1. I parametrized the knock strategy with strategy.getMaxKnockDeadwood(), and my player only knocks when deadwood is less than or equal to that value. Originally, I thought a value of 1 would be ideal because the simple player always knocks when its deadwood is <= 10. I figured that I would rarely get undercut, and if it knocked, I would undercut most of the time. However, I saw better results when only knocking with a deadwood of 0. I would assume this is because I never get undercut with this strategy.
2. Changes:

* Made it so that I can encode/decode my strategies. I tried to iterate through each and play a few games to find the best parameters for my strategies, but quickly realized that the combination between the several million games I would have to run and the high time complexity of the game made that infeasible.
* Made a static variable in my player and game state class so that I can change whether hands/cards are encoded as longs/ints or not, as it is MUCH easier to interpret Card objects when debugging. That said, it is a pain to program everything twice in slightly different ways, so I'm hoping to think of a more convenient way to do this.
* Before, when finding candidate cards to remove, after finding the highest-deadwood discards, I prefer to discard cards that can't be melded within 1 or 2 turns. I now made it so that it ignores other candidate discards when doing this.
* At the end of when I am finding a discard, if there are still several options left, I now ignore duples of cards within the candidates, as they are slightly more useful.
* Before, when my opponent knocked, I always presented my best meld set. Now, I iterate over every way to layoff cards, and find the best meld set for each way, taking the configuration which gives me the best deadwood after layoffs. I wasn’t expecting much of an impact from this, but the improvement was noticeable in both games against the simple player and an older version of my player.

1. 1. This paper’s main point is as follows:
      1. Because of the existence of chance in extensive games, we are put in a situation where it can be difficult to develop a strategy without creating some layer of abstraction and generality. Due to the incomplete information presented by the abstraction, however, we may lose information that is essential to the opponent’s strategy. As such, building our strategy off of this information could leave us vulnerable to exploitation by factors which are unknown to us. Rather than choosing any equilibrium, it is beneficial to tend towards less-exploitable equilibria; that said, abstraction doesn’t have any alternatives at the moment, so all we can do is be cautious about it.
   2. Gin Rummy is a game of incomplete information. Due to our lack of knowledge pertaining to what’s in the opponent’s hand (it’s very rare for us to know their entire hand), we have to create a layer of abstraction in order to interpret the game. However, according to the paper in question, this method could leave us vulnerable to factors we are unaware of. As such, it is important to be aware of our player’s vulnerabilities and reinforce them as well as we can, to reduce our exploitability as low as possible.
   3. The paper defined player 1’s exploitability as player 1’s best response strategy + player 1’s expected reward should they minimize their best-case loss. Player 2’s exploitability is defined as player 2’s best response strategy - player 1’s expected reward should they minimize their best-case loss. With respect to their discussion of Ledue Hold’em, they gave exploitability in terms of millibets/hand, stating how much a player was expected to lose per hand in the worst case. Strategies formed by an abstraction with a lower exploitability would be considered stronger strategies than those with higher exploitability, as they are closer to reality. In the case that exploitability is 0, refining the abstraction increases the accuracy of its representation of the actual game monotonically.
   4. After reading this paper, I would like to put some thought towards the development of strategies that are less exploitable, if it’s possible. Strategies that are beneficial in almost any circumstance would definitely be good choices to implement. The paper seems to imply that there isn’t currently a way to develop an effective strategy without some degree of abstraction, but I wonder if there’s a way to minimize the amount required so that we can reach a less-exploitable Nash equilibrium.