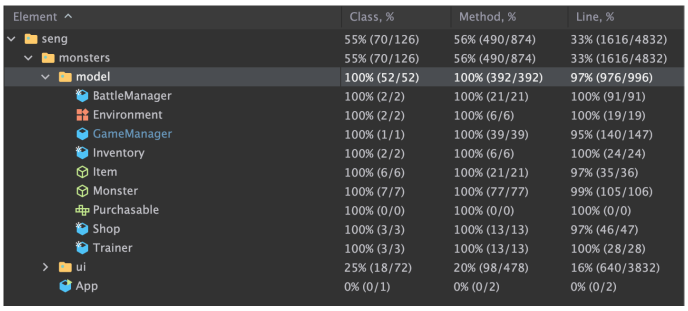
Structure of the Application

The application source code itself is divided into 2 main parts, the `model` and the `ui`. The `model` code is self-contained, and includes all the logic of the application, while the `ui` contains all interface elements. In the `model` portion, we decided the classes should be divided into an overall `GameManager`, a couple smaller `manager`-like classes that provide certain game features (i.e. inventory, battling, party management, etc.), and a base object, the `purchasable` interface, that defined the main game objects.

The `purchasable` interface only includes `Monster` and `Item`. We implemented both using abstract classes so that subclassing could be used to create multiple types of `Monster` or `Item`. We chose inheritance over interface for both since there is a lot of common functionality, and grouping those methods as non-abstract methods in a superclass was more efficient. Furthermore, we also took advantage of abstract classes by overriding common methods for certain special types of monsters / items. For example, the special `Eel` monster type, a monster whose life-stealing attack doubles as a self heal, overrides the superclass' `damage` method to also heal by `1/4` of the amount of damage dealt to an enemy monster.

A big decision we had to make is the implementation of the battle system, contained in the `BattleManager` class. We decided for simplicity's sake that we wanted the battle to be automatic and not require any player input, reducing the chance of anything breaking. Using this system, the battle would loop through a method with a delay, updating the battle state on each iteration. However, because the implementation of the delayed loop would differ between the CLI and the GUI, we decided that the `BattleManager` would only provide the method to be called on each iteration and the CLI and GUI would provide the loop separately in their implemented interfaces, which would also provide callbacks on each type of event that could occur in the battle.

Unit Test Coverage



Element	Class, %	Method, %	Line, %
seng	55% (70/126)	56% (490/874)	33% (1616/4832)
monsters	55% (70/126)	56% (490/874)	33% (1616/4832)
model	100% (52/52)	100% (392/392)	97% (976/996)
BattleManager	100% (2/2)	100% (21/21)	100% (91/91)
Environment	100% (2/2)	100% (6/6)	100% (19/19)
GameManager	100% (1/1)	100% (39/39)	95% (140/147)
Inventory	100% (2/2)	100% (6/6)	100% (24/24)
Item	100% (6/6)	100% (21/21)	97% (35/36)
Monster	100% (7/7)	100% (77/77)	99% (105/106)
Purchasable	100% (0/0)	100% (0/0)	100% (0/0)
Shop	100% (3/3)	100% (13/13)	97% (46/47)
Trainer	100% (3/3)	100% (13/13)	100% (28/28)
ui	25% (18/72)	20% (98/478)	16% (640/3832)
App	0% (0/1)	0% (0/2)	0% (0/2)

We managed to get 100% coverage for the classes and methods. We reach this high percentage by making sure that all our test code covers all possible and realistic cases where each class and method are used which include common edge cases. However, we didn't get the 100% coverage looking from the point of view of each line of code. We tried to make sure the tests cover areas that makes sense in making sure the code works properly, instead of just trying to reach a high percentage in the coverage. There are certain if statements (guard statements) in some methods that are not tested, mostly due them being very simple or have similar logic to other similar cases. We felt that testing those lines would be relatively difficult to make meaningful.

Thoughts and Feedback

Vincent

I personally feel that this project was a nice and fun way of applying the course materials. The only minor complaint I have is that it feel like that the labs and the lectures in the second half of the semester were a little disorganised because of the project. For instance, some the labs focused on topics for the GUI, while the lectures were discussing something else.

David

I think this project was, for the most part, a very good introduction to project management and program design. One complaint that I have, however, is that almost all the project management, design, and architecture course content came in the second half of the semester, which would have been invaluable in starting off and planning the project. I do understand why it was this way, as being able to write Java code comes first, but I can't help but feel as if there must have been a way to introduce these concepts earlier.

Brief Retrospective on the Project

One thing that lent well to the progression of the project was our planning. Rough drafts of psuedo-code, use case diagrams, and the CLI and GUI smoothed out coding and production, ensuring we didn't lose sight of what the final product would look like and how it would work.

The biggest issue we ran into was that we often over-estimated the time we had. This is particularly visible in our CLI and its unit tests, which was slightly rushed at the end. We could have prepared a bit better in allocating time to certain tasks.

For the battling, we were having difficulty visualising the process in an interesting manner. We ended with an implementation where the `BattleManager` keep track of its progression system and have the UI provide callbacks that could be notified to the user if certain events happened. While this new approach worked surprisingly well with the GUI, it was definitely rushed, fairly complicated, and something you could argue could be improved.

To sum up, most of the project went fairly well. While there are some small issues here and there, we were able to tackle them rather easily.

Effort Spent

Vincent

- `86-96` hours in total spent
    - `20-24` hours the first 2 weeks, spent **planning** and **designing**

- `32-36` hours during the 3 week break, spent **implementing** the `model` classes and unit test
- `16` hours the week after the break, spent **implementing** the `GUI` classes and unit test for the `CLI`
- `18-20` hours the last 2 weeks, spent **making necessary changes** that the project met the requirements

David

- `72 - 82` hours in total spent
  - `14 - 16` hours spent during weeks six and seven on **planning** and **designing**
  - `24` hours spent during the three week term break on **implementing** the `CLI`
  - `18 - 22` hours spent during weeks eight and nine on **testing** and **improving** the `GUI`
  - `16 - 20` hours spent during week 10 on **making final improvements** and ensuring **project deliverables** were completed

Agreed Contribution

We didn't explicitly agree on a strict % contribution. We just agreed that we both will try to reach a good portion of contribution, somewhere around `40%-60%` (optimally `50%`) for every part of the project. We didn't have either of us contribute strictly `50%` of the work because it would have been hard to split the work exactly evenly and we felt that it was unnecessary and not very productive.