Pokemon TCG Report

Jack Diver

January 18, 2018

Abstract

This report will discuss various aspects of our Software Engineering Project, and how the groups interacted to create the final result.

1 Introduction

We were tasked as a group to create a clone of the PokemonTCG, for our Software Engineering Project. My sub-group was tasked with implementing an API for the AI subgroups, and implementing the core logic in the game. Here I will discuss my role within the group and my peers contributions.

2 My Role

I assumed the role of project manager. I was responsible for organising all group members (not just my subgroup), and making sure everyone had a task to complete. As well as this I was responsible for most of the API and class design for the project.

2.1 Player interaction

I spent most of my time thinking about how we would implement a system that was open to AI's and human players. The AI would not necessarily be owned by the game itself, which was a problem as it meant we couldn't trust that the AI would not cheat and so had to think of ways to prevent it. Another difficult aspect was thinking about how the AI would communicate with the game. The original idea was that the AI would puppeteer a player, however I found that this would be problematic to implement. To solve both of these problems we opted for an abstract player interface that both the Human player class and all AI's would inherit from. This made the AI's players, rather than controllers of players. The player class has private access to the game, which allowed us to implement a set of public functions that the derived classes could access. Using encapsulation in this way was very useful for limiting the actions and information that the players had.

2.2 Player interface

The player interface contains several functions that all derived classes must implement, the most obvious of which is the *turn* function. The turn function contains all of the actions that the player would like to execute on their turn. It is composed of calls to the *Player* class's public interface, and also a return statement. The *turn* function must return the players attack decision, instead of calling an attack function as this prevents attacking mid turn, and attacking multiple times.

2.3 Cards

I made the decision early on to use *unique_ptr's* (smart pointers) to store our cards. The idea behind this choice was that cards should never be copied, they should only ever be moved. At no point in the game, should there ever be more or less cards than there were at the start. The standard library's *unique_ptr* offered this exactly. Because of this, all player choice functions work with indices, rather than with the cards directly, they should always be thought of as requests to do something to a card, and should never actually modify a card itself. I also made the decision to use *Python* to write our card scripts as I thought this would be quick and easy for the other sub-groups, however in hindsight this could be considered a bad decision as many of my peers had not practiced *Python* in a while. I was responsible for binding our Game API and other classes to *Python* using *pybind11*. I designed the interface for moving cards to and from board

slots, as well from pile to pile (gave this to Eric to implement). A tricky part of the development was allowing the cards to request choices from the players, originally we were having issues with telescoping function signatures as we were struggling to stuff as many enums and information into the parameters as possible. Obviously this was a bad approach that would be difficult to maintain, so I developed the card and slot filtering system that uses a match function to filter the options. This gave much more freedom to the cards and allows for virtually any kind of properties to be used as a filter.

2.4 Effect queue

A problem we faced frequently with cards was the use of abilities, who's effects wouldn't be felt until later turns. To solve this I created the effect queue. This is a system that allows the cards to add an effect to the queue, along with a trigger, and a turn to execute.

2.5 Code review

I assumed the role of code reviewer throughout the project. I checked my peers commits and used the github issue system to report bugs in peoples code. In many cases however I was the one who ended up correcting such issues. This is relevant to both the code written by my sub-group and also the cards written by other groups.

2.6 Other

Other parts of the project that I implemented were: The human player and it's accompanying playerCommands (Renat then implemented the input system mention in the next section), the observer-like system that is used to drive the GUI and the game logger (also the game logger), the random AI (based on Renat's original implementation), the (seemingly worse than random) example ai, the card pile interface, and the development of a turn simulation framework which uses a dummy game, information hiding and a strategy player.

3 My Peers

3.1 My sub-group

Unfortunately I can only talk about the contributions of 2 of my sub-group members as the third was absent for the entirety of the project. Eric was responsible for damage calculation and was the chief council for anything rule-specific. He was also responsible for a large portion of guide/documentation. Eric spent a lot of time testing the game and the cards looking for bugs and reporting those to the other sub-groups, or in many cases fixing them himself, he also implemented the initial versions of moveCards, pileToBench and benchToPile, which I then reviewed and fixed. Renat helped out with writing documentation for the project, he was also the artist who designed the ascii-card art and slots, which I then used to implement our simple printer. Renat originally wrote a random ai and an ascii-printer for the GUI, however I found that the latter was difficult to maintain and so replaced it with the current implementation. Later Renat extended the simple printer to display large slots and cards, when the player wishes to inspect them. Renat also implemented the input loop within human player, which prompts the user for commands. He then assumed the role that was left by Owlwine of trying to create a GUI using OpenGl, however didn't have time to implement it. The three of us were very productive together at the beginning of the project in our meetings; everyone put forward their ideas and contributed to the initial design and UML diagram.

3.2 Other groups

The other teams had very minimal contributions until the late stages of the project. Moira on the bright tide team, was the only person to consistently work on their cards and AI. Stacey on the same team has had minimal impact on the project. The roaring heat team has also left their contributions till the late stages, and have "completed" their deck to a poor standard, causing my team to re-implement many cards. The forest shadow team has had zero contributions to the project. I feel as though all teams have missed the point of the project; instead of driving the development of the API they require, they have sat back and waited for a finished game. I'll refer you to the insight's tab on our shared repository, although it's not completely accurate, I believe it demonstrates my point.