Purry Duty

Rebecca Coryell, Taryn Hovenkamp, Jamie Joung

Final State of System Statement

From the two previous projects, we have completed more of the game's functionality, as well as improved the graphical UI. We were able to implement the game's unique feature of the player's mood, depending on the choices made, to impact the gameplay during the trial. We've completed the introduction and one level of gameplay, as well as developed the store aspect that allows the player to purchase items with currency. Furthermore, we developed a save and load system, allowing players to save their progress and return to the level/stats they had before. Due to lack of time, we were unable to create more levels for the game as well as coding additional features that we had initially planned, such as using items in future levels. In addition, we have implemented the following design patterns: Model-View-Controller, Factory, Memento, and Singleton.

Final Class Diagram and Comparison Statement

UML Project 5:
https://drive.google.com/file/d/10unfYM-GsLbYhrIHtsN36RsKh_G3f8IY/view?usp=sharing

UML Project 6:
https://drive.google.com/file/d/12ynol-dBWHqKS6650R5HniFSHEH1OBN5/view?usp=sharing

Final UML:
https://drive.google.com/file/d/1xX3aK89akJ6ql4cqSLNVRp-ZzDYKtwqH/view?usp=sharing

Key Changes:

- We implemented a Factory design pattern for the buttons created in the game to allow users to make different choices. With Unity, we were able to create prefabs, which allowed us to reuse the button objects for different sections of the game.

- In terms of the Memento design , rather than saving the state of the game as we indicated in the Project 5/6 UML, we decided to save and load the UserState from the SingleUserModel. While this did not save the exact moment the player is in the game, it would save their progress (what level they are at) as well as their name and items.
- Regarding the menus, we created a separate PauseMenu and MainMenu
- Originally, we hoped to use Singleton as the GameDriver to run one instance of the game, however, we decided to use it for the UserModel as it worked better in Unity, and allowed us to keep track of a single user throughout the game.
- While we were planning to create an overall GameDriver, we instead decided to use the InkManager, as we quickly realized that dialogue is a pivotal part of our game. The InkManager, in essence, acts as the controller for our scenes. In a way, this was perfect for us, as we had planned on utilizing the MVC design pattern. In our levels, the InkManager acts as the controller, our assets as the models, and the scene GUI as the view.
- In other scenes/views outside of our level, our controllers took the form of other managers. In the shop, our ShopManager script acts as the controller for the shop view. Our model for the shop consists of the assets used, as well as the user and its data (e.g. how much kibble they have).

Third-Party code vs. Original code Statement

For the store aspect of our game, we referenced this tutorial, but some of the code, specifically the code used to store the items into the user model, was written by us.
https://www.youtube.com/watch?v=Oie-G5xuQNA

For creating a pause menu within the game, we referenced this tutorial:
https://www.youtube.com/watch?v=JivuXdrIHK0

For our dialogue parser which also included possible choices that the user could make, we used this tutorial, as well as the scripting language, Ink:
https://klaudiabronowicka.com/blog/2020-12-01-making-a-visual-novel-with-unity-2-5-integration-with-ink/

Ink: https://www.inklestudios.com/ink/

For saving and loading the game states, we referenced this tutorial and applied the Memento design pattern: https://www.youtube.com/watch?v=XOjd_qU2Ido

NOTE: We have commented these references in our code as well.

Statement on the OOAD process for your overall Semester Project

One issue we experienced throughout the project was during the development stage, using Unity for the first time with no prior experience. In theory, our design patterns made sense to organize the classes and functionality of our game, however, the actual implementation was much more difficult, trying to weave together both C# and Unity to make a working product. In some cases, Unity has its own modules and functions that were much more optimal to use, so there were most definitely struggles to try to work with Unity while building out our planned project.

On the other hand, one design process element that went positively was being able to design and improve. After our initial planning, we continued to analyze how we wanted the game to function, as well as further discussion of the initial ideas we came up with. In the end, we narrowed down our goals so they were basic and reasonable given our resources and time limit, and as we developed the project, we continuously went back to redesign certain portions as arising issues were analyzed. For many aspects, we went back and researched multiple methods of tackling the problem, as well as taking ideas to create our own solution as well.

In addition, our team experienced some roadblocks when trying to implement design patterns utilizing Unity. For example, we originally wanted the InkManager to be an interface, which would then have other classes which implemented it (one for each level). We wanted to do the manager this way because some levels have slight differences between each other, but they ultimately utilize all the same methods. However, Unity (and by extension, the Ink scripting language extension *for* Unity), made this process difficult. Ink has these objects called *stories*, and they're the meat of the dialogue processing. They read in JSON files from which the dialogue is stored, and then there are various methods that can be used to parse the dialogue and tags. The issue lied in the fact that Unity and Ink were reading whichever story/JSON file came first; regardless of what class the code was in, so long as the classes implemented the interface, the story was being misread. We looked into the problem a bit, and it turns out that Unity has a hard time with serializable classes, and the Ink stories were not being updated. To remedy this, we decided to make two InkManagers – one for the intro, and one for all of the other levels. We realize that this isn't ideal for OOP principles, but due to the limitations of Ink and Unity, this was our best solution to the problem.