Brendan Jang
CS 362
Assignment 2

Refactor

The refactoring that I did was relatively simple. I took the original code in both the testDominion1 and testDominion2 files and moved it to the new testUtility file. I created a new function called getBoxes in the testUtility file that will be called from the test files. To make this work, I had to replace the box dictionary instantiation with a call to a new function that I created in the testUtility file. In addition to these changes, I also had to make sure to import Dominion into the testUtility and testDomnion files as well to avoid a name error.

Test Scenarios

The first test scenario (testDominion1.py) that I envisioned was to simulate a typo in the code so I decreased the number of victory cards (nV) to 2 when there are more that 2 players. I expected this to greatly decrease the number of victory cards that were available for purchase therefore creating problems in strategy and win conditions thereby creating an endless game. When I tested this scenario, I noticed that this didn’t necessarily create an unwinnable game. There were instances where one of the players was able to win relatively quickly and other instances where the game was extended. I think this failed to cause a noticeable bug because although the number of victory cards was changed, the overall win strategy still stayed the same allowing for winners and losers. I wanted to further push this scenario, so I tried changing the number of victory cards to 0. I anticipated this to create an unwinnable game with no available victory cards but instead, this created a more noticeable bug where the game ended instantly with all players being the winner with the same number and type of cards. Upon further reflection, I think this outcome was inevitable because no victory cards available in the game means there is no win condition. Therefore, it makes sense that the game would end in a draw with all the players having the same cards.

The second test scenario (testDominion2.py) that I wanted to try was to create some typos in the supply_order to see if the player would still be able to purchase the items even if they were spelled wrong. In order to implement this bug, I changed a few of the names in the supply_order and changed the code in random10 to pick 30 random cards from the box to in the supply. I expected that the wrongly typed words would be displayed in the supply output but would not be purchasable in game due to the spelling error. However, when the program was run, I noticed that the wrongly typed names in the supply order did not display at all in the supply list output. When I tried to type in the wrongly spelled card in order to purchase it, I was given the error prompt that the card does not exist which was my expected outcome. I ran multiple test cases using 3 AI players to see what would happen. The AIs were able to bypass this spelling error and purchase the correct cards even though they were misspelled. This gave me an idea to try and purchasing cards that were not visible on the list using the correct card names. Typing the correct name of the card allowed it to be purchased regardless of if it showed up in the list of supply cards. This was an unexpected outcome, but it also makes sense because regardless of if the misspelled card were to show up on the list or not, the actual card still exists in the game. Therefore, it would still be possible to choose the said card if the game recognizes it as an existing card in the game.