ACV
Course Work - Cifras y Letras Game

Vicente Bosch Campos †
viboscam@posgrado.upv.es

May 26, 2011

Contents

List of Figures

1 Introduction

1.1 Context

For the ACV course project we will perform an implementation of the Cifras y Letras game in the Ruby programming language.

Cifras y Letras was originally conceived in as a game show. The game is based on the resolution of two type of logic problems: Numbers and Tests.

The rules for the numbers tests are as follows:

- The six selected numbers are chosen randomly from: 1,2,3,4,5,6,7,8,9,10,25,50,75 & 100 taking into account that the probability to choose one of the numbers between 1 and 10 must be double than choosing the rest of the numbers.
- The target number must be chosen randomly from the range 101..999.
- Operations allowed to combine numbers are: sum, subtraction, multiplication and division. (division can only be applied if it is exact)
- Operations must be performed with the selected numbers or with the numbers obtained by operating on them.
- Results of operations must always be integer and positive.
- If the exact number can not be obtained, the best result will be the one closest to it.
- 45 seconds are given to the players to come up with a valid answer.
- The player with the best answer obtains 10 points.

The rules for the Letters tests are as follows:

- The users select randomly 9 characters from the vowel or consonants groups.
- The target is to obtain the longest valid word from the Spanish language with the given letters:
  - Letters must only be used once.
  - Plurals of words are not valid
  - Personal forms of verbs are not valid.
  - Feminine forms of words are valid.
- The player that provides the longest word constructed with the set of letters given is the winner.
- The winner receives as many points as letters had the winning word it provided.

A cifras y Letras game is composed by a set of Letters and Numbers tests as described above. Number of tests and order depends on the game format. Usually (and in our implementation) the test set is as follows: Numbers - Letters - Letters - Numbers - Letters - Letters - Numbers - Letters - Letters - Numbers.

The winner of the game is the player with the highest score at the end of all tests.

1.2 Scope

In order to implement the game we will develop components in order to cover the following user cases:

- Generate problems of the described Numbers and Letters tests.
- Evaluate answers given by the players to the specific tests.
- AI Components in order to resolve and provide the players with better answers in order to increment their knowledge.

- Game component in order to ensure the mechanics and rules described.
- Client and Server components to allow for multiple users to join a game and play together players.
- Scores and Score list to record players with highest scores in order to provide recognition for their prowess in the game.

2 Development

2.1 Design

2.1.1 Decisions and Known Issues

Following architecture decisions have been made:

- Initially all communications where conceived to be performed with remote objects (by using distributed ruby) but due to the fact the communications using this technology are difficult to handle when there are many observers for the same object and the speed (due to serialization and deserialization of objects) of communications we swapped the architecture in order to use message based communication.
- As many games can be launched in parallel handling all communications discerning the different games and players from a single server was too complicated and also if there are any memory leak problems or errors on the games it would impact on all other games. In order to avoid this a game has been designed as an independent software component that is launched as an entirely different process from the game server. In this manner we have:
  - Simplified communications as there are no routings: players can only connect with games they are connected with and vice versa.
  - We have encapsulated the game and hence our architecture can escalate to serve as many games as the machine can hold.
- As the queue message server is not part of the game server but an independent software component in order to avoid any issues of communication between the game server and the clients in case we have an issue due to abrupt shutdown of the message queue communications from the game server to the client are performed with distributed ruby as those communications are not very frequent and the payload is light. Hence we ensure that the communications with the game server will continue even if disaster strikes the message queue server which adds resilience to the architecture.
- Both AI components are developed to exit and provide a best answer reached within a time constraint. This is performed so that we will always have an answer to provide to the user and hence avoid disruption on the expected functionality.
- The game loop is done with discrete simulation:
  - The different tests communicate with the game by usage of the observable pattern. When there is a modification that must be communicated to the observers it notifies them. For example time in which the test must be performed is measured by launching a separate process with a timer once the timer finishes the observers get notified.
  - The game object and the game client subscribe themselves to a message queue and wait to be called when specific messages arrives to the them regarding changes in the game or solutions to the current test.

2.1.2 Selected libraries

For our implementation of the sequence number problem we have implemented the algorithm in Ruby (v1.9.2) and the following libraries where used:

Trollop (1.16.2): Trollop was used in order to parse and generate validators of the command line options for the commands created.

Stomp (1.1.8): The Stomp gem was used to perform fast bidirectional communication from the game client programs to the the remote game. Stomp client is used to connect the process to a running ActiveMQ server.

Awesome Print (1.1.8): The awesome print gem is used in order to present the user with pretty colorful outputs from the application through the console.

ruby-debug19 (0.11.6): ruby-debug gem was used during the implementation of the software and is required for any feature addition or maintenance of the software components.

rgl (0.4.0): The rgl gem is used inside the operation graph class to generate a .png image representation of the operation described in the graph.

rmagick (2.13.1): rmagick is used in order to display to the user the operation graphs .png generated by the rgl library.

gnuplot (2.3.6): gnuplot gem is used by the Genetic Population class to present the user with demographic information of the genetic individuals during its evolution.

DRB: Distributed ruby standard library module is used in order to publish the stand alone game server process. DRB is used in this case to ensure that the server will keep running even if the ActiveMQ server fails.

Observable: Standard library module was used in order to produce the call back strategy for the game loop. In our implementation the objects will receive alerts when certain states are reached using a publisher - subscriber pattern as performed in discrete simulation hence avoiding active waits.

Timeout: Standard library object was used in order to enforce the time constraint on the execution of the genetic algorithm.

2.2 Software Installation

2.2.1 Ruby Installation

Ruby is a dynamic, open source programming language with a focus on simplicity and productivity. It has an elegant syntax that is natural to read and easy to write.

In order to execute the Cifras y Letras game the ruby virtual machine is required to be downloaded and installed. For this purpose please follow the instructions as described in `http://www.ruby-lang.org/en/downloads/`

Also a more multi-platform Ruby virtual machine based on the java virtual machine can be downloaded from `http://www.jruby.org/`.

2.2.2 Gems Installation

Once the base Ruby virtual machine installation has been concluded the Ruby environment has provided the gem command that will facilitate the download and installation of the libraries described above (internet access is required).

Open a standard OS console and execute the following commands:

- gem install trollop -v 1.16.2
- gem install stomp -v 1.1.8
- gem install awesome_print -v 1.1.8
- gem install ruby-debug19 -v 0.11.6
- gem install rgl -v 0.4.0
- gem install rmagick -v 2.13.1

Once the above commands are executed the specific library versions will be installed to out machine and automatically accessible from all ruby programs that require them.

2.2.3 Apache ActiveMQ Installation

Apache ActiveMQ is a popular open source messaging and integration services provider.

In our software components we will user ActiveMQ 5.5.0 as a stomp protocol server in order to allow bidirectional communication from the remote game to the game client and vice versa.

The base installation package contains the Unix/Linux distribution binary that can be downloaded from `http://activemq.apache.org/activemq-550-release.html`.

The software is downloaded and unzipped in resources folder. The only additional configuration performed was to add the stomp protocol as a transport service. In order to perform perform the following steps:

1. Open the file *./resources/apache-activemq-5.5.0/conf/activemq.xml*
2. Search for the xml tag transportConnectors at the end of the file.
3. Add the following line:

   ```
   <transportConnector name="stomp" uri="stomp://localhost:61613"/>
   ```

   so that the end result is as follows:

   ```
   <transportConnectors>
        <transportConnector name="openwire" uri="tcp://0.0.0.0:61616"/>
        <transportConnector name="stomp" uri="stomp://localhost:61613"/>
   </transportConnectors>
   ```

   .

2.2.4 Numbers Letters Code Download

A Github repository has been set up in order to maintain the code. You can download the latest version by either:

- Go to the webpage https://github.com/vbosch/NumbersLetters and download the code by means of the download button in the web user interface.
- If you have git installed you can download it by executing the command: git clone git@github.com:vbosch/NumbersL

If any work is to be continued on the repository the best way to proceed is to create yourself an account in Github and fork my repository. In this manner you will already have a nice source control set up and you can facilitate access to your new code to other students.

2.3 Program Structure

Next we will detail the main software components and characteristics for the sequence number resolver. We will not describe the software artifacts that pertain to the AI components as they are already described in the pertinent documentation. The implementation can be divided in the following main software components:

Application Configuration: Located in *./lib/application_configuration.rb* is a singleton class that provides global access to general resources (hence acting as a proxy) while ensuring that access is not duplicated. The class implements the singleton pattern by usage of the standard Singleton Module and holds the word dictionary and top score list. It presents the following public methods:

- initialize - Called upon creation of the singleton object loads the word dictionary and top scores list.
- dictionary - Accessor method to the loaded dictionary
- top_scorer - Accessor method to the loaded top scores list

- save_application_status - Saves the current dictionary and top score list to a predetermined file name in the resources folder.

Application Logger: Present in *./lib/application_logger.rb* is a singleton class that extends the standard logger object with some pre configuration. It provides the rest of classes with an easy to use logger that can be configured centrally.

Game: Coded in *./lib/game.rb.* The game class is the main class where the game mechanics are described, it has been coded to be launched in a stand alone process (for which the command remote_game.rb was generated). It presents the following characteristics:

- It communicates with the logic tests it launches by use of the observable pattern. Hence there is no active polling on the test status and timer, instead the game object is notified when the test status is modified to act appropriately.
- Communication of the remote game object with the game clients is performed by messages sent through ActiveMQ server. The game once started it generates a unique queue with its pid as identifier to send out messages to its players.
- Each game client creates a queue to send out messages to the remote game object. The game object subscribes to these queues.
- The nature,number and order of test to perform in the game are described as an Array of test classes that are executed in order. Hence adding new type of tests or adding more tests to the game structure is only a change in the description of the @tests array.
- The game loop is performed by the different events that occur as the tests occur and send out updates to the game or the game clients send messages to the game through the ActiveMQ server. Hence the game loop is completely discrete and there is no active polling on events or messages.

The game presents the following public methods:

- initialize - Creates the game queue, adds the owner user to the games player list and waits for incoming messages.
- game_owner - returns the name of the game owner.
- started? - returns true if the game has started.
- in_preparation? - returns true if the game has not started and players can still be added.
- players_in_game - returns the number of players in the game.
- add_player - adds a player to the list of game players and makes the game object subscribe to the game client message queue.
- player_list - returns the list of player names.
- my_id - returns the game id for a specific player name.
- start_game - starts the game and notifies the players.
- next_test - starts the next test and sends the problem specification to the players.
- set_user_solution - set the user solution for the current test.
- update - callback method used by the test objects to notify of test completion and winner.
- pause_game - can be used a game player to pause the game before the next test.
- resume_game - can be used a game player to resume the game if it is paused.
- kill_player - can be used by the game owner to kick out a player.
- end_game - private method called by the game once the final test is performed and notifies the players.
- winner - returns the current winner of the game.
- classification - returns the current score classification of the game.

Game Client: present in *./lib/game_client.rb.* The GameClient class is the wrapper object to allow communication with the remote Game Server (through the DRB library) and Game (through the ActiveMQ message queue). The GameClient has the callbacks or hooks methods that are called upon status modifications of the Game. The game client presents the following public methods:

- initialize - Connects the game client to the remote game server object through DRB.
- connected_to_server? - returns true if the game client is connected to a game server object.
- retrieve_games_list - returns the current list of games launched from the game server.
- in_game? - returns true if the player is in a game.
- leave_game - allows the player to leave a game.
- owner_of_game? - returns true if the player is in a game and is owner of the game.
- game_started? - returns true if the player is in a game and the game has started.
- list_users_of_game - lists the players of the current game.
- create_game - creates a new remote game object through the game server object through the DRB protocol and subscribes to its message queue.
- update - is the main callback method used by the remote game object to communicate status changes to the game client.
- game_joined - is the callback method used to communicate the game client that it has successfully joined a game.
- start_game - send the start game message to the remote server if invoked by the game owner.
- join_game - adds a user to an existing game and subscribes to its message queue.

Game Server: Coded in *./lib/game_server.rb.* The GameServer class it the main server component of the game it is an object published by means of the DRB gem. The GameServer creates the remote games as stand alone processes and manages them.The game server presents the following public methods:

- initialize - start the game server object and publishes it through the DRB gem and also launches the ActiveMQ server so that the remote game objects and game clients communicate.
- queue_game_port - returns the port where the ActiveMQ server is listening.
- update_top_scores - updates the top scores list and saves it to a file.
- get_top_scores - returns the current top scores list.
- create_game - launches a separate process through the *./bin/remote_game.rb* command to create a remote game object.
- list_games - returns the current list of games.

Message: Present in *./lib/messaging.rb.* The message class acts as vessel to call the methods of listening objects through the message queue.

Message Manager Coded in in *./lib/messaging.rb.* The MessageManager class acts as wrapper class to perform communications through the ActiveMQ server. The message manager has the following public methods:

- initialize - connects the object to the ActiveMQ server.
- subscribe - subscribes the object to a specific message queue.
- unsubscribe - unsubscribes the object from a specific message queue.
- freeze_subscriptions - joins the message queue listening thread so that the object will only be called once a message is received through one of the queues the object has subscribed to.
- publish -sends out a message object through the specific queue assigned to the object managed by the message manager.
- end_messaging - unsubscribes the object from all queues and stops listening for messages.

Letters Problem Generator: Coded in *./lib/letter_problem_generator.rb.* The class creates either manually by user selection or through a random selection a letter problem which consists of 9 letters from the spanish alphabet. The object presents the following public methods:

- initialize - generates the letter problem
- complete? - returns true if 9 letters have been selected.
- add_consonant - adds a letter from the consonant group to the selected set.
- add_vowel - adds a letter from the vowel group to the selected set.
- random_selection - selects 9 letters at random.

Number Problem Generator: Located in *./lib/number_problem_generator.rb.* The class creates a Number problem as described by the numbers test of the game by either random selection or by loading it from a file.

Solution: Coded in *./lib/solution.rb.* Is the base class for the solution of the different test types. It has the following public methods:

- initialize - generates the initial object.
- is_valid? - returns true if the solution contained is valid depending on the specific test.
- exact? - returns true if the solution is exact.
- distance - returns the distance from the target solution, hence when the distance is 0 the solution is exact.

Numbers Solution: Coded in *./lib/numbers_solution.rb* is a solution class child specific for the numbers test. It validates the solution by generating the operation graph associated to the operation.

Letters Solution: Coded in *./lib/letters_solution.rb* is a solution class child specific for the letters test. It validates the solution against the dictionary class.

Test: Present in *./lib/test.rb.* It is the base class for all of the test types. The test classes extend in order to provide the NumbersTest and LettersTest classes.

Score: Present in *./lib/score.rb* keeps the score of a player in a game. It has the following public methods:

- initialize - creates a new score object setting the points to 0
- add_points - adds an specified amount to the current score

Top Scores List: Coded in *./lib/top_scores_list.rb.* Maintains a top score list for a game server. Presenting the following public methods:

- initialize - generates the top score list object.
- has_scores? - returns true if the score list contains scores.
- best_score - returns the best score of the list.
- worst_score - returns the worst score of the list.
- add_scores - adds a set of scores to the top list score.
- add_score - adds a score to the top list score.
- save - saves the top list score to a file.
- self.load - loads a top score list object from a file.

2.4 Numbers Letters Demo Game Platform

The numbers letters client currently holds a basic client implementation that creates a game and starts it with one player.

2.4.1 Numbers Letters Server

A command that launches a stand alone game server.

```
macbosch:bin vbosch$ ./numbers_letters_server.rb -h
numbers_letters_server creates a game server that creates and mananges Numbers and Letters games.

Usage:
       numbers_letters_server [options]
where [options] are:
  --game-server-port, -g <i>:   Port where the remote game server must listen for incoming connections  (default: 9001)
   --game-queue-port, -a <i>:   Port where the queue server will be listening (default: 61613)
               --version, -v:   Print version and exit
                  --help, -h:   Show this message
```

2.4.2 Remote Game Process

The following command launches a stand alone remote game object. This is done by the game server and hence this will not be done neither by a game player or the game server admin.

```
MacBosch:bin vbosch$ ./remote_game.rb -h
remote_game creates a stand alone process for a Numbers and Letters game so that it can be consumed remotely.

Usage:
       remote_game [options]
where [options] are:
        --port, -p <i>:   Port where the remote game server must listen for incoming connections  (default: 9002)
  --user-name, -u <s+>:   User name of the game owner
   --wait-time, -w <i>:   Wait time for a user that pauses the game (default: 30)
         --version, -v:   Print version and exit
            --help, -h:   Show this message
```

2.4.3 Creating a game

In order to create a game the following command has been developed:

```
macbosch:bin vbosch$ ./create_game.rb -h
Create game lets you create Cifras y Letras game.

Usage:
       create_game [options]
where [options] are:
  --game-server-host, -g <s>:   Host where the game server is held (default: localhost)
  --game-server-port, -a <i>:   Port where the remote game server is listening for incoming connections  (default: 9001)
       --player-name, -p <s>:   Name of the player (default: Creator)
           --game-id, -m <i>:   Id of the game we want to join
         --wait-time, -w <i>:   Time to wait for player that has paused the game (default: 30)
               --version, -v:   Print version and exit
                  --help, -h:   Show this message
```

2.4.4 Joining a game

In order to join a game the following command has been developed:

```
macbosch:bin vbosch$ ./join_game.rb -h
Join game lets you join a created Cifras y Letras game.

Usage:
       join_game [options]
where [options] are:
  --game-server-host, -g <s>:   Host where the game server is held (default: localhost)
  --game-server-port, -a <i>:   Port where the remote game server is listening for incoming connections  (default: 9001)
       --player-name, -p <s>:   Name of the player (default: Anonymous)
           --game-id, -m <i>:   Id of the game we want to join
               --version, -v:   Print version and exit
                  --help, -h:   Show this message
```

2.4.5 Running the base example

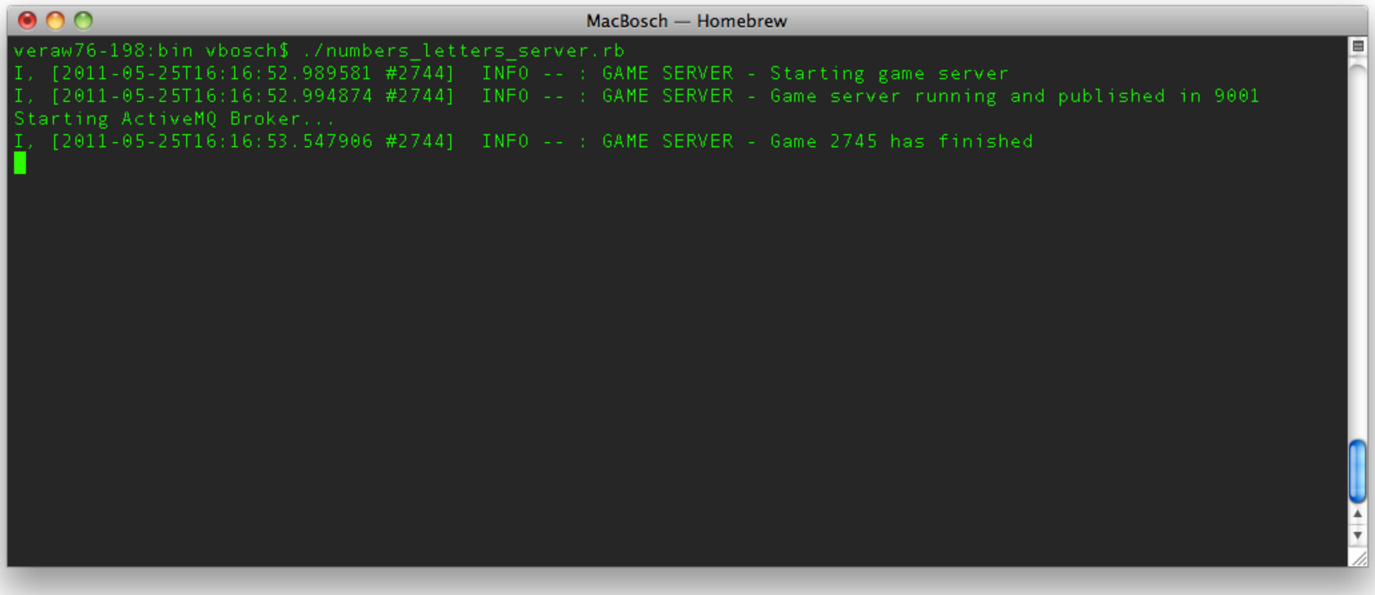
Next we will indicate the steps in order to launch the base game software on your local machine. The developed commands are preconfigured so that they will run with out problems when they are executed on the same machine.

Note that:

- If the server is executed in a different machine than the client (as it would happen in a real life scenario) the game-server-host (-g) option must be indicated when executing the create_game and join_game commands.
- If we choose to change the base port where the game server listens (game-server-port parameter) we will also have to specify the change when launching the create_game and join_game commands.
- Apache Active MQ default port for the stomp protocol is 61613. We have configured this in the Active MQ file in the installation section hence if we want to change the queue port used by the game numbers_letters_server we must first modify that configuration file.

In order to run the base demo of the Cifras y Letras game on the same machine you must perform the following steps:

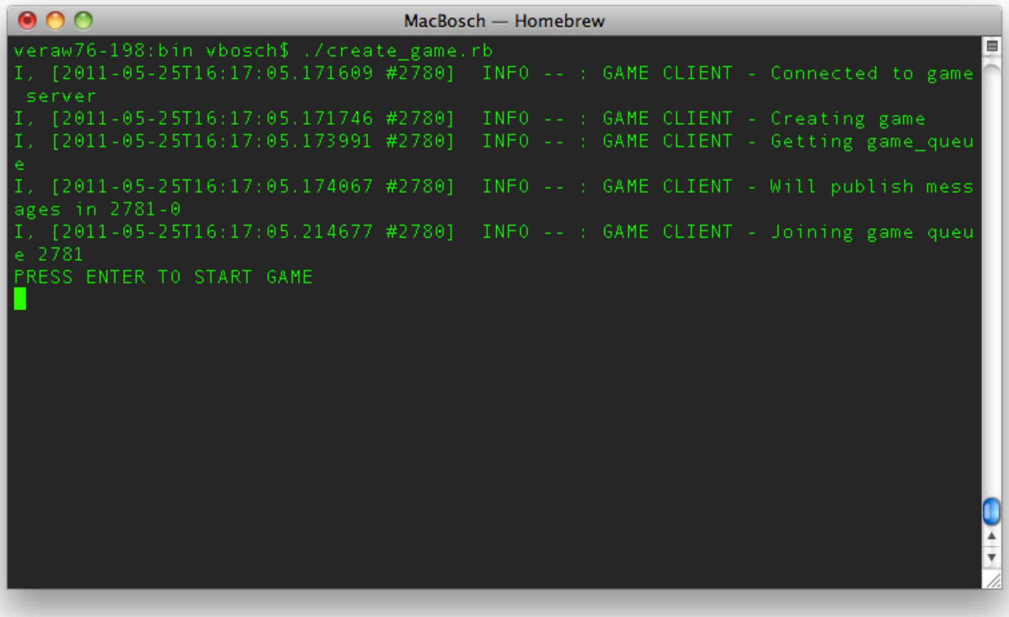
1. Launch the game server. In order to do this open a command window (server window) and type ./numbers_letters_server.rb you will get the following result:



```
veraw76-198:bin vbosch$ ./numbers_letters_server.rb
I, [2011-05-25T16:16:52.989581 #2744]  INFO -- : GAME SERVER - Starting game server
I, [2011-05-25T16:16:52.994874 #2744]  INFO -- : GAME SERVER - Game server running and published in 9001
Starting ActiveMQ Broker...
I, [2011-05-25T16:16:53.547906 #2744]  INFO -- : GAME SERVER - Game 2745 has finished
```

Figure 1: Picture shows the initial output when executing the server.

2. Create a game. Next we open a second command window (creator window) and execute ./create_game.rb. This makes a connection by a user named Creator (default name as none is specified in the command window) to the server and creates a game. We can see the following output on the creator window:

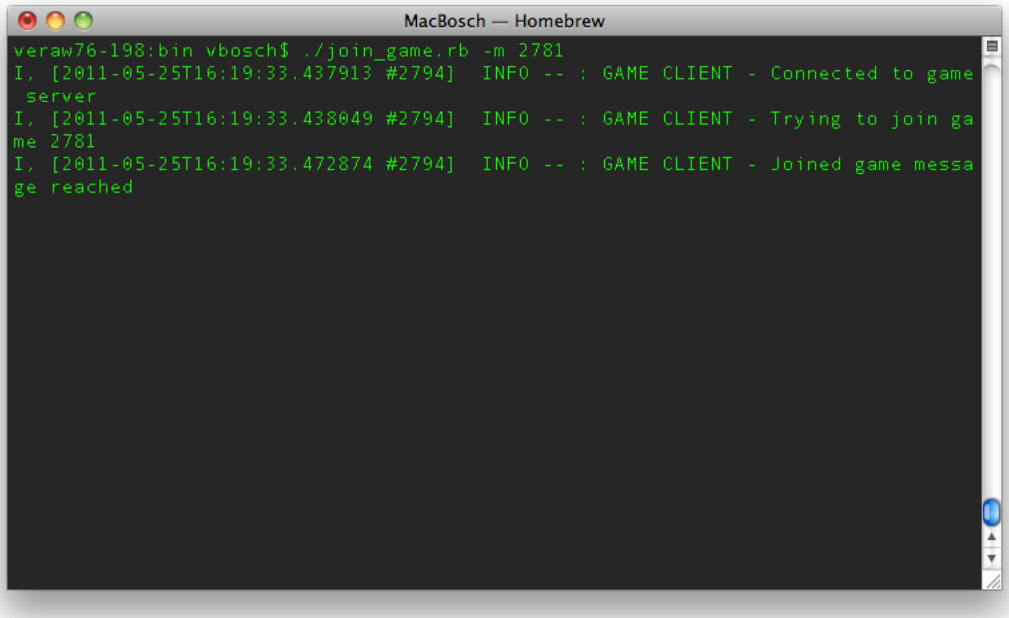


```
veraw76-198:bin vbosch$ ./create_game.rb
I, [2011-05-25T16:17:05.171609 #2780]  INFO -- : GAME CLIENT - Connected to game
 server
I, [2011-05-25T16:17:05.171746 #2780]  INFO -- : GAME CLIENT - Creating game
I, [2011-05-25T16:17:05.173991 #2780]  INFO -- : GAME CLIENT - Getting game_queu
e
I, [2011-05-25T16:17:05.174067 #2780]  INFO -- : GAME CLIENT - Will publish mess
ages in 2781-0
I, [2011-05-25T16:17:05.214677 #2780]  INFO -- : GAME CLIENT - Joining game queu
e 2781
PRESS ENTER TO START GAME
```

Figure 2: Picture shows the initial output when executing game creation command.

As we can see in the example the game has been created listening on the queue name 2781.

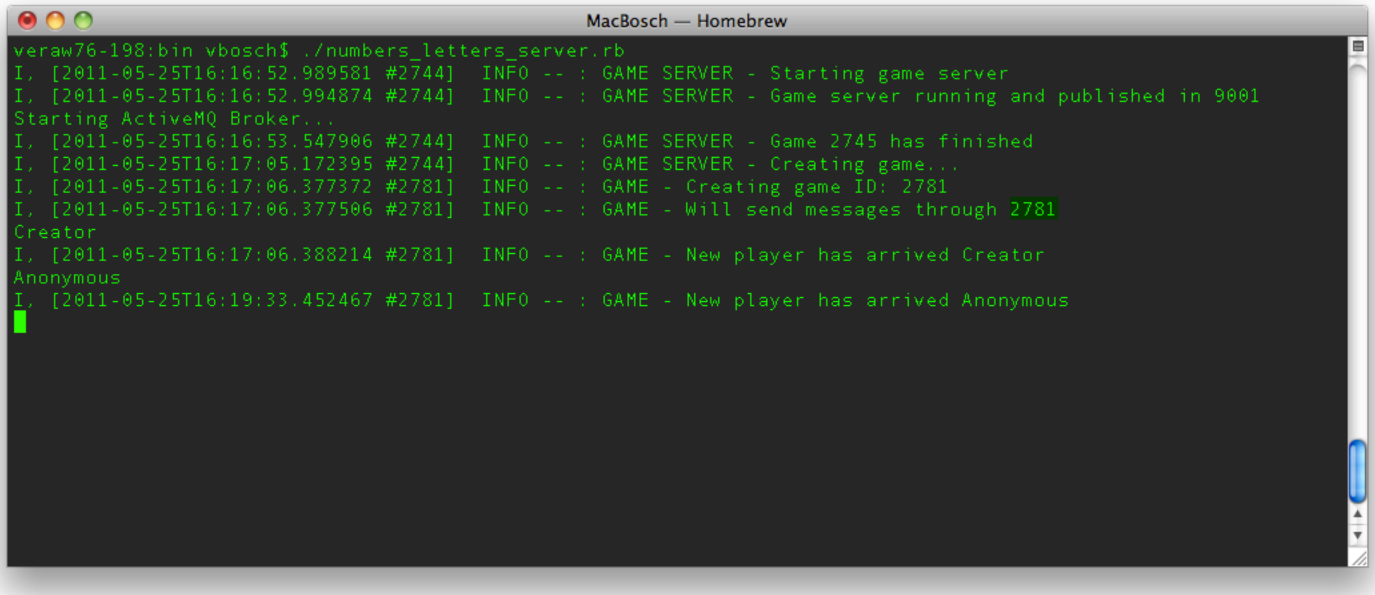
3. Join the game. Next we join the game by opening a new command window (joiner window) and executing ./join_game.rb -m 2781 and get the following result:



```
veraw76-198:bin vbosch$ ./join_game.rb -m 2781
I, [2011-05-25T16:19:33.437913 #2794]  INFO -- : GAME CLIENT - Connected to game
 server
I, [2011-05-25T16:19:33.438049 #2794]  INFO -- : GAME CLIENT - Trying to join ga
me 2781
I, [2011-05-25T16:19:33.472874 #2794]  INFO -- : GAME CLIENT - Joined game messa
ge reached
```

Figure 3: Picture shows the initial output when executing game join command.

If we look at this point the server window we can see the following:

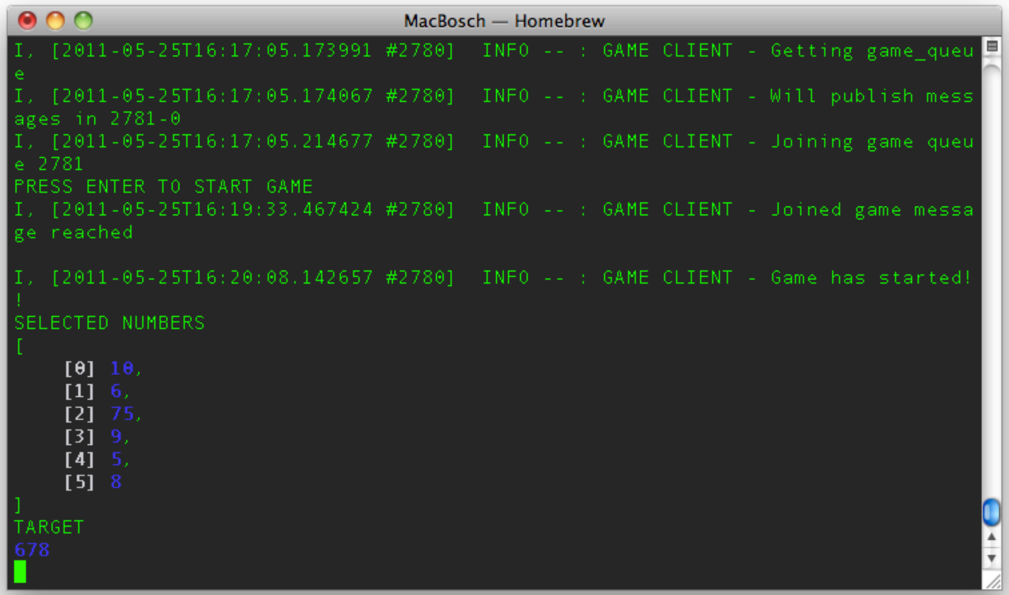


```
veraw76-198:bin vbosch$ ./numbers_letters_server.rb
I, [2011-05-25T16:16:52.989581 #2744]  INFO -- : GAME SERVER - Starting game server
I, [2011-05-25T16:16:52.994874 #2744]  INFO -- : GAME SERVER - Game server running and published in 9001
Starting ActiveMQ Broker...
I, [2011-05-25T16:16:53.547906 #2744]  INFO -- : GAME SERVER - Game 2745 has finished
I, [2011-05-25T16:17:05.172395 #2744]  INFO -- : GAME SERVER - Creating game...
I, [2011-05-25T16:17:06.377372 #2781]  INFO -- : GAME - Creating game ID: 2781
I, [2011-05-25T16:17:06.377506 #2781]  INFO -- : GAME - Will send messages through 2781
Creator
I, [2011-05-25T16:17:06.388214 #2781]  INFO -- : GAME - New player has arrived Creator
Anonymous
I, [2011-05-25T16:19:33.452467 #2781]  INFO -- : GAME - New player has arrived Anonymous
```

Figure 4: Picture shows log output when game is created and connected by additional players.

As we can see in the screen shot we can see the log messages for the game creation and also for the two players joining the game (Creator and Anonymous).

4. Start the game. Once a player has joined the game we can start it by pressing enter on the creator window. We will start to see the description of the first test on the screen of both creator and joiner windows the following:



```
I, [2011-05-25T16:17:05.173991 #2780]  INFO -- : GAME CLIENT - Getting game_queu
e
I, [2011-05-25T16:17:05.174067 #2780]  INFO -- : GAME CLIENT - Will publish mess
ages in 2781-0
I, [2011-05-25T16:17:05.214677 #2780]  INFO -- : GAME CLIENT - Joining game queu
e 2781
PRESS ENTER TO START GAME
I, [2011-05-25T16:19:33.467424 #2780]  INFO -- : GAME CLIENT - Joined game messa
ge reached

I, [2011-05-25T16:20:08.142657 #2780]  INFO -- : GAME CLIENT - Game has started!
!
SELECTED NUMBERS
[
    [0] 10,
    [1] 6,
    [2] 75,
    [3] 9,
    [4] 5,
    [5] 8
]
TARGET
678
```

Figure 5: Picture shows the initial description of the first numbers test being printed out after game is started.

3 Conclusions

We have performed an implementation of the Cifras y Letras game with the following characteristics:

- Is easily extendable to add other logic tests.
- Allows for multiple players to play over the network.
- The game loop is based on the publisher-subscriber implementation pattern hence we are performing discrete simulation.

For future versions the game could be extended to include:

- Other logical tests.
- Implement a graphical user interface:
  - If you decide to keep ruby executing on the standard machine (developed in C) you can implement this with: GTK, QT, tk, FxRuby,... by searching the internet you will find that there are many bindings to popular GUI toolkits for standard ruby.
  - If we execute ruby on the java virtual machine (jruby, recommended for windows) you can use swing, JMonkey Engine or any other GUI toolkit that is usable from java.
- Add functionality to the game class so that the tests to perform (currently held in a vector in the initialize method) and other configuration can be read from a file.
- Active MQ - Stomp protocol does not allow for broadcast messages in order to broadcast messages to all game players of a game we have had to implement a message envelope with address solution. Substituting Active MQ for RabbitMQ or another message queue that supports the AMQP protocol would resolve this.
- Current timing is performed by forking a process with a sleep system call. Although it is effective it is a bit rudimentary and might not be portable. This could be improved by using the Event Machine library so that we can have pure ruby timed events and repeating events.