1. **Personal information**

   Title: Make a merry Cate great again

   Creator's name: Huong Do

   Student number: 780443

   Study program: Quantum technology

   Date: 06/05/2022

2. **General description**

   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. I realized the scope of my project is too large, and I didn't have enough resource to accomplish it. My difficulty is thus below medium as the project is incomplete.

3. **Instructions for the user**

   The program can be launched using via the main.py

4. **External libraries**

   The PyQt library was used intensively. I also used the code from the robot assignment.

5. **Structure of the program**

   My program follows quite closely with the initial project plan's structure, with cat and dog being the 2 main classes; the central methods are probably the functions of attacking, especially for the dog class. The UI of the program is a combination between the robotWorld assignments, and my own writing based on what I read and understood from the documentation of PyQt; I used stacked and grids layout to implement most of the GUI.

   In addition, I implemented Game and Round classes to keep track of the progress of the program, though they are not fully implemented due to difficulties and confusion with the classes' functions. These 2 classes' main function is check_game_status.

6. **Algorithms**

   The mathematical formulas are mostly for calculating the damage output/input. Unlike the plan, I omitted the use of buff and debuff due to the complexity which I struggled a lot (although I have implemented the item causing the buff/debuff, I figured time is better spent on other functionalities). The AI is implemented through the Dog class (bosses and common dogs alike). I wrote different types of attacks in the parent class and from there, the

attack's styles of bosses and common dogs are determined depending on the difficulty of the round. Another possible solution is using random but with a more sophisticated method of taking seed, which can give better control of the game; however I didn't go through with it because I felt it doesn't have much to do with AI.

7. **Data structures**

   At first, I used a lot of dictionaries to store information of items with different names, value, etc., then I removed all of the dictionaries and stored most information in text files. Using dictionaries was indeed much simpler; however at one point my code became too long and had its readability reduced significantly. Other than that, I used a lot of list because I am more used to working with lists.

8. **Files**

   The files I incorporated in my program are text and JPG/PNG files. In the text file, the data of 1 item contains 4 different attributions, which are separated with a "/". Different data are separated by "/n".

9. **Testing**

   I hardly did any unittest for the project due to the time constraint as I spent way too much time on testing damage, skills, etc; the security and functionality of the program is thus jeopardized. Throughout the process of writing the program, I mostly test it by running the main.py file.

10. **The known shortcomings and flaws in the program**

    I left too little time to execute the GUI, which was surprisingly the most challenging part in my opinion. If I were to continue, I would find a way to join the GUI together with the functions because as of now, my GUI and functions are working separately.

    Unittests are also missing from my project. I think I was not used to writing code and testing it simultaneously. For half of the progress, I just wrote without testing and only started testing my code with main.py in the latter half.

    The way I wrote functions could have been improved as well. I realized my code doesn't work effectively, and that my writing style made it hard for me to find the problems.

11. **3 best and 3 worst areas**

    The 2 best things I can think of are probably the stack layer of the interface, and the idea itself. The 3 worst areas are the joining between GUI and program's function, the coding style, and the unittests. GUI is a quite new

knowledge, and PyQt was honestly not the best library for looking up tutorials online. My coding style, as mentioned earlier, is quite messy, even in the GUI for MainProgram which I claimed to be most proud of (I think I could use more OOP and make the file much more concise). I couldn't say much about unittests because I completely ignored the use of it, which was bad as problems happened toward to deadline and it was extremely hard to see where it has gone wrong.

12. **Changes to the original plan**

    I didn't implement the shopping and housing interfaces, which was a shame because I already had an image for them. It was mostly my failing to time-organization.

13. **Realized order and scheduled**

    Explain here generally in which order the project was ultimately implemented (preferably the dates as well). Was there a change made in the plan?

    I don't really understand this part.

    To my best understanding, I first implemented every class and function without testing them. Most of my functions are done at around the middle mark of the process; however those were the easier part.

14. **Assessment of the final result**

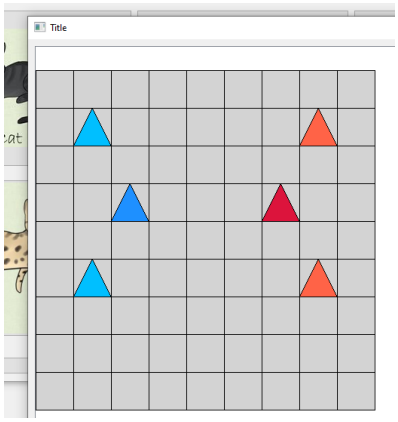
    This program of mine was poorly executed. I think I overestimated the complexity of the program in general, and the GUI in specific. The unittests could have been implemented better as well.

15. **References**

    I used robotWorld assignments, many preferences from stackoverflow which I don't remember, and the PyQt documentation.

16. **Attachments**

The main window



The battle window