COMS 4115
Final Project
Card Game Language (CGL)
December 18\textsuperscript{th}, 2012

Kevin Henrick
Ryan Jones
Mark Micchelli
Hebo Yang
Overview of CGL

• CGL is a programming language used for creating and compiling turn-based card games.

• The compiler allows the creation of games that employ cards from the standard 52-card deck:
Motivation

• Why Card Games?
  - Widespread popularity
  - Rich history

• Minimal data requirements, just player info and 52 symbols, but hard to define game rules using current languages.
  - Turn order
  - Shuffling and dealing
  - Player actions
  - Complex Win Conditions

• CGL was designed to simplify encoding these requirements.
Tutorial Introduction to CGL

A CGL program is defined using four types of blocks:

- PLAYER{}
- SETUP{}
- TURN 1{}
- TURN n{}
- WIN{}
**SETUP { } Block**

- The only mandatory block.

- Runs immediately after an optional PLAYER { } block, and serves as the entry point into the program.

- Global declarations of variables and functions.
  - Function declarations **ONLY** in and at beginning of SETUP { }.

- Never runs again after initial termination.
/* This setup block declares two players, sets out the player order, creates a standard deck, shuffles it, and finally calls the turn function on the first player. */

SETUP
{
string name1 = scan();
string name2 = scan();
player p1 = <name1, 1>;
player p2 = <name2, 1>;
p1.next = p2;
p2.next = p1;
list deck = STANDARD;
deck = shuffle(deck);
turn(p1);
}
PLAYER {} Block

- Defines data structure for all players
  (defaults of name: String, turnID: Int)
- Optional, but necessary for most non-trivial games
- Player data accessed through p.varName where p is a player reference.

CGL Source Code:

```cgl
/* This gives each player in the game a score, a turn count, and a next player */

PLAYER
{
    int score = 0;
    int turnCount = 0;
    player next = NEMO;
}
```
TURN n {}

- Describes game rules and player strategy (both human and AI)

- Examples:
  - query human player for move
  - conservative AI agent’s logic
  - aggressive AI agent’s logic

- Has access to the current player through the “your” keyword

```python
turn(player p) : your = p
```
/* If the top card of the deck is a red card, give the player a point. Then, put the card on the bottom of the deck. If the player has moved five times, move to the win block. */

TURN 1
{
if (your.turnCount >= 5)
  win();

card c = <- deck;
print(your.name ^ " drew " ^ intToString(value(c)) ^ suit(c) ^ "\n");
if (c == $*D || c == $*H)
  your.score = your.score + 1;
print(your.name ^ "'s score is " ^ intToString(your.score) ^ "\n");
deck <+ c;
your.turnCount = your.turnCount + 1;
turn(your.next);
}
The `WIN {}` block is used to check win conditions and terminate the program.

This block runs whenever the `win()` function is called.

It has access to each player reference, global variables, and functions.

Unlike `TURN n {}`, there is no current player reference.
WIN { }

CGL Source Code:

/* Tests to see which player drew more red cards, and declares that player the winner. */

WIN
{
if (p1.score > p2.score)
print(p1.name ^ " wins\n");
else if (p1.score < p2.score)
print(p2.name ^ " wins\n");
else
print("draw\n");
}
Example 1: High-Low

CGL Program:

the first card has value 10
will the next card be (h)igher or (l)ower?  

l

new card's value is 2
correct prediction
will the next card be (h)igher or (l)ower? 

h

new card's value is 8
correct prediction
will the next card be (h)igher or (l)ower? 

h

new card's value is 5
incorrect prediction; game over

total score = 2
Example 2: Black-Jack

Setup Stage:

Please enter Player name
Professor Edwards
Please enter 1 if human, or 2 if AI
1
Please enter Player name
Mark
Please enter 1 if human, or 2 if AI
1
Please enter Player name
Kevin
Please enter 1 if human, or 2 if AI
1
Please enter Player name
Dealer
Please enter 1 if human, or 2 if AI
2
Example 2: Black-Jack

Gameplay Stage:

Kevin’s turn; press enter to continue
you have KD 4H
type "h" for hit; anything else for stay
h
you got a 2S
Kevin's turn; press enter to continue
you have KD 4H 2S
type "h" for hit; anything else for stay
h
you got a 3H
you have KD 4H 2S 3H
Type “h” for hit; anything else for stay
s

Professor Edwards scored 21
Mark scored 16
Kevin scored 19
Dealer scored 0
Professor Edwards wins
How CGL was Implemented

- **OCAML** –
  1) Scanner.mll (ocamllex), parser.mly (ocamlyacc), ast.mli
  2) generator.ml, corelibrary.ml, javalibrary.ml, cgl.ml
  3) sast.mli, semantic_analyzer.ml

- **JAVA** –
  Main.java, CGLLList.java, Card.java, Player.java.

- **CGL** –
  Unix commands to compile and run:
  ```
  $ .cgl/ -j source.cgl
  $ javac *.java
  $ java Main
  ```
Flow of Control / Dependencies
Roles and Responsibilities

- Kevin Henrick (Team Leader) – Semantic Analyzer / SAST, test cases, and Makefile.
- Ryan Jones – Semantic Analyzer / SAST, test cases, CGL Executable and Makefile.
- Mark Micchelli – Scanner, Parser, Abstract Syntax Tree, Generator, CGL Executable, and Makefile.
- Hebo Yang – Test cases, and Bash Script.
CGL Games Created

• Finding the First Ace – Kevin Henrick and Hebo Yang

• RedCard – Mark Micchelli

• HighLow – Mark Micchelli

• Blackjack – Mark Micchelli
Summary of the Project

• We implemented almost all of the LRM, our original conception of the language.

• Future Work: external libraries, more complete semantic analysis, more options in the executable, and bug fixes.
Lessons Learned

• Don’t be afraid to change design choices at the last minute.

• Try to keep all of the parts moving at once.

• Prioritizing starting early was really beneficial!