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

Language Tutorial

Introduction

This will be a short tutorial showing how to build an online RPG-card game using our GameWizard language. We will go through example programs that demonstrate the use of GameWizard.

This tutorial focuses on the basics of our language – configuration, types, constants, expressions, built-in functions. The aim of this tutorial is to provide a guide so that the programmers can start to write interesting simple games in GameWizard very quickly.

First, let's define the RPG-card game supported by GameWizard. RPG stands for Role-Playing Game, in which each player plays a role, with attributes like HP and skills. In our RPG-card games, players make moves when they are prompted to(there is no preemptive moves). Usually the move can be putting a card or use a skill. One good example is "bang!", a wild-west-themed card game, in which each player will pick a character like Bart Cassidy, Calamity Janet or others. Every card has its effects. For example, if a player, in his turn, puts a "Bang!" card and specifies a target(another player), the HP (Health Point) of the target should be deducted by one if he does not put a "Missed!" card. Each character has its own skills. Take "Sid Ketchum" in "Bang!" as an example, who can discard 2 cards from his hand to regain one life point(HP).

The game is to be played online, which means several players in the same LAN can connect to one server to play the game together using their own computer. The network support part will be taken care of by the compiler. Programmers using GameWizard only needs to configures the listening port of server.

GameWizard is to provide an easy way to design a RPG-card game. However, other simple games like "rock paper scissor" can also be implemented with GameWizard. With the help of various build-in functions of GameWizard, programmers can design diverse RPG-card game very easily.

Before we show you a Hello World program, a quick word on input and output of GameWizard.

Input and Output

GameWizard is to create RPG-card games which people can play online. The output of a GameWizard compiler is two java programs. The first one is the server program, which keeps listening on a specified port. The second one is the client program, which communicates with the server program to get status of the game or upload players' actions. Since the logics of the game will be defined in the server program, the client program is usually small and easy to spread.

Section Defination

There are 5 basic sections in the GameWizard program: Game-config, Cards-definition, Characters-definition, Groups-definition and Procedure-List.

Game-config

The definition of "game" will be here as several configurations, which usually include the name of our game, the number of players and also the port the server program will be listening on.

Cards-definition

In this section, we will define a list of cards, each of which usually can have a name, a value and/or a function that specifies the effect of the card in our game.

Characters-definition

The most interesting part of a role-play game is the setting of characters. You may want the warriors have strong blade skills, and the magicians have powerful magic to use. GameWizard can provide you a simple way to define and build skills for each character.

Groups-definition

We could group players into groups, each of which can have its own attributes and name.

Procedures-List

In this section, we define a list of procedures in a game.

As programmer can freely design various procedures, a few of them are very essential to a RPG-card game and we "require" programmers to include them(otherwise the compiler will take it as that there is a default empty procedure there). The "required" procedures include init, round and close.

Init

Generally, we assign characters to each player and initialize the CardStack where players get cards.

Round

Our RPG-card games will be played on rounds. The procedures for the beginning of rounds can be defined in “roundbegin”. Each player will have its own turn in a round and the procedure of a player’s turn can be defined in “turn”, which might include letting the player draw certain number of cards from draw pile and prompting the player to put some cards. “roundend” will be the place where procedure at the end of a round is defined.

Close

A procedure at the end of a game.

Now let’s look at some examples.

Sample 1: Empty Game (minimal)

```
define game{
        name: “Empty Game”;
        num_of_players: 6;
        server_listening_port: 4115;
}
define cards[
         myOwnCard{

         }
]
define characters[
         whateverCharacter{
                  hp: 2;
         }
]
%%
Init{}
round{
          turn{}
          roundend{     //this part is optional
                  PlayerList[0].win();
          }
}
close{}
```

This program will be compiled into a java server program and a java client program. The server program will set up a game and listen on the port configured. Players need to run the client program to connect to the server and join the game. The server program will display the IP of the host while players need to input the IP address of the server in the clients. In this game, after one round, the first player wins.

Now, a simple Hello World program to demonstrate how GameWizard example codes work.

Hello World

This hello world program creates an online card game that requires 2 people to play. There are only two cards in this game (0, 1). Each person plays for a normal character (initially hp=3, no skill) get a card at the beginning of his turn, and he has to put a card in this turn. If the card he puts is 0, then his hp will be deducted by 1. If the card is 1, then nothing happens. The game ends when one of the player' hp turning to be zero. This is a very simple game, which is a little bit more interesting than the above one.

Sample 2: Hello Game

```
define game{
        name: "Hello Game";
        num_of_players: 2;
        server_listening_port: 4115;
}
define cards[
        card0{
                value: 0;
        }
        card1{
                value: 1;
        }
]
define characters[
        normal_cha{
                hp: 3
        }
]
%% //configuration part ends, rule part starts
init{
        foreach(player in PlayerList){
                player.setCharacter(CharacterList[0]);   //initialize the character of players
```

```
        }
        CardStack = [card0]*1000+[card1]*1000;       //initialize the Card Stack
        shuffle(CardStack);
}
round{
        Roundbegin{}  //before going through the turn of the first player (optional)
        Turn{  //specific player's turn
                player.getCards(1);    //obtain this player, make getCards() call
                List<Card> cards = player.dropCards(1); //put cards
                If(cards[0].value==0){   //check which card it is
                        player.HPdrop(1);
                }
        }
        Roundend{}  //after going through turns of all players (optional)
}
dying(Player player){  // a player might be dead, a good place to check if the game ends.
This function will be automatically called when hp of a player becomes zero.
        List<Player> list = player.getOtherPlayer();
        list[0].win();  //end the game

}
close{}
```

Let's explain what is going on here.

In the first half we configure our game. We configure game (name, num_of_players, port), cards (card0, card1) and characters (normal_cha).

In the second half, we design the rules of the game and how the game goes on. "init{}" is the block in which we can design that at the beginning of the game a character is assigned to each player, and the CardStack is initialized. CardStack is a global List object to represent the draw file, where getCards() function let players to draw cards from.

Inside of "round{}", we have "roundbegin", "turn" and "roundend". Each round has one turn for each player. Inside the turn, we design what is going to happen for each player. In this case, the player, at his turn, first get one card from CardStack, and the put the card, if the card is card0, he loses one hp. Same thing happens for each player in a round in turn. Rounds will keep going until one player's hp get decreased to zero.

Once the hp of a player turns to zero, then this player loses and the other wins as we defined in dying() function. dying() will be called automatically whenever hp of a player turn to zero.

For the last part, we use "close{}" to denote the source program ends.

To compile this program, use the command
./gamewizard hellogame.gw

If everything goes as expected, once the program is compiled, you will find two new files in the current directory, whose names are “hellogame-server.java” and “hellogame-client.java”.

This java file can be compiled (javac) and run on a server host. Your friends in the same LAN can use the client program to connect to the server, and play the game together using command line.

Enjoy yourself with the game using command line. Try not to lose yourself in the game, or lose yourself after reading the rest of our tutorial.

Other sample codes:

Sample program 3: Rock-paper-scissors

```
define game{
        name : “rock_scissors_paper”;
        num_of_players : 2;
        server_listening_port: 4115;
}

define cards[
        rock{
                value: 1;
        },
        scissors{
                value: 2;
        },
        paper{
                value :3;
        }
]

define characters[
        normal_character{
                hp: 1;
        }
]

init{
        foreach(player in PlayerList){
                player.setCharacter(normal_character);
```

```
        }
        CardStack = [rock]*3 + [scissors]*3+[paper]*4;
        shuffle(CardStack);
}


round{
        roundbegin{
                map = new Dict<Card, List<Player>>();
        }

        turn{
                player.getCard(1);
                map[player.putCard(1)].append(player);
        }


        roundend{
                        card1 = map.keys[0];   //get the first key
                        card2 = map.keys[1];   //get the second key
                        if( (card1.value – card2.value + 3) % 3 == 1 )
                                win(map[card2]);  //this set of players win

                        else if ( (card1.value – card2.value + 3) % 3 == 2 )
                                win(map[card1]);

        }//roundend
}
close{}
```

In this rock-paper-scissors game, we have 10 cards for two players to draw. In each turn a player will draw and play a card. By comparing the value of card at the end of a round, we can know which player can win.

Sample 4: Tiny War

In this sample, we have 3 kinds of cards: “attack”, “defense” and “barbarian invading”. Also, 4 kinds of characters: “merchants”, ”warriors”, “priest” and “magician”. We have two teams in this game, the “Red” team and the “Blue” team. “Red” will win if all players in the “Blue” team have their hp decreased to zero, and vice versa.

```
define game{
        name:”Tiny_War”;
        num_of_players: 8;
        server_listening_port: 4115;
```

```
}

define cards[
        attack{
                method(Player dealer){
                        Player target = dealer.chooseTarget();
                        if ( ! wait_card(target, defense) ){
                                target.HPdrop(1);
                        }
                }
        },
// When a player play a “attack” card to a target player, the target player must play a
defense card, otherwise the target player will lose 1 hp.

        defense{
                method(Player dealer){
                }
        },
//defense card only can be used when someone attacks you.


        Barbarian_invading{
                method(Player dealer){
                        playerList target = dealer.getOtherPlayers();
                        foreach(player in target){
                                if ( ! wait_card(player, defense) ){
                                        player.HPdrop(1);
                                }
                        }
                }
        }
// When a player plays a barbarian_ invading card, every other player must put a
defense card to resist the damage.
]


define characters[
        Merchant{
                hp: 4;
                skill: [
                        mercy{
                                method(Player dealer) {
                                        target = chooseTarget();
                                        cards_list = dealer.dropCards();
                                        target.getCards(cards_list);
                                        if (cards_list.size() >=2 ){
                                                dealer.HPadd(1);
                                        }
                                }
```

```
                }
            ]
        },
// Merchant can give 2 of his cards to other player and restore 1 hp

        Warrior{
            hp:5;
            skill: [
                charge{
                    method(Player dealer) {
                        if(wait(dealer, attack)){
                            target = chooseTarget();
                            if(!wait(target,defense))
                            {
                                target.HPdrop(1);
                            }

                        }
                    }
                }
            ]
        },

//Warrior can use his skill as a attack card to attack a player each turn.

        Priest{
            hp:3;
            skill: [
                heal{
                    method(Player dealer) {
                        target = chooseTarget();
                        dealer.dropCard(1);
                        target.HPadd(1);
                    }
                }
            ]
        },
//Priest can heal a player by 1 hp, but need to drop a card.

        Magician{
            hp:3;
            skill: [
                fireball{
                    method(Player dealer) {
                        target = chooseTarget();
                        dealer.dropCard(1);
                        target.HPdrop(1);
                    }
                }
```

```
                ]
        }
//Magician can use fireball to attack a player by 1 hp, but need to drop a card.

]

define groups[
        Red{
        },
        Blue{
        }
]

init{
        foreach(player in PlayerList){
                player.setCharacter(CharacterList[random(1,4)]);
                player.setGroup(GroupList[random(1,2)]);
        }
        CardStack = [attack]*40 + [defense]*30+[babarian_invading]*10;
        shuffle(CardStack);
}

round{
turn{
        player.getCard(2);
        boolean usedSkill = false;
        boolean isEnd = false;
                while(!isEnd){
                        input = wait_player(player)
                        switch( input ){
                                case “skill”:
                                        if( !usedSkill ){
                                                usedSkill = true;
                                                player.useSkill();
                                        }
                                        break;
                                case “end”:
isEnd = true;
                                        break;
                                default:
                                        card = player.dropTheCard(input).
                                        card.method(player);
                        }
                }
                //players can play cards and use skills before they end their turn.
}
}
dying(Player player){
        // check condition if the game ends
```

```
        foreach( p in PlayerList ){
                if(p.isInGroup(player.group) && p.hp != 0){  //if there is someone alive
in this group
                        return false;
        }
        }
        if(p.group == “Blue”){   //all people in this group die
                Red.win();
        }else{
                Blue.win();
        }
        return true;
}
close{}
```

Conclusion

This tutorial covers the core of GameWizard and hopefully it will assist users to write their own online RPG-card games. It shows the basic of grammar of the language. A more detail language definition is included in Language Reference Manual.