Team 19: Ke Liao(kl2735) Yue Huang(yh2640) Liyuan Zheng(lz2375) Chaozhong Lian(cl3190) Songqiao Su(ss4555)

Language Reference Manual

Table of Contents

Introduction

This manual describes the numerous features that GameWizard supplies to create on-line RPG card game. We start with an overview of the lexical conventions used within the language, follow with the language syntax, and end with a grammar to represent GameWizard.

Keywords

The following identifiers are reserved for use as keywords, and may not be used otherwise:

if, else, for, foreach, while, switch, case,
init, round, roundbegin, roundend, dying, close,
define, cards, groups, characters, game

Reserved

1. Built-in type names

int, double, string, boolean, Card, Player, Character, Group

2. Built-in global function names

wait_card
wait_input
getOtherPlayer
shuffle

3. global variables

PlayerList
CardList
CharacterList
GroupList
CardStack
DiscardedCardStack

4. Reserved Characters

The following characters are reserved for use in the grammar
* / [ ]

= ; : ==
> < >= ,
|| != . +
- ( ) {
} <= && $

Built-in Global Functions

boolean wait_card(Palyer,Card)
Wait for a specific player to put a specific card. If that player finally give that card within the given time, return true, else return false.

void wait_input()
Wait for the player who currently in turn to give instruction.

List<Player> getOtherPlayer()
Get a list of player exclude the one whom currently in turn.

void shuffle(List)
shuffle any given list

Types

Primitive types

int: 128 bits in size. Same implementation in Java 7.
double: floating numbers are the same as the implementation in Java 7.
boolean: a boolean encapsulates true and false values.
string: A string is a sequence of characters surrounded by double quotation marks.

Character(or char) is not one of primitive types. There is only string with only one character.
Certain character in string must be represented with escape sequences.
newline \n
horizontal tab \t
carriage return \r
backslash \\
single quote \’
double quote \”

Built-in types

GameWizard provide some basic types of elements to construct an on-line RPG game.

```
Card{
        void method(Player)
}
```

Literally a very basic element in the game, players can use card for different things.

```
Player{
        void HPadd(int)         %%Add player's HP if not reach the upper bound
        void HPdrop(int)        %%Drop player's HP
        void setCharacter()     %%Randomly set a character for the player
        void setHandCards(List<Card>) %%Let a list of cards become the hand card
of the player
        void setGroup()         %%Randomly set the group of the player
    Player chooseTarget()       %%Wait the player to choose a target player
    void getCards(int)          %%Get the card for the player from the card stack
    List<Card> putCards(int)%%Put certain number of card
    List<Card> dropCards(int)%%Drop certain number of card
    Card dropTheCard(string)%%drop the card specified by the string
    void useSkill()             %%Let the player calls the character's skill
    void win()                  %%Mark the player as winner
    boolean isInGroup(Group)%%Judge whether a player belongs to one certain
group
}
%%Player type stands for a unique identifier for the player who enrolled in the game.
Character{
}
%%Each player can act as different characters, which render different skills for the
player in the game.

Group{
        void win()              %%Set the certain group of players to win
}
%%A group of players are those who have the same winning condition.
```

Extended types

List

A list is an ordered data structure that holds zero or more items. A list can be initialized with a comma-separated list of items surrounded by '[' and ']'. List elements can be accessed using the get() method. Additional items can be added and removed with the append() and delete() methods respectively. Lists are dynamically sized as it grows and shrinks, limited only by the machine's available memory.

Dict
A dictionary is mutable and is a container type that can store any number of GameWizzard objects. Dictionaries consist of pairs (called items) of keys and their corresponding values, dictionaries are also known as associative arrays or hash tables.

Scope

Lexical Scope
Besides all the global variables provided by the system, all programmer-defined variables takes effect from the end of its declaration to the end of a block, normally ends by a closing curly brace. Variable names in the same scope must be unique.

Global Scope
Several useful global variables and functions are provided for the programmers. Global variables exist within all functions and persist through the entirety
of the program. Programmers cannot declare their own global scoped variables, all user-declared variables are within some logic block.

Statement Block Scope
Variables declared inside a statement block persist within the block and dies upon exiting the block.

Function Scope
For variables declared inside a function's statement block, rules for Statement Block Scope apply. Moreover, variables present in parameters list are also in scope of the statement block of the function.

Constants(Values)

There are four types of contants supported by GameWizard.

String:	string surrounded by quotation marks
Number:	integer or float number in decimal
Boolean:	'true' or 'false'

In the Grammar section, we use Value to represent them.

Operators

Multiplicative: *, /, %
Addictive: +, -
Relational: ==, !=, >=, >, <, <=
Logistic:
    Not: !
    And: &&
    Or: ||

Expressions

GameWizard has two kinds of expressions, which are Arithmetic Expressions and Conditional Expressions. The will be represent as Arith-expression and Cond-expression respectively in the Grammar section.

Arithmetical Expressions

For simplicity, we just use 'E' to represent an Arithmetical Expression here.
E:
    E Addictive-Op T
    T
T:
    F
    T Multiplicative F
F:
    ID
    Number
    Function-call
    '(' E ')'

Conditional Expressions

For simplicity, we just use 'E' to represent an Conditional Expression here.
E:
    logical-OR-expression
logical-OR-expression:
    logical-AND-expression
    logical-OR-expression Or logical-AND-expression
logical-AND-expression:
    logical-AND-expression And logical-term
    logical-term
logical-term:
    Not logical-term

Boolean
Function-call
Arith-expression Relational-Op Arith-expression
‘(’ E ‘)’

Grammar

Source-code :
Game-config Components-config Procedures-List

Game-config:
‘define’ ‘game’ Json

Components-config:
Cards-definition Characters-definition [Groups-definition]

Cards-definition:
‘define’ ‘cards’ Config-List

Characters-definition:
‘define’ ‘characters’ Config-List

Groups-definition:
‘define’ ‘groups’ Config-List

Procedures-List:
Procedure-definition
Procedures-List Procedure-definition

/*
The source code in GameWizard has a skeleton defined above. It configures the basics of a game in a Json, and then defines sets of cards, characters and groups respectively in a Config-List, at last, defines procedures.
*/

Config-List:
‘[’ Config-List-content ‘]’

Config-List-content:
Config
Config-List-content ’;’ Config

Config:
    ID Json

Json:
    ‘{’ Json-content ‘}’

Json-content:
    Json-item
    Json-content ‘;’ Json-item

Json-item:
    ID ‘:’ Value
    ID ‘:’ Config-List
    Function-definition

/*
    Config-List is supported by GameWizard to define fancy sets of characters, cards and groups for a RPG card games. Config-List is list of Configs, which are pairs of IDs and Jsons, in which ID becomes identifier(name) of a character, card or group while Json defines the attributes and methods of it.
    Json consists of lists of key-value pairs and/or functions.
    For characters, we usually put key ‘hp’ with an integer value as max-HP of the character and key ‘skill’ with a Config-List value as its skills list within their Json. For cards and skills, we usually put function-definition of ‘method’ in their Json to define the effect of cards or skills. (For details please the Tutorial)

*/

List:
    ‘[’ List-content ‘]’

List-content:
    List-item
    List-content ‘;’ Item

Item:
    ID
    Value

Function-definition:
    Function-signature Statements-block

Function-signature:

Type ID ‘(’ [Parameters-config-list] ’)’
Built-In-Function-Signature

Parameters-config-list:
Parameters-config
Parameters-config-list ‘,’ Parameters-config

Parameters-config:
Type ID

Functions-list:
Function-definition
Functions-list Function-definition

Procedure-definition:
Function-definition
Round-procedure-definition

Round-procedure:
‘round’ ‘{’ Statement-list Function-list ‘}’

Built-In-Function-Signature
‘method’
‘init’
‘roundbegin’
‘turn’
‘roundend’
‘dying’
‘close’

/*
The function definition in GameWizard are very similar to that in Java, which consists of function signature and statements block. The function signature defines the return type, function name(ID), and parameters list of the function. The statements block defines the actions of the function call.
There are several built-in functions whose signatures do not need to be complete in GameWizard. For them, the name alone will suffice.
*/

Statement-list:
Statement
Statement-list ‘;’ Statement

Statement:

Assignment
Function-call
Selection-Statement
Loop-Statement
Statement-block

Assignment:
ID '=' Expression

Function-call:
ID '(' [Parameters-list] ')'
ID '.' ID '(' [Parameters-list] ')'

Parameters-list:
ID
Parameters-list, ID

Selection-Statement:
'if' '(' Condition-expression ')' Statements-block ['else' Statements-block]

Loop-Statement:
'for' '(' [Assignment] ';' Condition-expression ';' Assignment ')' Statements-block
'foreach' '(' ID 'in' ID ')' Statements-block
'while' '(' Condition-expression ')' Statements-block

Statements-block:
'{' Statement-list '}'

Expression:
Cond-expression
Arith-expression