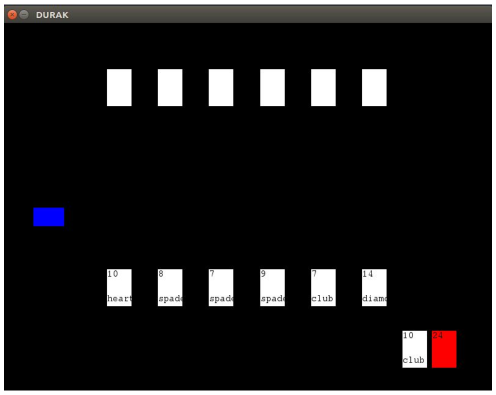
## *Project Overview*

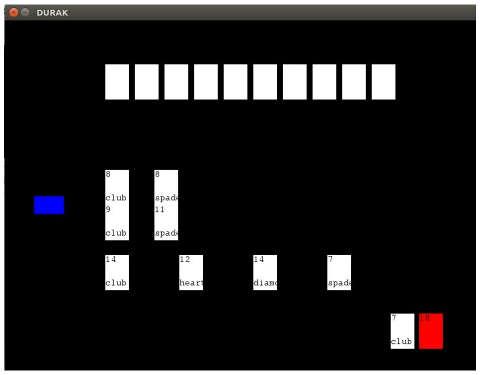
The short-term goal of our game was to create an engine to run card games out of. Although for this project we chose to play Durak, we hope to use this engine as a base to a personal project we both want to create: a randomized card game where everything (including the rules) is randomized.

## *Results*

What we accomplished in this project was first to create an engine for a future project for us to use. We have a very versatile engine, which means although we had to code in some things (such as the number of cards in the deck), we used a constants document to allow for easy changing in the future. We also created the possibility for easy expansion into the completely randomized game we envision for the future.
The first thing we did was created a playable hand class that held a single hand that we could display through another function, which we created and named to be card_game.py. We built off of this initial model to create multiple views for cards, the deck, and a turn button to allow for different actions based on the event each part experienced (seen below).



Screenshot of the start of the game.
We then began fully implementing our rules, which determines when cards can/can't be played as per the rules of Durak. A sample turn is laid out below, after the AI had been forced to pick up cards proving that it picked up cards as normal:

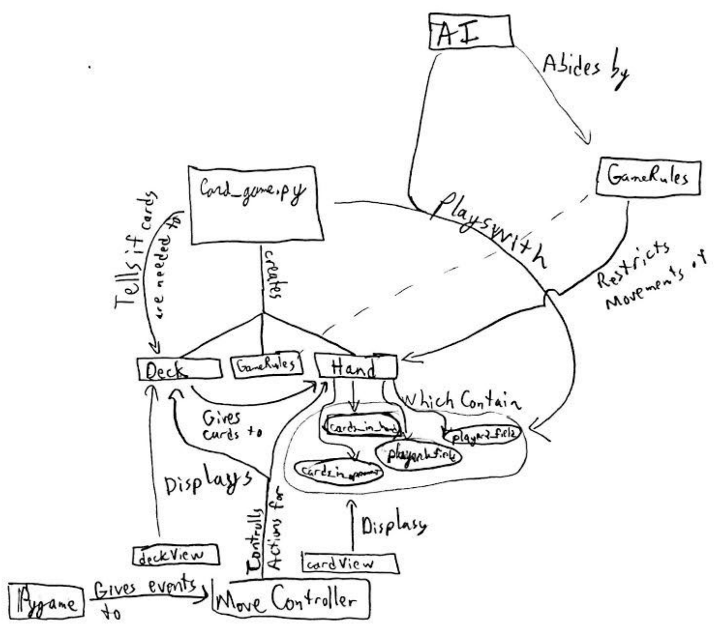


Overall, we had a working game that will exit the window and print out congratulations or you lost when a player runs out of cards.

## ***Implementation***

Our system's whole goal is to be an engine for future card games, and that is what we have accomplished at the core of this project. We have a very versatile card, hand, and deck classes that with slight tweaks could allow us to randomly generate cards and populate both one (or more) decks and hands with them. Our card_game.py function runs everything, initially creating one hand object which holds all hands (and the hands that are shown as the game field), a deck object (from which all Card objects are created and then taken from), appropriate views/controllers for all objects that the player sees, and then finally an ai class that is what a player plays against. The biggest component of this for us was to create a viable engine, which we did through allowing scalability throughout the design of our code. As we moved to implement this specific game, however, the hardest component was getting the computer ai working and displaying properly. It was surprisingly easy to implement the logic behind the AI but having the game display what it was supposed to took a lot of work because of a mixture of not enough communication about pre-written functions and the two of us being knowledgeable about pygame and about how to best display everything correctly. One design decision we made while designing the program was the choice to have one hand class that held all hands within it. This wasn't the initial idea behind the class, but as our coding progressed we found that it made the most sense if one object held all of the hands within it to keep all of the data in one easy to access place. What we were going to do was have a hand class and define a new hand for each relevant hand (namely, a user hand, a computer ai hand, a hand for cards the offense played, and a hand for the cards the defense played). As we coded we realized how inefficient having all the different hands were instead of having a single class that held a list for every "hand" that would contain the cards in each hand.

UML Diagram



## *Reflection* [~2 paragraphs]

Writing the code initially went very well; both of us were able to work together very smoothly. However, as we reached the conclusion of the project we realized we weren't fully informing each other of all of the functions we had implemented, causing needless rewriting of functions among other things (NOTE: some comments were lost in the final few commits and we did not rewrite them. The commenting was better but they had to be deleted due to weird merge conflicts). In order to improve this in the next project, we are both going to comment better s well as fully explain all methods/functions that we are implementing. Our unit testing was good and we were able to test simultaneously thanks to often commits from both people, which helped when we were working together.

We divided the work by choice; we tried to make sure that the work was divided as evenly as possible but that we worked on what we were most interested in. We divided the work and agreed upon a naming format for the initial classes/methods and as the project became more complicated we would break for short whiles to make sure our code was compatible and then one person would continue working on that code as a whole while the other went and completed mini side projects. The only issue we had while working together was the not in-depth-enough explanation of written classes/methods/variables, which we agreed to make sure we remedy in future projects.