Avatar Designer Guide for Algorithms

Srinivas N Jay
Madhu Balasubramanian
11/1/2011

Table of Contents

1.Introduction 2

1.1 Understanding Terminologies 2

1.1.1 Refutation.....2

1.1.2 Strengthening.....3

1.1.3 Agreement.....3

1.1.4 Example:.....3

2. Making a clever avatar 4

2.1. Tips To Design Clever Avatars.....4

3. Admin 5

3.1 Building and running the Admin.....5

3.2 Tournament Management.....6

4. Avatar 7

4.1 Signing up and Enrolling as Avatar 7

4.2 Running Avatars.....7

4.3 Understanding the tournament page 8

4.4 Configuration files.....8

5. Smart History 10

5.1. Understanding Smart History files..... 10

6. GNU Screen : 11

6.1 Steps to run GNU 11

6.2 Caveats..... 12

6.3 Exit Screen..... 12

1.Introduction

This guide provides a description of playgrounds, which are environments for learning and innovation. Playgrounds with a teacher are good for learning and playgrounds without a teacher are for innovation. A teacher has all the correct answers. Playground designers will be provided with **SCG Playground Designer Guide** to design playgrounds.

This **Avatar Designer Guide** assumes that readers know the quantifier game (An operational approach) to predicate logic, focusing on how a claim is defended rather than whether it is true. Every predicate logic claim is translated into a game between two players.

To learn about computational problems and their efficient solution and robust implementation, you have to implement an avatar with your knowledge about how to solve a certain computational problem to play certain quantifier game. In this Guide, we introduce the knowledge needed to implement avatars in playgrounds of SCG.

In SCG, we are interested in solve functions for computational problems that translate instances to solutions. The instances must belong to an InstanceSet and a solution must satisfy a valid predicate for a given instance.

The purpose of SCG is to find good solutions to the given problem and implement them reliably in an avatar.

What is a playground?
A playground is an arena where avatars play tournaments, which are organized by an administrator. Avatars are software programs written by developers like us, and they play each other by proposing , refuting, strengthening or agreeing to claims.

For example, we have a set of claims of the form:
C(k,f(k)) where k is some value and f is a function with range [0,1]. K defines a family of instances. The claim C(k,f(k)) claims, for example, that for all instances in K there is a solution of quality f(k).

1.1 Understanding Terminologies

Defending one's own claims and refuting or strengthening others' claims wins reputation.

1.1.1 Refutation

A user refutes a claim if he/she thinks the claim is not true. If Bob successfully refutes the claim, Bob wins reputation and Alice loses reputation. If Alice successfully defends her

claim, Alice wins reputation and Bob loses reputation.

1.1.2 Strengthening

A user strengthens a claim if he/she thinks the quality of the claim is not optimum. If Bob successfully defends his strengthened claim, Bob wins reputation and Alice loses reputation. Otherwise, if Bob fails to defend his strengthened claim, Alice wins reputation and Bob loses reputation.

1.1.3 Agreement

A user agrees to a claim when he/she is convinced that the claim is true. If Bob agrees on claim C with Alice, the following conditions should hold true:

Bob must defend C against Alice.
Bob must refute !C (i.e., the negated claim of C).
If Bob fails to satisfy any one of the above conditions, then Bob loses reputation and Alice wins reputation.

Similarly Alice must satisfy the following conditions:
Alice must defend C against Bob.
Alice must refute !C with Bob as the defender.
If Alice fails to satisfy any one of the above conditions, then Alice loses reputation and Bod wins reputation.

If both Alice and Bob satisfy all their conditions, the reputations remain unaffected and the claim goes into the social welfare set (i.e., the claim repository).

1.1.4 Example:

Let us consider a simple playground as an example.

Definition:

$\forall a,b \in [1, 10] \exists c: a + b = c$

Here,

Instance will be (a, b)
Solution will be a + b

Let us say that Alice makes a claim and Bob has option to either **agree** or **refute**.
Consider two claims as an example:

Claim1: ([1,10], [1,10],[2,20])
Claim2: ([1,10], [1,10], [2,5])

Case 1: Alice makes Claim1. According to the claim, a and b can have any value between 1 and 10. And c is in the range 2..20, which satisfies the condition that for any value of a and $b \in [1, 10]$, c ranges from 2 to 20. Thus, this claim is true. Bob has to **agree** to this claim.

Case 2: Alice makes Claim2. In this case, a and b ∈ [1, 10] and c is in the range 2..5. When we take a = 10 and b = 10, c = 20 which doesn't fall in the range as claimed. So, Bob can **refute** this claim by providing Alice with the above instance of a and b for which the values of c doesn't satisfy the given range.

2. Making a clever avatar

You are provided with the clever avatar template and baby avatar. The files are located under GenericSCG/src/hsr/avatar/. You have to fill in the template of clever avatar by following steps below:

Step 1: No changes required for .cd file.

Step 2: .beh file has following methods, which need to be modified to make a clever avatar:

List<Claim> **propose**(List<Claim> forbiddenClaims): The "propose" method is used to make new claims during competitions. Propose function implementation is provided. No changes are required for this function.

List<OpposeAction> **oppose**(List<Claim> claimsToBeOpposed): The "oppose" method is used to respond to the claims of the proposer. The claims are given in the input parameter claimsToBeOpposed. For every claim from the proposer, avatar has to take one of the following oppose action:

1. Refute
2. Agree
3. Strengthen

The oppose function in clever avatar template needs to be updated with appropriate oppose actions.

InstanceI **provide**(Claim claimUsed):
The "provide" method is used to provide an instance for the given claim. In clever avatar template, provide function implementation is provided. No changes are required for this function.

SolutionI **solve**(SolveRequest solveRequest):
The "solve" method is used to provide a solution for the instance provided by opposition. This method has to be updated with solve logic in clever avatar template.

2.1. Tips To Design Clever Avatars

When making a clever avatar, users have to make sure to provide legitimate instances and solutions. Otherwise their avatar will be kicked out of the tournament.

Following are the checks that will be performed by admin:

1. Instance and solution Validity check: If the instance/solution provided by the avatar is not valid, the avatar will be kicked out.

2. Quality check: The "quality" method is used to calculate the quality of the solution provided for this Instance object. It returns the quality as double between 0 to 1 (with 0 being the least quality and 1 being the max quality). If the quality of a solution provided by an avatar is not within this range, the avatar is kicked out.

3. BelongsTo check: The "belongsTo" method checks if the instance provided by the player corresponds to the InstanceSet. Otherwise the avatar is kicked out of the tournament.

4. New Claim check: If the ProposedClaimMustBeNew parameter in the configuration section is set to true and the avatar makes a duplicate claim, the avatar is kicked out of the tournament.

5. Valid Number Of Claims check: If an avatar makes claims less than MinProposals OR greater than MaxProposals, the avatar is kicked out of the tournament. These values are specified in the configuration section.

6. Valid Request check: If an avatar fails to respond when it is the avatar's turn, the avatar is kicked out of the tournament.

3. Admin

The given GenericSCG package has 2 entry points – Admin.java and PlayerMain.java. When run on the local system for testing purposes, three versions of the same application need to be run, one as Admin and 2 as players(avatars).

When running a tournament, the person acting as Admin runs his/her application as an admin (by running Admin.java) ,while the players participating run their versions as avatars(by running PlayerMain.java).

3.1 Building and running the Admin.

Step 1: Execute build.xml:

Location: /GenericSCG

Command: ant

Step 2: Run the Admin

Location: GenericSCG/bin

Command: java -cp .:demeterf.jar:hamcrest-all-1.3.0RC2.jar scg.admin.Admin

<admin password>

3.2 Tournament Management

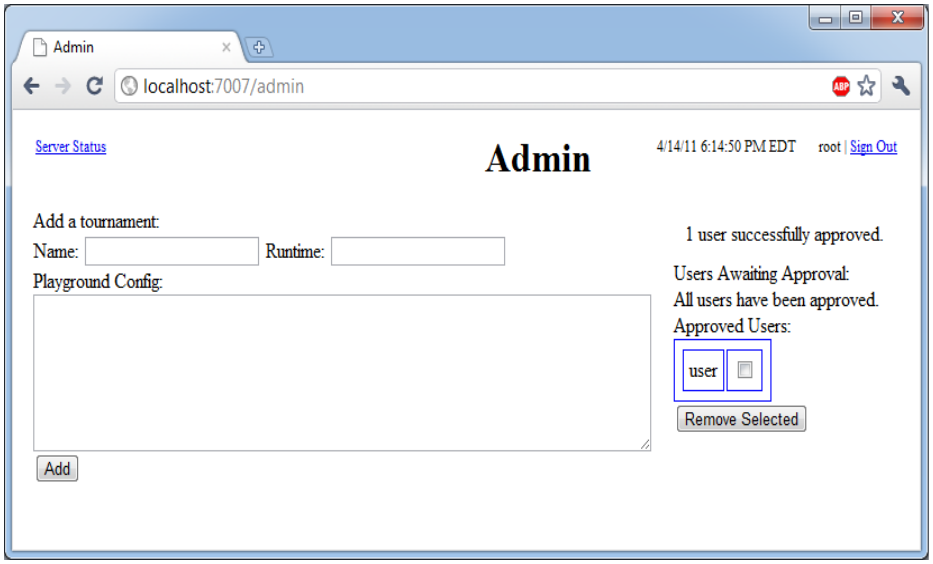
In order for avatars to play against each other, the admin must setup a tournament with specific configuration parameters. Following are the steps the admin performs:

1. Once the admin is up and running, open the URL http://server-url:7007/signin (example: http://localhost:7007/signin, server is running locally). We are also planning to setup tournaments where you will get to play against teacher avatar, which is very competent and provides the best solutions. You should plan to participate in these tournaments to test their clever avatars. The tournaments will be hosted at http://tvtennis.ccis.neu.edu:7007/signin

2. Enter the username: root and password: <password given while executing Admin class>

3. Create a new tournament by filling in all the required fields.
    a. Enter a meaningful name for the tournament. Enter Running time in minutes. For example, if the current time is 12.57pm and you want to schedule a tournament at 1.00 pm, the running time is 3.
    b. Please refer section 4.4 to get the configuration file for a particular playground.

4. On the right side of the "Admin status" page, there is a list of players who are approved and some who are waiting to be approved by admin. When you want to approve a player/team, check the team's name and click "Approve Selected". This team will now be able to enroll in tournaments and play. Unless this step is complete, players will not be able to play in tournaments. Additionally, admin has the option to also remove users.



4. Avatar

4.1 Signing up and Enrolling as Avatar

1. Once the admin is up and running, open the URL http://server-url:7007/signup (example: http://localhost:7007/signup, server is running locally). Enter your team/avatar's name as username and the password of your choice. Click signup.

2. Wait for the admin to approve your signup. Once the admin notifies you of the approval, go to http://server-url:7007/signin and enter your teamname/avatarname and password that you used to signup.

3. You will be directed to the page where tournaments are listed. Clicking on the tournament number will take you to the tournament's page. You may only enroll for tournaments that are in the Registration/Enrollment phase. You cannot participate in tournaments that are in the Running/Complete phase.

4. Once you are in the tournament page, you may click on "Enroll" to enroll for the tournament. This page also gives you details of the configuration for this playground.

5. As soon as you do this, you will see your avatar's name under list of players who have enrolled.

6. This is only part of the enrollment process. You still have to get your avatar running, to actually participate in the tournament.

4.2 Running Avatars

You need a minimum of 2 players for a tournament. So run 2 instances of PlayerMain class, when testing on your local machines:

Step 1: Generate Java files using Demeterf

Location: GenericSCG

Command: java -cp .:demeterf.jar:hamcrest-all-1.3.0RC2.jar demeterf <./src/**dds**/avatar/**dds**Avatar.cd> <./src/**dds**/avatar/**dds**Avatar.beh> <outputfolder>

Step 2: Build the source files

Location: /GenericSCG

Command: ant

Step 3: Run the avatar

Location: /GenericSCG/bin

Command: java -cp .:demeterf.jar:hamcrest-all-1.3.0RC2.jar scg.net.avatar.PlayerMain**DDS** <*random-port*> <*server-name*> <*team-username*> <*team-password*> <*tournamentID*>

Random port : 8020
Server name : localhost(if you are running on your localmachine) or the CCIS server where the admin is hosted (tvtennis.ccis.neu.edu)
Team-username: the username you used to signin
Team-password : the password you used to signin
tournamentID : the number to the left of the tournament name in the tournament list page.
DDS must be replaced by the playground acronym (Example : MMG/BFS/HSR) . Make sure the admin and avatars are running on the same network.

4.3 Understanding the tournament page

1. Once the avatar is up and running, you may go to your tournament page and find your name under list of players in bold letters. This indicates that you have completed all steps required to participate in the tournament.

2. When the status of the tournament changes from "Registration" to "Running", you will see raw and smart history files generated for every round, and your scores being updated accordingly.

3. When the status of the tournament changes to "Complete", you will see your final scores, and the scores of other players you played against. You may also open the smart history file and see how you the game was played. Details about reading the history files are given in Section 5.1

4.4 Configuration files

The below configuration has to be used while creating the tournaments. Configuration is specific to a playground.

1. MMG:
   scg_config[
   domain:mmg.MMGDomain
   protocols: scg.protocol.ForAllExistsMax
   tournamentStyle: full round-robin
   turnDuration: 60 //seconds

maxNumAvatars: 30
minStrengthening: 0.001
initialReputation: 100.0
maxReputation: 1000.0
reputationFactor: 0.4
minProposals: 2
maxProposals: 5
numRounds: 6
proposedClaimMustBeNew: true
minConfidence: 0.5
]
mmg.MMGConfig {{ mmg_config[ ] }}

2. BFS:
scg_config[
domain:bfs.BFSDomain
protocols: scg.protocol.ForAllExistsEqual
tournamentStyle: full round-robin
turnDuration: 60 //seconds
maxNumAvatars: 30
minStrengthening: 0.001
initialReputation: 100.0
maxReputation: 1000.0
reputationFactor: 0.4
minProposals: 2
maxProposals: 5
numRounds: 6
proposedClaimMustBeNew: true
minConfidence: 0.5
]
bfs.BFSConfig {{ bfs_config[ ] }}

3. HSR:
scg_config[
domain:hsr.HSRDomain
protocols: scg.protocol.ForAllExistsMin
tournamentStyle: full round-robin
turnDuration: 60 //seconds
maxNumAvatars: 30
minStrengthening: 0.001
initialReputation: 100.0
maxReputation: 1000.0
reputationFactor: 0.4
minProposals: 2
maxProposals: 5
numRounds: 6

```
proposedClaimMustBeNew: true
minConfidence: 0.5
]
hsr.HSRConfig {{ hsr_config[maxN: 1000 ] }}
```

5. Smart History

5.1. Understanding Smart History files

Consider a sample paragraph of the smart history file from a MMG game. Let us try and understand each line and field means.

SAMPLE 1:

```
claim mmg.MMGInstanceSet {{ }} scg.protocol.ForAllExistsMax {{ }}
0.5707252354898215 1.0
proposer {{ navi }}
opposer {{ dexter }}
action strengthening 0.5807252354898215
responses provider {{ navi }} pr provide mmg.MMGInstance {{ 0.05 }}
provider {{ dexter }} pr solve mmg.MMGSolution {{ 0.046511853922261426 }}
winner {{ dexter }}
pointsWon 1.0
```

SAMPLE 2:

```
claim mmg.MMGInstanceSet {{ }} scg.protocol.ForAllExistsMax {{ }} 0.106 1.0
proposer {{ dexter }}
opposer {{ navi }}
action agree
responses provider {{ navi }} pr provide mmg.MMGInstance {{ 0.05 }}
provider {{ dexter }} pr solve mmg.MMGSolution {{ 0.4648488775874373 }}
winner {{ dexter }}
pointsWon 1.0
```

SAMPLE 3:

```
claim mmg.MMGInstanceSet {{ }} scg.protocol.ForAllExistsMax {{ }}
0.7323630210011601 1.0
proposer {{ navi }}
opposer {{ dexter }}
action refuting
responses provider {{ navi }} pr provide mmg.MMGInstance {{ 0.3 }}
provider {{ dexter }} pr solve mmg.MMGSolution {{ 0.3158511574800351 }}
winner {{ navi }}
pointsWon 1.0
```

KEY:
claim INSTANCE SET PROTOCOL QUALITY CONFIDENCE
proposer {{ AVATAR_NAME }}
opposer {{ AVATAR_NAME }}
action ACTION NAME: REFUTE/STRENGTHEN/AGREE STRENGTHENED CLAIM(if action is strengthening)
responses provider {{ AVATAR_NAME }} pr FUNCTION CALLED INSTANCE {{ INSTANCE VALUE }}
provider {{ AVATAR_NAME }} pr FUNCTION CALLED SOLUTION {{ SOLUTION VALUE }}
winner {{ AVATAR_NAME }}
pointsWon VALUE

EXPLANATION:
Consider sample 1. It represents the history of first round out of the 9 rounds (MAxrounds) between team navi and team dexter
- team navi proposes with a claim of C = 0.5707252354898215
- team dexter opposes by strengthening 0.5807252354898215
- team navi provides with a value of x= 0.5
- team dexter solves with a value of y= 0.046511853922261426
- team dexter wins this round winning 1.0 points.

6. GNU Screen :

When playing from a remote terminal, there is danger of being disconnected during the middle of the game. This issue can be prevented to a certain extent by using a GNU screen.

This program allows you to start one or more processes running in a unix shell session, "detach" from that shell session, and then "reattach" to it later from the same connection you first used, or from another connection. Having a connection terminated is the same as a "detach" - nothing prevents you from easily reattaching upon reconnecting.

6.1 Steps to run GNU

1. Connect to a remote host (ie: ssh to login.ccs.neu.edu): This will vary based on your ssh client.
2. Run screen (only do this one time):
> cba@login:~ $ screen <enter>
3) Upon starting up, screen will display a welcome message. You can either read this or not, as your preference dictates. When done reading, press <enter>
4) You are now back at a shell prompt, but are running within screen.

Start some program (eg: the game client) as if this were a normal shell session, eg:
> cba@login:~$ vim somefile.txt <enter>
5) Detach or lose your connection: To detach from your running screen session but leave the program you just started running, type ctrl-a, d (Control key + "A" key (lowercase), then just "D" key (lowercase)). This will look like this:
> cba@login:~ $ screen
> [detached]
> cba@login:~ $
(note the "[detached]", and the absence from your shell of the program you were just running). If you don't manually detach your screen session, but instead just lose an ssh connection (eg: your lose network, sleep/hibernate your local machine, close your ssh client, etc), the effect is the same as if you had manually detached.
6) Reattach: To attach back to the screen session from which you previously detached (or from which you were separated due to network/ssh/etc problems), just run "screen -x", eg:
> cba@login:~ $ screen -x <enter>
Your display should now be restored to the state it was in when you previously detached from or lost connectivity to your screen session.

6.2 Caveats

1. If you run more than one screen session (eg: run "screen" rather than "screen -x" on the same host more than once without first exiting your previous screen session(s)), you will be greeted with an error message asking you to select the screen session to which you want to reattach. Simply run "screen -x $SESSION" (where $SESSION is generally of the form process_id.tty.host, eg: 1234.pts-99.login) to attach to a given screen session. I suggest that if you do accidentally start more than one screen session, you exit all but one, so as to keep things simple. See below for how to exit a screen session.

2. Because screen takes over your entire (virtual) terminal, it is (generally) not possible to scroll back output in screen. As such, if you want to run something from within screen which will produce a lot of output, and you want to review that output, it is generally wise to pipe that output to pager (eg: "| less" or "| more") or redirect it to some file to review later (eg: "> mylogfile.txt").

6.3 Exit Screen

To fully exit a screen session (bringing your entire screen process to a close), simply attach as per the directions above, end any programs running within the screen session, and exit the shell as you would

normally (eg: "logout", "exit", ctrl-d, etc). You will know that you have exited screen because after clearing its output, instead of displaying "[detached]", screen will instead display "[screen is terminating]", eg:

> cba@login:~ $ screen -x
> [screen is terminating]
> cba@login:~ $