Ben Hallihan, Wyatt Stagg, Nathan Kimball

COMP 3911

Milestone 4 Report

Addressing Feedback

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 GameStateModel. These objects track the actions that have been taken, so that meowssion conditions may be evaluated.

In the Frontend, a BonusEffect interface/abstract class was added (it is an abstract class that is treated as an interface because the frontend is written using React). Each concrete subclass of BonusEffect corresponds to one of the possible bonus cards, and has functions toggleEffect, checkConditions, and getRequestBody. Using a BonusEffect factory it is then easy to use the toggleEffect function to switch the gamestate in order to trigger a series of selection events on the screen for the user. For example, if the Welcomed As Heroes bonus card is earned, the WelcomedAsHeroes toggleEffect function will then trigger a switch to the gamestate that makes it possible to select a planet to liberate, and then select a cat to heal. When each of these things has happened, checkConditions will return true, which will then send a post request to the /meowssion endpoint containing welcomedAsHeroesInstance.getRequestBody() as a body. (That is just an example variable name).

SOLID

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 getActionRequestBody and an ActionBodyFactory. This increased adherence to the Open-Closed principle and overall made our code cleaner and easier to read.

There were also some blatant violations of the SRP in the frontend:

- The GlobalFascismScale component was responsible for rendering both galaxy news card messages, as well as the global fascism scale. This was solved by creating a separate GalaxyNewsCard component.
- Although this was fixed before the third milestone, there were component classes called “CatComponents” and “GameComponents”, which were vaguely named and were each responsible for the rendering of more than one component (CatComponents rendered each action button, GameComponents rendered the turn display on the right hand side as well as the game board). This was fixed by separating each of these components into their own classes which can be found within the “components” folder.

Design Patterns

Backend

One of the design patterns we chose to use is Observer. We used this for our websocket setup, treating each frontend connection 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.

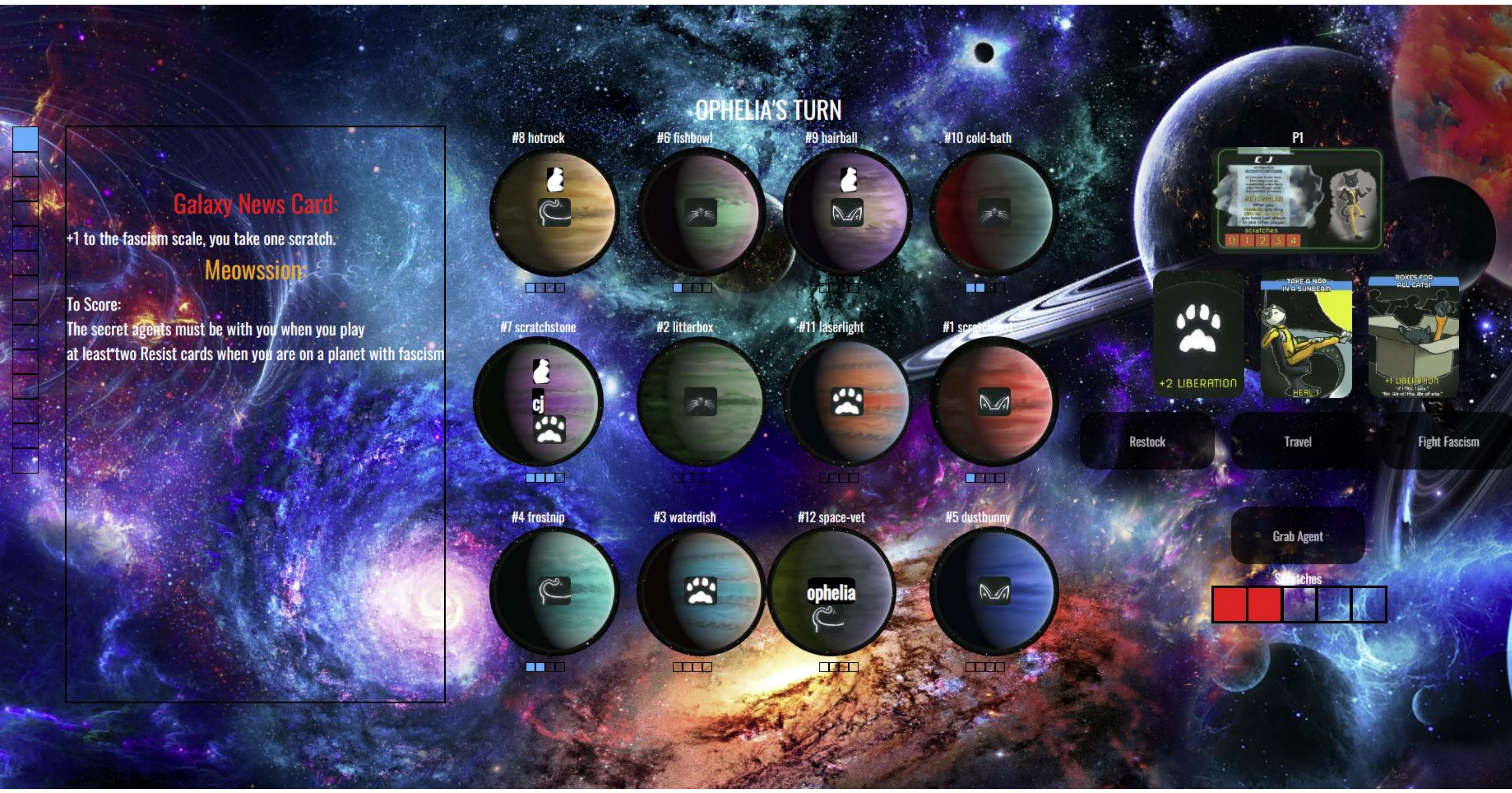
Frontend

The only design pattern added to the frontend was Factory. When actions are taken, the useAction function in the FrontendGameStateController is called. Since each action requires its own specific request body, the useAction function uses the getActionRequestBody function. This uses an ActionBodyFactory which maps an actionName to it's corresponding ActionBody instance, in order to then get the request body. (Since each subclass of ActionBody has a getBody function). Originally the mapping logic from the Factory was contained directly in the FrontendGameStateController’s getActionRequestBody function, and was added in order to eliminate a large if/else chain as was mentioned above. For this milestone, this seemed like a natural place to apply the Factory design pattern since the logic already existed within the larger function.

Similarly, with the new Milestone we need to be able to create new instances of the BonusEffect subclasses depending on the type of bonus card that has been awarded to the player. This was a natural place to use the Factory design pattern, since we can map each possible card name to a concrete subclass of BonusEffect in order to get that subclass.

Division of Labour

Division of labour 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, and helped with the descriptions of the frontend in this report. 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).



Galaxy News Card:

+1 to the fascism scale, you take one scratch.

Meowssion:

To Score:

The secret agents must be with you when you play at least two Resist cards when you are on a planet with fascism