COMS 362 Project

Iteration 2: Adding A Game

Overview

Your team's primary tasks in this iteration are to:

- complete a design for adding the game of Slapjack,
- implement a working version of the game.

You will also:

- implement two of the game's features.

Detailed instructions follow.

Details

Requirements

Slapjack is a simple card game where all players have the same view of the game. From Wikipedia:

"A 52-card deck is divided into face-down stacks as equally as possible between players. One player removes the top card of their stack and places it face-up on the playing surface within reach of all players. The players take turns doing this in a clockwise manner until a jack is placed on the pile. At this point, any and all players may attempt to slap the pile with the hand they didn't use to place the card; whoever covers the stack with his or her hand first takes the pile, shuffles it, and adds it to the bottom of their stack. If another player puts their card over the jack before it is slapped, the jack and the cards underneath can't be taken by a player until the next jack is revealed. When a player has run out of cards, they have one more chance to slap a jack and get back in the game, but if they fail, they are out. Gameplay continues with hands of this sort until one player has acquired all of the cards."

The version of the game to implement requires exactly two players. When play of the game starts there should be a deal button that either player can press to start play of the game. The title of the game should display "Slapjack". After pressing deal, 52 cards are evenly split into face down piles, one for each player. The players must take turns revealing cards from their pile. This is done by selecting the card at the top of the pile. Selected cards are place on a face up pile in the center of the table. If a Jack is revealed the first player to select to card wins the center pile, those cards are transferred to the bottom of the player's pile. When a player runs out of cards they lose and the other player wins, ending play of the game. Each player can see their score, which is a count of the number of cards in their pile.

Team Design Phase

Create a class diagram of all **relevant** aspects of the game, clearly marking in red what will be added or modified. Every new class and interface and every public method should be identified at this stage of the design. There are some implementation tasks that will depend on the completion of others before the game can be tested end-to-end. However, there should be no reason for a team member to be waiting on other team members to complete their implementation, because all public methods are fully described in the design. Implementation and unit testing should not be blocked by other tasks.

Team Planning Phase

Some of these features depend on each other and some are completely independent, as a team identify the dependencies. If the design is very clear (i.e., all of the class signatures are defined) it should be possible for everyone to begin working at the same time. Nevertheless, it may be a good idea for the faster implementors on your team to take the tasks that others depend on.

Every team member will complete two features, optional features can be skipped by smaller teams. The features are:

- How does the GameController know that a game is selected and what does it need to do before it can call match.start( )? Set up the required infrastructure for GameController to start the match.
- How does MatchController know when the game can begin and what does it need to do before it can call mainloop.play( )? Set up the required infrastructure for MatchController to start the match.
- At the start of play there should be a deal button and the title of the game should be set to "Slapjack".
- Pressing the deal button results in two even piles of shuffled face down cards. There are many simple shuffle algorithms, any is fine.
- During play, the player's alternate selecting the card at the top of their pile. Their card is placed face up on a center pile. Players are ignored if they select a card out of turn or from a pile that does not belong to them.
- When a player selects a Jack on the center pile all cards in the pile are transferred to the bottom of their pile. A player that selects a card incorrectly is ignored.
- The display of the players score always represents the number of cards in their pile.
- When a player runs out of cards the other player wins at the end of their turn. Set the title of the game to "Player X Wins".
- (optional) Extra game play rule: if a player improperly selects a center card that is not a Jack the other player wins all of the cards in the center pile.
- (optional) Extra game play rule: on each play the center pile is moved to a random location to prevent a player from hovering over the pile.

- (optional) When the game is finished show the deal button. This will be helpful.
  String remoteId = view.getRemoteId(DealButton.***kSelector***);
  view.send(**new** ShowButtonRemote(remoteId));
- (optional) When the deal button is selected for a new game, deal the existing cards (not new ones) to the two players.

Your final product will be judged on how much you follow the existing patterns and how little code you write, the less the better. There are several classes in the coms362.cards.fiftytwo package (e.g., DealButton) which are not specific to the game of 52 card pickup and can be reused in other games. Other classes can be reused by extending them and overriding a method. For example, see how the single player version of 52 card pickup only required extending two classes. If you like, you can move some of the generic elements of 52 card pickup into a different package.

Dates

First commit of the diagram and team plan due: Sunday, April 11

Final commit of diagram and code: Sunday, April 18

Individual Responsibilities (50 points)

- Complete two of the game features.

Team Responsibilities (50 points)

- Generate the design documentation (i.e., class diagram) with planed changes clearly indicated and push it to your repository. The design must follow the existing design patterns and used existing classes and interfaces as much as possible.
- Complete a short planning document detailing the tasks that need to be completed, dependencies between tasks and any other risks to the success of the project.
- It is likely that not all team members will complete their tasks, but the team must make a best effort to integrate the code that is completed into a working (demo ready) game. Being able to demo what is completed at the end of the iteration is more important than having a lot of features but no working demo.