Saint Petersburg State University of Information Technologies, Mechanics and Optics

Department of Computer Technologies

V.V. Vedeneev, P.S. Solovjev

The Control System of “Zavalinka” Text Game

Version 0.1b

Programming with explicit state selection

Project Documentation

Saint Petersburg
2003

Contents

Introduction

SWITCH-technology has been proposed by A.A. Shalyto for the algorithmization and programming of logic control. This technology was further developed by the author together with N.I. Tukkel and applied to the development of the event and object-oriented programs. For more details about this technology and different examples of its usage see the following Web sites: http://is.ifmo.ru and http://www.softcraft.ru.

This technology is convenient for technical objects control problems. In the current work it is used for the IRC text game management.

The game "Zavalinka" is developed for the IRC (Internet Relay Chat). Similar game was developed earlier, but at that time SWITCH-technology wasn't used (the game "Quiz", server campus.ifmo.ru:6667, channel #womend). The goal of this work is to concentrate control logic in the program code, to improve its readability and to create the proper documentation. Finite state machine is used to describe the logic of the game.

Perl is chosen as the programming language, because it is convenient for text processing. Perl doesn't have the *switch* operator, so the state-machine is implemented with the *if-else* construction.

Regular expressions are used for text processing in Perl. They are also implemented with finite state machine. Perl programs are generally platform-independent.

1. Target Setting

The aim of the work is to create the program for "Zavalinka" game management. The rules of this game are similar to the rules of the verbal game with the same name.

There are several Internet sites where the rules of the game are described. You can find the rules at the following site:

http://ng.perm.ru/bin/kniga/konkurs/zavalina/zaval_1.htm.

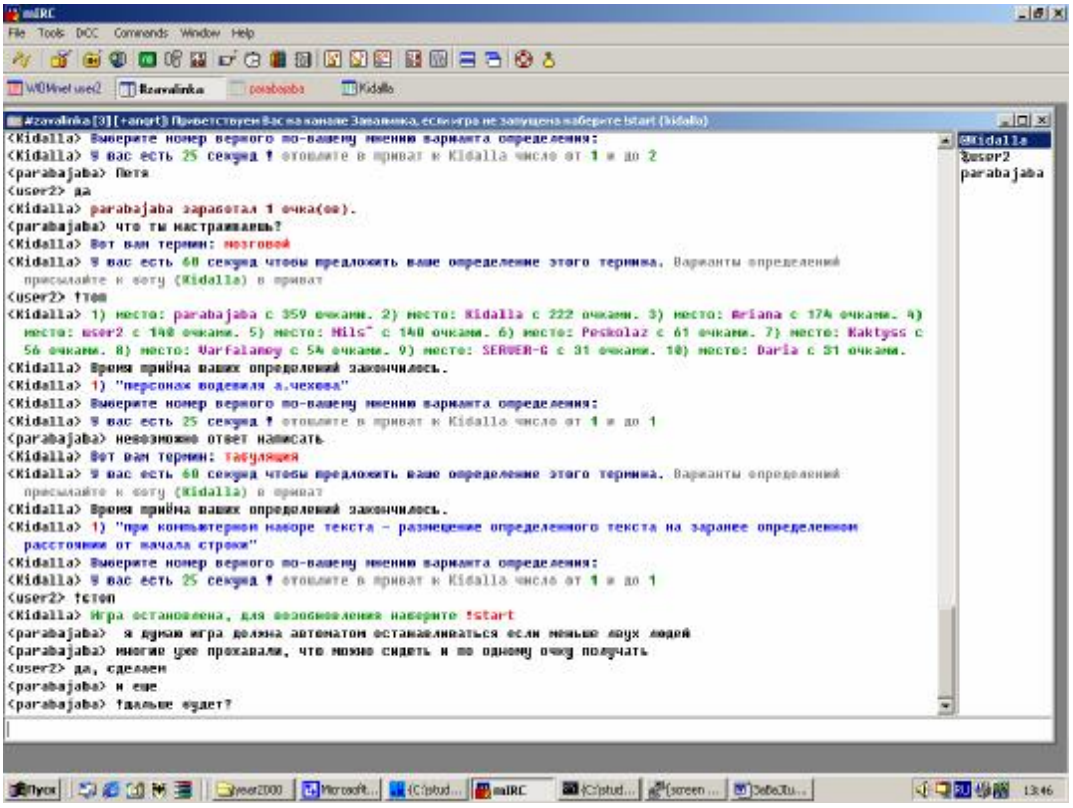
The rules used in the current work follow:

Game rules. Quizmaster (bot) gives a term to the players (term is an unintelligible word). During some predefined period of time the players should think of the possible definitions of this term. The definitions should be convincing for other players, to make them choose one of these definitions. The players tell their versions of definitions secretly to the quizmaster. After that bot tells all given versions with the original one and every player should choose the most suitable one.

Score calculation. The player who has guessed the right definition gets one point. Besides, the players get points if their definitions were chosen by other players. (2 points for every player). The quizmaster calculates all scores after each turn and announces the results.

The game is managed by the game bot. The current work describes the development of this bot. The project was called “Zavalinka” after the name of the verbal game with same rules.

Pic. 1. The screenshot of the game.



Pic.1. The screenshot of the game

2. The description of automata-based approach

The program is developed for the event-oriented IRC-system. The automata-based approach, described below is used.

1. The analysis of the problem is performed, set of the events and input variables are designed.

2. One automaton or system of interconnected automata is designed.

3. The verbal description (the list of the tasks, etc) is developed for each automaton.

4. Each event is described in the verbal form. Conditions, that cause the occurrence of the event, and some notes for processing should be given.

5. The diagram of connections is designed for each automaton, in case there are several automata – the diagram of their interactions, too.

6. The transition graph is designed for each automaton.

7. First, the template with “stubs” is created. It is isomorphic to transition graph. After checking, that it properly corresponds to the transition graph, the stubs are changed with functions that implement output actions, for example, “score calculation” output action.

3. Event Definition

The set of the following events has been designed according to the analysis of the problem.

e1	–	Game started
e2	–	Switched to sleep mode
e3	–	Started receiving the definitions
e4	–	Finished receiving the definitions
e5	–	The players have made their choice
e6	–	The player has sent a message

3.1. Event “Game started” (e1)

The event occurs when the program has been started or the command “!start” has been received from one of the players.

During the processing of this event, bot executes some initiating actions that are necessary for the successfull start.

3.2. Event “Switched to sleep mode” (e2)

The event occurs when the command “!stop” has been received from the players or the players have been silent for some predefined period of time at the channel.

When the event occurs, bot stops the game and switches to sleep mode.

3.3. Event “Started receiving the definitions” (e3)

The event occurs when term is successfully displayed at the channel.

When the event occurs, bot starts receiving the definitions. The corresponding message is displayed. Bot starts tracking the time for players to think of the term definitions.

3.4. Event “Finished receiving the definitions”(e4)

The event occurs when the time for players to think is over.

When the event occurs, bot starts waiting for the players' choice. The corresponding message is displayed and bot stops tracking the time for players to think about the definitions.

3.5. Event "The players have made their choice" (e5)

The event occurs when the time for definition receiving is over.

When the event occurs, bot calculates new scores. The corresponding message is displayed.

3.6. Event "The player has sent a message" (e6)

The event occurs when some message from one of the players has been received. The message should differ from the control command.

This event can be processed in different ways depending on the type of message and on the state of the automaton.

4. Definition of the Output Actions

The following output actions are used in the program:

z1	–	Finish receiving the definitions
z2	–	Switch to sleep mode
z3	–	Score calculation and display the results
z4	–	Announce the start of the game
z5	–	Announce the term and start receiving the definitions
z6	–	Store version of the player's definition
z7	–	Store the definition, that player has chosen

4.1. Action "Finish receiving the definitions" (z1)

The action adds the right definition to the list of players' definitions and shuffles all of them. It assigns a number for each definition and displays the definitions at the channel.

4.2. Action "Switch to sleep mode" (z2)

The action displays the message telling that the game bot is switching to sleep mode.

4.3. Action “Score calculation and display the results” (z3)

The action creates the result table, displays it, adds scores to the players and stores the scores in the database.

4.4. Action “Announce start of the game” (z4)

It displays message about the start of the game.

4.5. Action “Announce the term and start receiving of definitions” (z5)

The term is chosen randomly, then it is displayed at the channel for players. The bot starts tracking the time for players to think of the definition.

4.6. Action “Store version of the player’s definition” (z6)

If the player has given the definition, it is stored otherwise the warning for the player is displayed.

4.7. Action “Store the definition, that player has chosen”(z7)

The player’s choice is being analyzed. The warning is displayed if:

- the player has already given the definition;
- the player has entered not a number or a number is out of range;
- the player has chosen his definition.

Otherwise the player’s choice is stored.

5. Automaton “Management of the game” (A0)

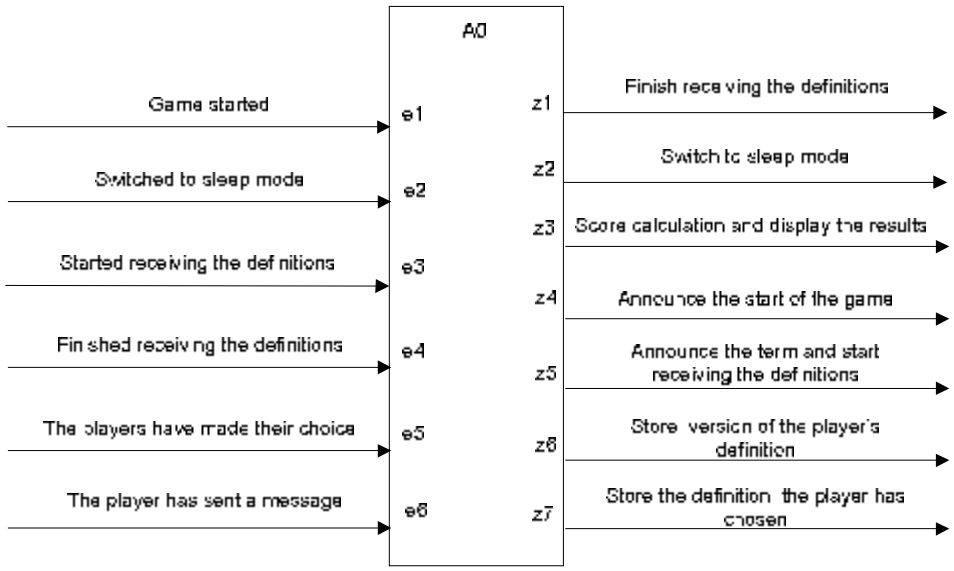
5.1. Verbal description

This automaton controls the game.

Special “waiting” mode was created to avoid the idle work of the bot. Events that make bot switch back and forth to the waiting mode have been added.

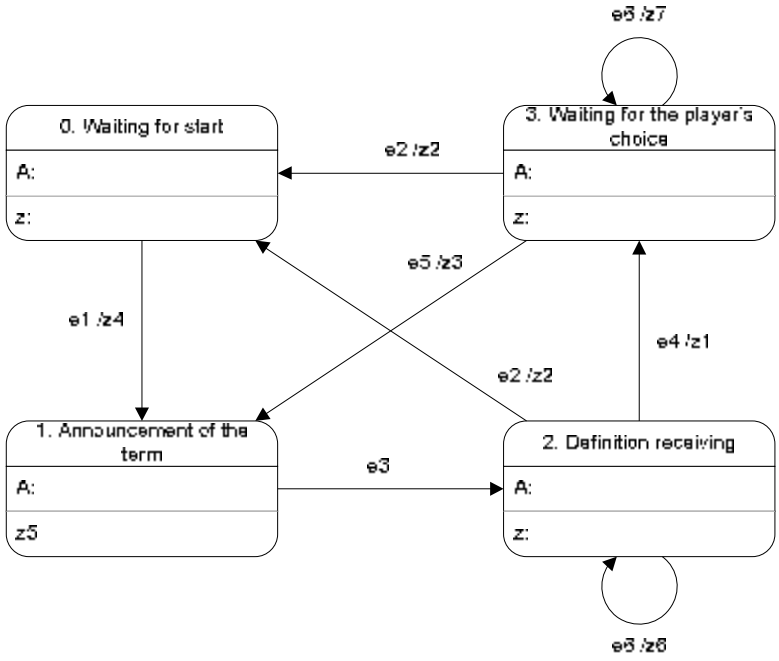
For the diagram of the connections and transition graph of the automaton see the pic. 2, 3.

5.2. The scheme of the automaton connections



Pic.1. Connections diagram of the automaton «Management of the game»

5.3. Transition graph



Pic.2. Transition graph of the automaton «Management of the game»

6. Conclusion

At the first glance it seems, that there is no sense in using automata programming for such a simple problem. However, the experience gained during the development of the previous text game (*Quiz*) has shown that the traditional methods of programming (with Boolean flags, etc) may cause many errors. It's often difficult to find and fix these errors.

The program correctness can be checked right after creation of the code with "stubs". The authors used this opportunity for debugging. The messages about state ids and output actions were logged in the stubs. The code worked properly. The transition graph was used to check all of the transitions and all of the executed output actions.

Nowadays beta-testing is performed to find out the players' wishes. The automata-based programming should simplify the modification of the program.

7. Program listing

```
#!/usr/bin/perl -w

#   Module:                   bot_II.pl
#   Description:              Bot “Zavalinka”
#   The last modification:    2003-01-24

#
#   ==[ Used modules ]==
#
use strict;     # warning activation
use Net::IRC;   # module connection for IRC usage
use nickdb;     # module connection for nick database usage

#
#   ==[ Constants ]==
#
my $BotVersion='v0_1b';         # bot version
my $Channel='#zavalinka';       # channel name where bot will be located
my $ServerName='irc.wom.ru';    # IRC server address
my $ServerPort=6667;            # port which is used by IRC server

#
#   ==[ Colours in IRC ]==
#
my $cwhite =        "\00300";
my $cnormal=        "\00301";
my $cblue=          "\00302";
my $cgreen=         "\00303";
my $cred =          "\00304";
my $cbrown=         "\00305";
my $cpurple =       "\00306";
my $corange =       "\00307";
my $cyellow =       "\00308";
my $clightgreen =   "\00309";
my $ccyan =         "\00310";
my $clightcyan =    "\00311";
my $clightblue =    "\00312";
my $cpink =         "\00313";
my $cgrey =         "\00314";
my $clightgrey =    "\00315";

#
#   ==[ Console colours ]==
#
my $anormal =       "\033[0m";
my $agrey =         "\033[1m";
my $ainvert =       "\033[7m";
my $ared =          "\033[0;31m";
my $agreen =        "\033[0;32m";
my $ayellow =       "\033[0;33m";
my $ablue =         "\033[1;34m";
my $apink =         "\033[1;35m";
my $alightblue =    "\033[1;36m";
my $awhite =        "\033[1;37m";

my $timetogetask = 60;      # waiting period (in seconds) for defenition receiving
my $timetogetanswer = 25;   # waiting period for players’ choice
# salutation message from bot after connection to the server
my $HelloMessage = "$cred Hello everybody! It’s me again!";
# farewell message from bot before disconnection from server
my $ByeMessage = "$cred Nice talking to you. But it’s time for leaving. Bye, everybody";
my $State = 0;              # initial state of automaton “Zavalinka”.

my $basefile = "../bot/logs/macbase.txt";   # file with term base
my $logfile = ">>zavalina.log";             # file with channel logs

my $termin = '';            #
my $opred = '';             #
my $countv = 0;             #
my $gamers = 0;             #

#
```

```
#   ==[ Initialization ]==
#
my $BotNick=NextNick('');   # setting bot's name at the channel
my $BotRealNick=$BotNick;   #
my %defin=();
my %idnick=();
my %gamer_choice=();
my %defin_owner=();
my %scores=();
my $I='undef';
$SIG{INT}=\&quit;           # action at program exit

#
#   ==[ The automaton "Management of the game" ]==
#
#    The automaton "Management of the game"
#
#   Invocation:      OnEvent ( Event, [ Param1 [, Param2] ] )
#   Parameters:  Event   -   event
#                Param1  -   optional parameter, it depends on the event
#                Param2  -   optional parameter, it depends on the event
sub OnEvent {
    my $Event = shift;          # reading of the event

    # State of the automaton s0 "Waiting for start"
    if($State == 0){
        if ($Event eq 'e1'){    # if the event e1 "Game started" has happened
            z4();               # execute action z4 "Announce the start of the game"
            $State = 1;         # switch to mode s1 "Announcement of the term"
        }

    # State of the automaton s1 "Announcement of the term"
    }elsif($State == 1){
        z5();                   # execute action z5 "Announce the term and start
receiving the definitions"
        if ($Event eq 'e3'){    # if the event e3 "Started receiving the definitions"
            $State = 2;         # switch to mode s2 "Definition receiving"
        }

    # State of the automaton s2 "Definition receiving"
    }elsif($State == 2){
        if ($Event eq 'e2'){    # if the event e2 "Switched to sleep mode" has occured
            z2();               # execute the action z2 "Switch to sleep mode"
            $State = 0;         # switch to mode s0 "Waiting for start"

        }elsif($Event eq 'e4'){ # if the event e4 "Finished receiving the definitions"
has occured
            z1();               # execute action z1 "Finish receiving the definitions"
            $State = 3;         # switch to mode s3 "Waiting for the player's choice"

        }elsif($Event eq 'e6'){ # if the event e6 "The player has sent a message" has
happened
            z6(shift, shift);   # execute action z6 "Store the definition, the player
has chosen"
        }

    # State of the automaton s3 "Waiting for the player's choice"
    }elsif($State == 3){
        if ($Event eq 'e2'){    # if the event e2 "Switched to sleep mode" has occured
            z2();               # execute action z2 "Switch to sleep mode"
            $State = 0;         # switch to mode s0 "Waiting for start"

        }elsif($Event eq 'e5'){ # if the event e5 "The players have made their choice"
has occured
            z3();               # execute action z3 "Score calculation and display the
results"
            $State = 1;         # switch to mode s1 "Announcement of the term"

        }elsif($Event eq 'e6'){ # if the event e6 "The player has sent a message" has
occured
            z7(shift, shift);   # execute action z7 "Store the the definition, the
player has chosen"
        }
    }
}

# Action z1 "Finish receiving the definitions"
#
```

```
# The action adds the right definition to the list of players' definitions and shuffles
# all of them. It assigns a number for each definition and displays the definitions
# at the channel
sub z1{
    my @s = ();
    %defin_owner = ();
    my $var = 0;
    my $var2 = 0;
    for (my $i = 0; $i < ($gamers+1); ++$i){
        @s=(@s,$i);
    }
    $I->privmsg($Channel, 'The time for definition receiving is over.');
    for (my $i = 0; $i < ($gamers+1); ++$i){
        $var = int(rand($gamers+1-$i));
        $var2 = $s[$var];
        $s[$var] = $s[$gamers-$i];
        $I->privmsg($Channel, "$cred".($i+1)."$clightblue)
\"".$defin{$idnick{$var2}}."\"");
        $defin_owner{($i+1)} = $idnick{$var2};
    }
    $I->privmsg($Channel, $cblue."Please, choose the number of the right definition:");
    $I->privmsg($Channel, $cblue."You have ".$cgreen.$timetogetanswer.$cblue.
        " seconds!".$cgrey." In private message send to $BotRealNick number from ".
        $cgreen."1".$cgrey." to ".$cgrey.($gamers+1));
    $I->schedule($timetogetanswer, \&Get_Choice_Timeout);
    %gamer_choice=();
}

# Action z2 "Switch to sleep mode"
#
# The action displays the message telling that the game bot is switching to sleep mode
sub z2{
    $I->privmsg($Channel, $cgreen."The game is stopped, for its resumption please type
in $cred!start$cnormal");
}

# Action z3 "Score calculation and display the results"
#
# The action creates the result table, displays it, adds scores to the players
# and stores the scores in the database
sub z3{
    my $gamer_nick;
    my $gamer_score;
    foreach $gamer_nick (sort keys %scores){
        $gamer_score=$scores{$gamer_nick};
        if ($gamer_nick ne $BotRealNick){
            $I->privmsg($Channel, $cbrown.$gamer_nick." Has got".$gamer_score."
Score(s).");
        }
        AddScore($gamer_nick, $gamer_score);
    }
    %scores=();
    FlushBase();
}

# Action z4 "Announce the start of the game"
#
# It displays message about the start of the game
sub z4{
    $I->privmsg($Channel, $cgreen."The game has started(I'm awake!)$cnormal");
}

# Action z5 "Announce the term and start receiving of definitions"
#
# The term is chosen randomly, then it is displayed at the channel for players
# The bot starts tracking the time for players to think of the definition

sub z5{
    Get_random_termin();
    $I->privmsg($Channel, $cblue."Here is a term for you:$cred $termin");
    $I->privmsg($Channel, $cblue."You've got ".$cgreen.$timetogetask.$cblue.
        " seconds to offer its definition.".$cgrey.
        " Send all your versions to bot (".$cgreen."$BotRealNick".
        $cgrey.") in private message");
    $I->schedule($timetogetask, \&Get_Definitions_Timeout);

    %defin=();
    $defin{$BotRealNick}=$opred;
    %idnick=();
```

```
    $idnick{0}=$BotRealNick;
    $gamers=0;
}

# Action z6 "Store version of the player’s definition"
#
# If the player has given the definition, the given definition is stored otherwise
# the warning for the player is displayed
sub z6{
    my $msg=shift;
    my $nick=shift;

    if($defin{$nick} eq ''){
        $I->privmsg($nick, $cblue."For term $cred\"$termin\"$cblue the definition is
accepted $clightblue\"$msg\"");
        $defin{$nick}=$msg;
        $gamers=$gamers+1;
        $idnick{"$gamers"}=$nick;
    }else{
        $I->privmsg($nick, $cred."You’ve already given the definition.");
    }
}

# Action z7 "Store the definition, the player has chosen"
#
# The player’s choice is being analyzed. The warning is displayed if:
#        the player has already given the definition;
#        the player has entered not a number or a number out of range;
#        the player has chosen his definition.
# Otherwise the player’s choice is stored
sub z7{
    my $msg=shift;
    my $nick=shift;

    my $num=$msg-0;
    my $d_owner;

    if($gamer_choice{$nick} eq undef){
        if (($num>0) && ($num<=($gamers+1))){
            $d_owner=$defin_owner{$num};
            if ($d_owner ne $nick){
                $gamer_choice{$nick}=$num;
                if ($defin_owner{$num} eq $BotRealNick) {
                    $scores{$nick}=$scores{$nick}+1;
                }
                $scores{$d_owner}=$scores{$d_owner}+2;
                $I->privmsg($nick, $cgreen."Your choice\"".$num."\" has been taken in
account.");
            }else{
                $I->privmsg($nick, $cred."It’s not allowed to choose your own definition
!");
            }
        }else{
            $I->privmsg($nick, $cred."You need to enter the number from
".$cgreen."1".$cred." to $cgreen".($gamers+1)."");
        }
    }else{
        $I->privmsg($nick, $cred."You’ve already chosen the definition:
".$cgreen.$defin{$d_owner});
    }
}

#
#   ==[ Auxiliary procedures ]==
#

#
# Timer event creation
#
sub Get_Definitions_Timeout{
    OnEvent('e4');
}

sub Get_Choice_Timeout{
    OnEvent('e5');
    OnEvent('e3');
}

# It determines the number of terms in the base
```

```
sub calc{
    $countv = 0;
    my $answ2;
    open(INP, $basefile);
    while(my $s = <INP>){
        $s =~ s/\n//;
        $s =~ s/\r//;
        if (($s =~ /\@(.*)/)){
            if(length($1) < 15){
                $answ2 = $1;
            }else{
                $answ2 = "";
            }
        }else{
            if ((length($answ2) != 0) && (length($s) < 30)){
                $countv++;
            }
        }
    }
    close(INP);
}

# It returns the term which was chosen at random from the database
sub Get_random_termin{
    my $vopr = int(rand($countv));
    my $answ;
    print "The term: $vopr from $countv the terms\n";
    open(INP, $basefile);
    while(my $s = <INP>){
        $s =~ s/\n//;
        $s =~ s/\r//;
        if (($s =~ /\@(.*)/)){
            if(length($1) < 15){
                $answ = $1;
            }else{
                $answ = "";
            }
        }else{
            $vopr--;
            if ((length($answ) != 0) && (length($s) < 30)){
                if ($vopr < 0){
                    $opred = $s;
                    $termin = $answ;
                    last;
                }
            }
        }
    }
}

# Nick setting
sub NextNick{
    my $a=$_[0];
    my $r = 'undef';
    my @l = ( 'Kidalla', 'Pechka', 'Bot', 'Zavalinka' );
    for (my $i = 0; $i <= $#l; $i++){
        if ($l[$i] eq $a) {
            $r = $l[$i + 1];
        }
    }
    if ($r eq 'undef') {
        $r = $l[0];
    }
    if ($a eq '') {
        $r = 'K'.time;
    }
    return $r;
}

#
# ==[ IRC handlers ]==
#

# Successful connection with server
sub on_connect {
    my $self = shift;

    print "$ayellow Joining $ServerName.($alightblue $Channel $ayellow)...$anormal";
    $self->join($Channel);
```

```
}

# Server messages after connection
sub on_init {
    my ($self, $event) = @_;
    my (@args) = ($event->args);
    shift (@args);

    print "$ablue*** @args $anormal\n";
}

# Somebody has left the channel
sub on_part {
    my ($self, $event) = @_;
    my ($channel) = ($event->to)[0];

    printf "$alightblue*** $awhite%s$alightblue has left channel %s$anormal\n", $event-
>nick, $channel;
}

# Somebody has joined channel including us
sub on_join {
    my ($self, $event) = @_;
    my ($channel) = ($event->to)[0];

    if($event->nick eq $BotNick){
        $I = $self;
        print "$agreen DONE$anormal\n";
        $BotRealNick = NextNick($BotNick);
        $self->nick($BotRealNick);
        $BotNick = $BotRealNick;

        $self->privmsg('NickServ',"identify <here_password>");
        $self->privmsg('ChanServ',"identify $Channel <here_password>");
        $self->privmsg('ChanServ',"op $Channel $BotRealNick");
        $self->privmsg('ChanServ',"protect $Channel $BotRealNick");

        calc();
        $self->privmsg($Channel, $HelloMessage);
        OnEvent('e1');
        OnEvent('e3');
    }else{
        printf "$alightblue*** $awhite%s$alightblue (%s) has joined channel
%s$anormal\n", $event->nick, $event->userhost, $channel;

        open(MSG, $logfile);
        print MSG "[".localtime(time)."]".$event->nick." Joined $Channel\n";
        close(MSG);

        if (GetScore($event->nick)<0){
            $I->me($event->nick, "A very interesting game is offered at the channel!");
            $I->me($event->nick, "Game rules. Quizmaster (bot $BotRealNick) ".
                "offers a term to the players (term is a strange word). During set"
                " period of time the players should think of the term definitions ".
                ". Then the players tell their versions of definitions secretly to the
                " quizmaster in private messages. After all bot tells all versions ".
                "with the original one. The players shoose the most suitable ".

            $I->me($event->nick, "Score calculation. Every player who guessed ".
                " the right definition gets one point. Besides,the players ".
                "get points if their definitions were chosen by others ".
                "2 points for the mistaken player ".
                "The quizmaster calculates scores and announces the results");
            $I->me($event->nick, "The following commands are accepted at the channel
"!start !stop !won !help !top.");
        }
    }
}

# The private message has been received PRIVMSG
sub on_msg {
    my ($self, $event) = @_;
    my ($nick) = $event->nick;
    my ($arg) = ($event->args);

    OnEvent('e6', $arg, $nick);

    print "$agreen*$nick*$awhite  ", ($event->args), "$anormal\n";
```

```
    open(MSG,$logfile);
    print MSG "[".localtime(time)."]".$nick."***> $arg \n";
    close(MSG);
}

# Message at the channel
sub on_public {
    my ($self, $event) = @_;
    my @to = $event->to;
    my ($nick, $mynick) = ($event->nick, $self->nick);
    my ($arg) = ($event->args);

    open(MSG,$logfile);
    print MSG "[".localtime(time)."]".$nick."> $arg\n";
    close(MSG);

    print "$agreen<$nick>$anormal $arg\n";

    if
(($arg=~/^\s*?\![cCсС][тТТ][аАaA][рРpP][тТТ]\s*?$/)||($arg=~/^\s*?\![sS][тТТt][аАaA][rR]
[тТТt]\s*?$/)){
        OnEvent('e1');
        OnEvent('e3');

}elsif(($arg=~/^\s*?\![cCсС][тТТ][oOоО][пП]\s*?$/)||($arg=~/^\s*?\![sS][тТТt][оОoO][pPрР
]\s*?$/)){
        OnEvent('e2');

}elsif(($arg=~/^\s*?\![вВB][оОoO][нНH]\s*?$/)||($arg=~/^\s*?\![wW][оОoO][nN]\s*?$/)){
        my $sc=GetScore($nick);
        if ($sc>=0){
            $I->privmsg($Channel,"$cred You $nick have got ".$sc." scores.");
        }else{
            $I->privmsg($Channel,"$cred $nick, have got any scores.");
        }

}elsif(($arg=~/^\s*?\![hHН][еЕeE][lL][pPрР]\s*?$/)||($arg=~/^\s*?\![хХxX][еЕeE][лЛ][пПn]
\s*?$/)){
        $I->privmsg($nick, "Game rules. Quizmaster (bot $BotRealNick) ".
                "offers a term to the players (term is an unintelligible word). During
set"
                " period of time the players should think of the term definitions ".
                ". Then the players tell their versions of definitions secretly to the
                " quizmaster in private messages. After all bot tells all versions ".
                "with the original one. The players choose the most suitable.");

        $I->privmsg($nick, "Score calculation. Every player who guessed ".
                " the right definition gets one point. Besides,the players ".
                "get points if their definitions were chosen by others ".
                "2 points for the every player ".

        "The quizmaster calculates scores and announces the results");
        $I->privmsg($nick, "The following commands are accepted at the channel !start
!stop !won !help !top.");

}elsif(($arg=~/^\s*?\![тТТ][oOоО][пП]\s*?$/)||($arg=~/^\s*?\![тТТt][оОoO][pPрР]\s*?$/)){
        my @top=GetNickList();
        my $message='';
        for (my $i=1; $i < scalar @top; ++$i){
            if ($i <= 5){$message=$message.$cgreen.($i).") место:$cpurple
".$top[$i]."$cgreen c ".GetScore($top[$i])." очкаами. ";}
        }
        $I->privmsg($Channel, $message);
    }
}



# Message display from the nick
sub on_umode {
    my ($self, $event) = @_;
    my @to = $event->to;
    my ($nick, $mynick) = ($event->nick, $self->nick);
    my ($arg) = ($event->args);
    print "$ared<$nick> $arg $anormal\n";
}

# DCC CHAT attempt.
sub on_chat {
```

```
    my ($self, $event) = @_;
    my ($sock) = ($event->to)[0];

    print '*' . $event->nick . '* ' . join(' ', $event->args), "\n";
    $self->privmsg($sock, 'I Don\'t want to DCC chat just now.');
}

# Get the list of people at the channel
sub on_names {
    my ($self, $event) = @_;
    my (@list, $channel) = ($event->args);

    ($channel, @list) = splice @list, 2;

    print "$ablue*** Users on $channel: @list $anormal\n";
}

# there was inquiry for DCC SEND or CHAT
sub on_dcc {
    my ($self, $event) = @_;
    my $type = ($event->args)[1];
    if (uc($type) eq 'CHAT') {
        $self->new_chat($event);
    } else {
        print STDERR ("$ared Unknown DCC type: " . $type . "$anormal\n");
    }
}

# CTCP PINGs registration
sub on_ping {
    my ($self, $event) = @_;
    my $nick = $event->nick;

    $self->ctcp_reply($nick, join (' ', ($event->args)));
    print "$ablue*** CTCP PING request from $nick received$anormal\n";
}

# PINGs processing
sub on_ping_reply {
    my ($self, $event) = @_;
    my ($args) = ($event->args)[1];
    my ($nick) = $event->nick;

    $args = time - $args;
    print "$ablue*** CTCP PING reply from $cwhite $nick $cblue: $args sec.$anormal\n";
}

# if nick has been lost, we try to define next nick in the list.
sub on_nick_taken {
    my ($self) = shift;
    $BotRealNick=NextNick($BotNick);
    $BotNick=$BotRealNick;
    $self->nick($BotRealNick);
}

# CTCP ACTIONs representation
sub on_action {
    my ($self, $event) = @_;
    my ($nick, @args) = ($event->nick, $event->args);

    print "$apink* $nick @args $anormal\n";
}

# "notice" representation
sub on_notice {
    my ($self, $event) = @_;
    my ($nick, @args) = ($event->nick, $event->args);

    print "-$nick- @args\n";
}

# in case of disconnection with server we try to reconnect
sub on_disconnect {
    my ($self, $event) = @_;

    print "$ared Disconnected from ", $event->from(), " (",
          ($event->args())[0], "). Attempting to reconnect...$anormal\n";
    $self->connect();
}
```

```
# topic of the channel is defined
sub on_topic {
    my ($self, $event) = @_;
    my @args = $event->args();

    if ($event->type() eq 'notopic') {
        print "No topic set for $args[1].\n";
    } elsif ($event->type() eq 'topic' and $event->to()) {
        print "Topic change for ", $event->to(), ": $args[0]\n";
    } else {
        print "The topic for $args[1] is \"$args[2]\".\n";
    }
}

# the scripts are called at the end of the work
sub quit{
    print "$ared Exiting...$anormal\n";
    if($I ne 'undef'){
        $I->privmsg($Channel, $ByeMessage);
        $I->quit("$cgreen has gone$cnormal");
    }
    exit 0;
}

#
# ==[ Start of the program ]==
#

# IRC launching
my $irc = new Net::IRC;

# connection to the server
print "$ared Creating connection to IRC server $ServerName port
$ServerPort...$anormal\n";

my $conn = $irc->newconn(Server   => ($ARGV[0]  ||  $ServerName),
             Port     => $ServerPort,
             Nick     => $BotNick,
             Ircname  => 'Bot_by_Petka.',
             Username => "Bot-$BotVersion")
    or die "$awhite irctest: Can't connect to IRC server($ServerName).$anormal\n";

# setting of IRC handlers
print "$ared Installing handler routines...$anormal";
$conn->add_handler('cping',   \&on_ping);
$conn->add_handler('crping',  \&on_ping_reply);
$conn->add_handler('msg',     \&on_msg);
$conn->add_handler('chat',    \&on_chat);
$conn->add_handler('public',  \&on_public);
$conn->add_handler('caction', \&on_action);
$conn->add_handler('join',    \&on_join);
$conn->add_handler('umode',   \&on_umode);
$conn->add_handler('part',    \&on_part);
$conn->add_handler('cdcc',    \&on_dcc);
$conn->add_handler('topic',   \&on_topic);
$conn->add_handler('notopic', \&on_topic);
$conn->add_handler('notice',  \&on_notice);

$conn->add_global_handler([ 251,252,253,254,302,255 ], \&on_init);
$conn->add_global_handler('disconnect', \&on_disconnect);
$conn->add_global_handler(376, \&on_connect);
$conn->add_global_handler(433, \&on_nick_taken);
$conn->add_global_handler(353, \&on_names);

print "$ared done.$anormal\n";

print "$ared starting...$anormal\n";

#
# work cycle launching
#
$irc->start;
```