Joker

a Card Game Programming Language
W*hitepaper*

Design and Development Crew

jge15@columbia.edu	Jeffrey Eng	*Team Leader*
jlt93@columbia.edu	Jonathan Tse	*Organizational Leader*
hhc42@columbia.edu	Howard Chu	*Team Member*
tks21@columbia.edu	Timothy SooHoo	*Team Member*

DOCUMENT HISTORY

2003 September 21 - Document Created

Joker: An Overview

Introduction

The goal of our language is to allow programmers to succinctly describe the rules of a card game and to create a runtime card game engine. The structure and the rule-driven nature of card games create a ripe domain to construct a language for. Our language creates a framework for programming any turn-based card game that uses the standard 52-card deck. Blackjack, big two, and bridge are examples of turn-based games in this domain. Spit is an example of a card game *not* in this domain.

The Language

Joker is a simple, portable language for systematically specifying multiplayer turn-based card games using the standard 52-card deck.

Simple and Intuitive

Our language is simple and intuitive. The language provides a platform for programmers to easily define and describe the structure, rules, and game flow of a card game. Our language abstracts all turn-based card game concepts and simplifies the programming mechanics involved in coding a card game. For example, the concept of “cards” and “decks” is embedded into the language as fundamental data structures. This allows programmers to focus on specifying the rules and the algorithmic components of the game flow. The syntax is designed to be programmer-friendly and intuitive. A snippet is given below (see *Code Sample*).

Portable

Our language can be run on many different platforms. We will be using ANTLR to interpret and translate syntax, and Joker code will be compiled down to Java™ bytecode, which can be run by the Java Virtual Machine, available for virtually any machine.

Memory Managed

Programmers do not need to worry about the complexity involved with memory management. Using Java as our underlying foundation, memory allocation and garbage collection is handled by the system automatically.

Language Features

Custom loops

Inherent in the concept of turn-based card games, is, well, the turn. A “turn” or a

“round” is typically implemented in other generic languages using a while or for loop. Our language understands the idea of a turn. Such a loop iterates across the players and stops when a specified special “winning condition” (in addition to other conditions) is met.

Custom data types

Our language has several specialized (and fundamental) data types unique to the domain of card games. For example: the “card” and the “deck”. Cards have values and can be compared to determine dominance. Cards can be grouped, stacked, given, taken, pushed, and popped. Decks are finite card repositories. Decks can be shuffled, have cards added to, have cards removed from.

The Runtime

The output of our compiler is a playable game in the form of our runtime: **DEALR** (Dynamically-Enabled Abstract Language Runtime), executable by the Java Virtual Machine.

The core of this executable is a rules engine, determining what actions are permissible for what player at what time. The user interface for each player will allow players to perform actions, such as drawing a card from the deck, placing down cards, taking cards from other players, and so forth. Currently, this interface is a simple text-based interface and is scalable to allow expansion to other interface models.

Code Sample

Below is a first glimpse at the Joker programming language. The following snippet defines a game where two players take turns drawing a card. The first player to draw a card better than a Jack of Spades wins.

```
// specify the suit, card hierarchy
<SUIT> := SPADE > HEART > CLUB > DIAMOND;
<CARD> := A > K > Q > J > 10 > 9 > 8 > 7 > 6 > 5 > 4 > 3 > 2;

<DECK> := [SUIT, CARD]; // specify how the deck looks like

/* 1) loop over all the players
   2) see if their card is bigger than a Jack of Spade
   3) if so, the player wins
*/
loop over @players {
   $player draw <DECK>;
   if ((current_card of $player) > SPADE.J) {
      $player wins;
   }
}
```