CGDL: Card Game Description Language

Angela Wei – Project Manager

Abraham Tseng – Language Guru

Raghavan Santhanam – System Architect

Kshitij Bhardwaj – System Integrator

Deepak Nayak – Tester/Validator

What is CGDL?

- For card game developers and hobbyists
- Idea for a new card game?
  - Create an interactive prototype
  - Sell it to a gaming company
- OR just code for fun and show off..


JOKER
CGDL is:
Simple
Familiar
Specialized
Flexible
Interactive
Portable

Speaking of showing off…
It's demo time…

A Crash Course in CGDL

- Primitive data types:
  - number, string, bool, attribute, visibility
- Data structures:
  - Sets (eg. Card[ ])
  - Records
- Conditionals:
  - if, else if, else, switch/case

A Crash Course in CGDL

- ** This is a comment **
- C-like functions
- Loops:
  - forEach x in set
  - loop i in n
  - repeat/until

User I/O

- message(string)
- Query
  - string queryString(string)
  - number queryNumber(string)
  - Card[ ] querySelection(player, string)
- string
  choose("Please choose","c1 c2 c3")

Specialized Class: Player

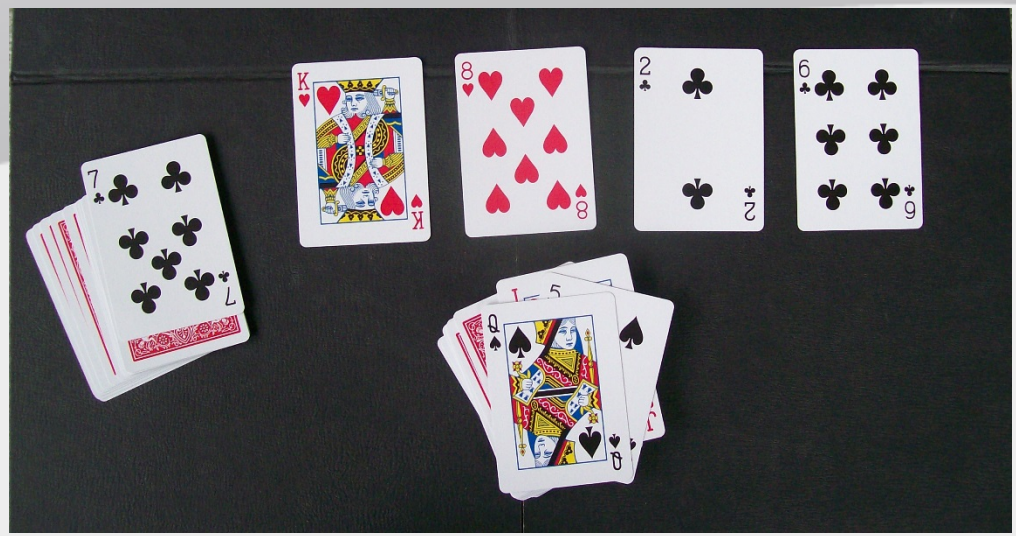
- Extensible fields of any type
- Hand pile
- currPlayer

Specialized Class: Card

- Modifiable attributes
- Tracks association
- Standard deck included

Specialized Class: Pile

- Pile is a set of cards with properties inspired by real card games
- Built in functions



Game Structure

- "Rounds within rounds"

```
Game innerGame {
        Setup {
                ...
        }
        Round {
                ...
        }
}
```

Hello Ace of Spades

addCardAttribute suit

addCardAttribute rank

```
Game main {
        Setup {
                …
        }
        Round {
                …
        }
```

Setup: In Depth

- Setup {

```
addToDeck(STD_SUIT, STD_RANK);
        deckPile.shuffle();
        numPlayers = 1;
        Player player1 = players[0];
        player1.hand.visible = self;
}
```

Round: In Depth

```
Round {
        currPlayer = player1;
        Card card = deckPile.getFromTop();
        player1.hand.putAtFront(card);
        if (card.rank == ACE and
                card.suit == SPADE)
        {
                winner = player1;
                message("Hello Ace of Spades!");
        }
}
```

System Architecture

CGDLC

Input: Card Game Description language (CGDL) source program: **game.cgdl**(Character stream)

cgdl.l(Lexical Analyzer)

Token stream

cgdl.y(Syntax Analyzer)

Parsed and matched pattern in the token stream

semantic_actions.c

Addition of individual nodes of Abstract Syntax Tree(AST)

ast_builder_and_walk_initiator.c

AST

Processes AST nodes and does code-generation if semantic-analysis passes

ast_translator.c ⟷ symbol_table.c

Generates C++ code with the quintessential CrashHandler and the CGDL source line information to ease the debugging of the generated game when needed.

Output/Target: Card Game C++ code: **game.cpp**

System Architecture

game.cpp + cpplib

C++ Compiler

Final Card
Game Executable:
game

Software Development Environment

- Source code version control: Git
- Development Language: C
- Lexical analysis: Lex
- Syntax analysis and Parsing: Yacc
- Target language: C++
- Makefile and bash scripts

Code Organization

- Root directories:
  - /examples: *.cgdl files
  - /cpplib: library header files in C++
  - /kernel: Source code of compiler
  - /test: test suites

Runtime Environment

- Built-in library functions control the runtime behavior
  - Card, Player, Pile structs with attributes and built-in functions
  - Dynamically configure attributes of structs
    - Dynamic.h : generated during compilation of cgdl code
  - I/O functions to display messages and UI

.cgdl to game executable

- Copy your cgdl file to /examples
- In /kernel, ‘make’ to build compiler
- Run script ‘buildGame.sh’ with cgdl file as argument
- Game executable is created in /examples
- Start playing!!

Testing

- Test Plan Evolution
  - Phase I: Manual testing of lex and yacc
  - Phase II: Some small toy test cases to unit test grammar production and AST
  - Phase III: Same toy test cases from Phase II along with some real sample game used during code generation and semantic analysis to do integration testing

(continued…)

Testing

- Phase IV: Test script to automate testing from source cgdl to executable, and also for regression
- Issues Tracking
  - Shared Google Spreadsheet

Testing

```
        ------------------------------------
        congratulation your game ./declaration/decln_5 is ready
==========================================
compiling ./declaration/decln_3.cgdl
==========================================
        cgdl compile passed
        reference file ./declaration/decln_3.cppref exist
        regression passed
        ------------------------------------
        compiling ./declaration/decln_3.cpp
        ------------------------------------
        congratulation your game ./declaration/decln_3 is ready
==========================================
compiling ./declaration/decln_1.cgdl
==========================================
        cgdl compile passed
        reference file ./declaration/decln_1.cppref exist
        regression passed
        ------------------------------------
        compiling ./declaration/decln_1.cpp
        ------------------------------------
        congratulation your game ./declaration/decln_1 is ready
==========================================
compiling ./declaration/decln_4.cgdl
==========================================
        cgdl compile passed
        reference file ./declaration/decln_4.cppref exist
        regression passed
        ------------------------------------
        compiling ./declaration/decln_4.cpp
        ------------------------------------
        congratulation your game ./declaration/decln_4 is ready
==========================================
compiling ./cpperror/decln_5.cgdl
==========================================
        cgdl compile passed
        reference file ./cpperror/decln_5.cppref exist
        regression passed
        ------------------------------------
        compiling ./cpperror/decln_5.cpp
        ------------------------------------
```

Final Demo

CGDL UNLEASHED!

Lessons Learned

- What went right:
  - Frequent meetings, coding together
  - Small kernel first
  - C, C++ with lex and yacc
  - Respecting teammates
- Improvements
  - Use Git earlier on
  - More tests from the beginning

Conclusion

- CGDL is:
  - Flexible game creation
  - Easy to Learn
- Great learning experience

Source Code:
https://bitbucket.org/flyawei/plt-project/