Milestone 4 report

Design patterns

One of the design patterns we choose to use is the Observer strategy. We used this for our websocket setup, treating each frontend as a subscriber. This allowed us to facilitate easy communication between the frontend and the backend, eliminating the need for additional backend queries.

Another pattern which we used in multiple places is the Strategy pattern. One example of this is for our Action classes. Each implementation of the Action interface comes with two method implementations; one for checking if conditions have been met and one for resolving the effect of the card. These can be easily switched between as needed.

Similarly, each implementation of GalaxyNewsCard and ResistCard implements an effect method which carries out the effects stated on the card. Using the strategy pattern in these ways helped to keep our code simple, and allow the classes that interact with cards and actions to do so without implementing logic specific to each action/card.

Addressing feedback

No written feedback has been made available for us to address. During our in-person meeting, it was mentioned that we should include more detailed reasoning in our documentation, and so we have made an effort to do so for this milestone.

Design Decisions

For this milestone, we had to make a series of new design decisions. One example of this is that we decided to store agents as integers. The reason for this is that the only information that is really needed about an agent is its location on the board, which can be represented numerically. We also decided to make cats have an agent attribute, so that they may take an agent with them when they move to a new planet.

For meowssions, we created a new class called ActionLog, which is instantiated by GameStateController and stored in GameStateMode. These objects track the actions that have been taken, so that meowssion conditions may be evaluated.

Overall, there was very little refactoring to be done, as our code was largely in line with SOLID principles already. However, we did decide to refactor some of the code in FrontendGameStateController. Specifically, we got rid of an if/else chain by implementing a new function called getBody. This increased adherence to the open-closed principle and overall made our code cleaner and easier to read.

Division of labor

Throughout this project, Wyatt was responsible for writing the backend and Nathan was mostly responsible for the frontend. Nathan also created the first and second set of use cases. I (Ben) also contributed to the frontend, and was responsible for the updated use cases in third and fourth milestones, as well as the UML for each milestone, and the accompanying documentation. Some of the code in our project was also written collaboratively (i.e. one person wrote while we made decisions as a group).