Deck: A Card Game Language

COMS W4115 Programming Languages and Translators
Project Proposal

Cecilia Jou (cj2386@columbia.edu)
June 11, 2012

Introduction

The Deck language is a programming language designed for defining card decks and implementing card games. Deck provides the structure and data types necessary for creating these games in an easy manner. The language defines a syntax for representing various types of cards and for operating on these elements in the context of finite sets. Cards in a deck have certain traits, and these traits can be described as specific types of different features that span across the deck. For example, a standard 52-card deck has three features (suit, rank, and color) and each feature has certain types (the suit feature has four types—hearts, diamonds, clubs, and spades). With the Deck language, a user may define the card deck traits traits of a deck of cards as well as the rules to a certain game.

Motivation

The Deck language is motivated by the popular card game called Set. The real-time game requires a special deck of 81 cards. Each card is unique and on one side displays a certain combination of four features, which are count, shape, color, and shading. There are three possible types for each feature. A card can display one, two, or three of diamonds, ovals, or squiggles. The shapes can be colored red, green, or purple, with either a solid, open, or striped shading. For example, a card can display two purple striped diamonds. Or, three green open squiggle-shapes. The figure below shows three distinct cards with no features in common among them.

[1]

The game Set works by laying out twelve cards face up at once. Three-card "sets" are identified from the group and removed as quickly as possible. The player who identifies the "set" keeps the cards in his/her own hand. Then, cards from the dealer's hand are revealed and replace the removed cards. When the dealer's hand is empty, the game ends and the player with the most cards wins.

[1] http://en.wikipedia.org/wiki/File:Set-game-cards.png

A “set” is defined as three cards that for each feature, the cards are the same or all different. For example, the cards in the figure above constitute a “set” because the cards are all different for each feature. If the diamond card were purple instead, then this would not be a set, because two of the cards are purple while the other is red.

A user can write a Deck program by defining the four features of this deck of cards and the different types in each feature. The user can define the rules of this Set game, or he/she may define modified rules to create an easier or harder game. Other games that may be created with Deck include Uno and War.

Description

A Deck program requires a ‘main’ function that controls the flow of the defined game. It may call other defined or built-in functions. The language primitives help to define the card deck and the rules of the game. The Deck compiler is written in OCaml and executes the interactive text-based game in the terminal. Deck files end in the extension ‘.deck’.

Basic Structure

The ‘main’ function is required and neither takes parameters nor returns arguments. A function’s input parameters are listed within parenthesis ‘( )’ immediately following the function’s name. The return type of a function does not precede its name as it does in Java or C++, but the type of the return must be the same as the variable the caller is set to. Statements end in a semicolon ‘;’. Blocks are contained within curly braces ‘{ }’. Lists are contained within square brackets ‘[ ]’. Comments are contained within ‘/*’ and ‘*/’.

Types

Deck supports the standard basic types *int, bool,* and *string*. The following lists the specified types in the Deck language.

card
A `card` is a single primitive element.

feature
A `feature` is a category that spans all cards.

featureList
The `featureList` lists the names of the features. Each game contains one `featureList`.

type
A `type` belongs to a certain `feature`. Each `feature` has one or more `type`.

typeList
A `typeList` lists the names of the types of a given `feature`. The size of the `typeList` does not have to be the same for all features. Types listed in the `typeList` are named in increasing order of precedence.

deck
The `deck` is the collection of all cards. Each game contains one `deck`. The product of the total number of types determines the number of cards in a `deck`. That is, there is exactly one card for every possible combination of features.

pile
A `pile` is an unordered collection of cards.

hand
A `hand` is an ordered collection of cards.

player
A `player` represents a person involved in the game. Every `player` possesses a `hand`.

playerList
The `playerList` lists the names of the players.

Expressions

Deck supports the standard basic operators = (assignment), + (addition), - (subtraction), etc. Specified operators are described below.

pile add card, hand add card
Add the card to the pile or hand

pile minus card, hand minus card
Take away the card from the pile or hand

until(condition){ do; }
Keep doing until condition is satisfied. Oftentimes in card games, an action is repeated until something happens.

Built-in Functions

shuffle(hand name)
Randomize the order of cards in the hand

deal(pile name, hand , int number)
Make visible to everyone a specified number of random cards from the pile. Output the cards to console

give(hand from_name, hand to_name, int number)
Gives the number of cards from one player to another

count()
Returns number of cards in the hand or pile

query(string message)
Outputs message to console and expects input

Example Program

SetGame.deck

```
featureList = [Count, Shape, Color, Shading];
Count.typeList = [1, 2, 3];
Shape.typeList = [Diamond, Oval, Squiggle];
Color.typeList = [Green, Purple, Red];
Shading.typeList = [Open, Striped, Solid];

isSet(card one, card two, card three)
{
      bool set = false;
      for feature feat in featureList
      {
            if (one.feat.type == two.feat.type == three.feat.type) or
            (one.feat.type != two.feat.type != three.feat type)
            {
                  set = true;
            }
            else{}
      }
}

select(player pname, card one, card two, card three)
{

      if (isSet(one, two, three) == true)
      {
            pname.hand add one, two, three;
            Center minus one, two, three;
      }
      else
      {
            print("Not a set");
      }
}

main()
{
      playerList = [A, B];

      pile Center;
      pile Dealer;
      Dealer add deck;        /* Dealer starts with all cards */

      until Dealer == empty   /* Or until no more sets in Center */
      {
            deal(Dealer, 12 – Center.size());

            /* Player selects the index of the cards that make a set */
            player, int card1, int card2, int card3
                  = query("Select cards in set");
      }
}
```