Table of Contents

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 solution 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 admin. Avatars are software programs written by developers like us and they play each other by proposing claims, 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 satisfying property k there is a solution of quality f(k).

1.1 Understanding Terminologies:

Defending one's own claims and refuting or strengthening other's 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 $\in$ [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:

The students are provided with the clever avatar template and baby avatar. The files are located under GenericSCG/src/hsr/avatar/. Students 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 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 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:

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

Following are the checks that will be performed by admin:

1. Instance and solution Validity check: The "isInvalidInstanceOrSolution" method is used to check if the instance/solution provided by the avatar is valid. Otherwise 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).

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

4. New Claim check: The "getProposedClaimMustBeNew" method checks if the proposed claims are new. If they are not new, avatar will be kicked out of the tournament. In the code example section, it is illustrated how to make sure the claims are always new.

5. Valid Number Of Claims check: While creating a tournament, we provide configuration parameters specific to a playground. For example in HSR playground, the parameters, minProposals and maxProposals are set to two and five respectively. So the valid number of claims avatar should make in a round is in a range from two to five.

6. Valid Request check: Avatars have to take action when it is their turn. Otherwise avatars will be kicked out of the tournament.

3. Compiling and running Admin & Avatar:

3.1 Admin: 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 Avatar: Running an Avatar

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

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>*

DDS must be replaced by playground acronym. Make sure the admin and avatars are running on the same network.

4. Managing Tournaments & Users

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:

4.1 Tournament Management:

1. Once the server is up and running, open the URL http://server-url:7007/signin (example: http://localhost:7007/signup, 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 best solutions. Students should plan to participate in these tournaments to test their clever avatars. The tournaments will be hosted at http://tvtennis.ccis.neu.edu:7007/signup
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. Please refer section 2.2 to get the configuration file for a particular playground.
4. All the users who are willing to participate in the tournament must enroll in a particular tournament and then run their avatar (Step 3 of 1.2 should be done after enrolling into a tournament)

4.2 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: 20
   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[ ] }}

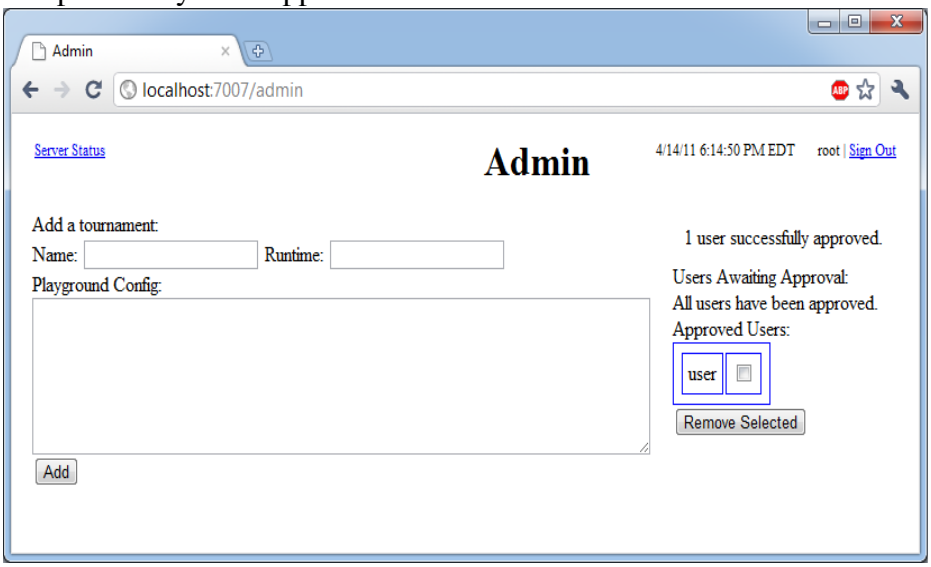
2. BFS:
scg_config[
domain:bfs.BFSDomain
protocols: scg.protocol.ForAllExistsEqual
tournamentStyle: full round-robin
turnDuration: 60 //seconds
maxNumAvatars: 20
minStrengthening: 0.001
initialReputation: 100.0
maxReputation: 1000.0
reputationFactor: 0.4
minProposals: 2
maxProposals: 5
numRounds: 6
proposedClaimMustBeNew: 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: 20
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] }}

4.3 User Management:

In order to play in a tournament, a player needs to enroll with the admin and sign-up for a tournament. Here are the steps to do that:

2. Sign up: Open the URL http://server-url:7007/signup to sign-up and user has to wait until Admin approves the request.
3. Sign In: Once the admin has approved the sign up request, user can login. The URL for login page is http://server-url:7007/signin
4. Approve/Remove users:
   Administrator can approve or remove users directly from the admin control panel. After logging in, pending users (i.e., "users awaiting approval") will be shown on the right. Additionally, the administrator can elect to remove users that had previously been approved.



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.