COMS 4115 Project Proposal

Kevin Henrick (kph2115)

Ryan Jones (rlj2122)

Mark Micchelli (mm3710)

Hebo Yang (hy2326)

1 Project Description

Card Game Language (CGL) is a language for compiling turn-based card game variants which employ a standard 52-card deck (`http://en.wikipedia.org/wiki/Standard_52-card_deck`). CGL will make it easier to translate popular card games to a digital form than it would with general-purpose languages such as Java or C. We intend for the language to be elegant enough that even the invention of new card games will be quick and fun.

2 Motivation

Our interest in this domain developed from our perceived contrast between the widespread popularity and rich history of card games, and the lack of simple, flexible languages for describing them. Hundreds of game variants have been developed over the half-millenia that the standard deck has been used, but only a handful are commonly found on an average home computer (solitaire, poker, hearts, etc).

Although the data requirements of card games are minimal, in our case requiring only the tracking of 52 symbols and simple player information, algorithms that emerge from various rule sets can be more difficult to define with current languages. We aim to simplify this process.

3 Objectives

Our team decided to place constraints on the games considered primarily due to the time frame and our review of past card-game projects. Constraining the domain of games as previously mentioned still yields a surprising diversity of genres, including single and multiplayer games, betting and non-betting games, and stochastic and deterministic games. Games within this domain include common varieties such as Solitaire, Texas Holdem Poker, Hearts, and many others.

We aim to implement a minimum of four card game examples (Blackjack, Five-Card Poker, Solitaire, and War) and also to incorporate a simple AI framework for the non-deterministic multiplayer games (Blackjack and Five-Card Poker). If time allows, we hope to highlight the modularity of our language design through the construction of variants of these examples, by employing only minimum modifications to the original source code. For example, we could make the limit 25 before you bust in Blackjack, we could change the number of piles you draw from in Solitaire, or we could make new hands in Five-Card Poker that beat a royal flush.

4 Language Features

Each program in CGL will be a turn-based game playable with a standard 52-card deck. This class of games can be fully represented with three programming components: setup, turn actions, and win conditions. CGL will also provide easy-to-use data types commonly found in card games, such as deck, collection (i.e. hand), card, and perhaps others like suit, color, and value.

5 Program Structure

Each program in CGL will have the following layout:

```
GAME GAME_NAME
{
    SETUP
    {
        /* setup goes here */
    }

    TURN
    {
        /* turn actions go here */
    }

    WIN
    {
        /* win condition checks go here */
    }

    /* other variables, functions, and AI details go here */
}
```

6 Data Types

CGL will contain primitives for int, double, and bool. We also want to include objects for card, collection, deck, player, and possibly lower-level elements like value, suit, and color. (Were not sure yet whether or not any of these objects should be primitives in the language.) These objects will also contain default operators and functions; for example, decks will contain a shuffle() function, collections will contain a getFirst() function, and cards will contain a setVisible() function.

```
GAME NO_BETTING_FIVE_CARD_POKER
{
    SETUP
    {
    PLAYER_LIST = {player1, player2}

Deck deck = Deck.STANDARD;

Collection hand1;

Collection hand2;

int player1Wins = 0;

int player2Wins = 0;

int turnCount = 1;
    }
```

```
TURN
{
    if (turnCount <= 10)
        checkWin();
    else

    {

        deck.deal(hand1, 5);

        for Card c in hand1:

            c.setVisible(player1);

        int player1Points = 0;

        if (c == ROYAL-FLUSH)

            player1Points = 4;

        else if (c == FLUSH)

            player1Points = 3;

        else if (c == STRAIGHT)

            player1Points = 2;

        else if (c == FOUR-OF-A-KIND)

            player2Points = 1;

        deck.deal(hand2, 5);

        for Card c in hand2:

            c.setVisible(player2);

        int player2Points = 0;

if (c == ROYAL-FLUSH)

            player2Points = 4;

        else if (c == FLUSH)

            player2Points = 3;

        else if (c == STRAIGHT)

            player2Points = 2;

        else if (c == FOUR-OF-A-KIND)

            player2Points = 1;

        if (player1Points > player2Points)
```

```
            player1Wins++;

        else if (player2Points > player1Points)

            player2Wins++;

        turnCount++;
        }
    }

    WIN
    {
        if (player1Wins > player2Wins)
            player1.makeWinner()
        else if (player2Wins > player1Wins)
            player2.makeWinner()
    }

    Collection ROYAL-FLUSH = ([AC, KC, QC, JC, 10C], [AD,
KD, QD, JD, 10D], [AH, KH, QH, JH, 10H], [AS, KS, QS, JS, 10S]);

Collection FLUSH = ([*C, *C, *C, *C, *C], [*D, *D, *D,
*D, *D], [*H, *H, *H, *H, *H], [*S, *S, *S, *S, *S]);

Collection STRAIGHT = ();

for Value $v from ACE to 10:
        STRAIGHT.add([$v*, $(v+1)*, $(v+2)*, $(v+3)*, $(v+4)*]);

Collection FOUR-OF-A-KIND = ();

for Value $v from ACE to KING:

FOUR-OF-A-KIND.add([$vC, $vD, $vH, $vS, **]);
}


GAME SOLITAIRE
{
    SETUP
    {

    PLAYER_LIST = {player1};

    Deck deck = Deck.STANDARD;

    Collection stack1;

    Collection stack2;

    Collection stack3;

    Collection stack4;

    Collection stack5;

    Collection stack6;

    Collection stack7;
```

```
Collection spades;
Collection hearts;
Collection diamonds;
Collection clubs;
Collection pile;
deck.deal(stack1, 1);
deck.deal(stack2, 2);
deck.deal(stack3, 3);
deck.deal(stack4, 4);
deck.deal(stack5, 5);
deck.deal(stack6, 6);
deck.deal(stack7, 7);
deck.deal(pile, deck.getNumCards());
stack1.getTop().setVisible(player1);
stack2.getTop().setVisible(player1);
stack3.getTop().setVisible(player1);
stack4.getTop().setVisible(player1);
stack5.getTop().setVisible(player1);
stack6.getTop().setVisible(player1);
stack7.getTop().setVisible(player1);
pile.getTop().setVisible(player1);
}
```

TURN

```
{
    for Card c in stack1:
    {
        if (c.isVisible(player1))
            print( stack 1 shows: + c + \n );
    }
    /* do same for stacks 2–7 and pile */
    input i = nextLine();
    /* checks for validity go here */
    checkWin();
```

```
    }

    WIN
    {

if (spades.top == KS && hearts.top == KH

&& diamonds.top == KD && clubs.top == KC)

{
            makeWinner(player1);
        }
    }
}
```