Requirements for Scientific Community Game (SCG) courts

Table of Contents

1. Introduction

Scientific Community Game (SCG) is aimed at providing framework for game developers to develop games which involve proposing and opposing claims related to constructive domains (e.g. computational science, engineering, etc.).

2. Functional requirements

1. Game designers should be able to define new game domains and protocols or reuse existing game domains and protocols and customize it according to their needs by providing new configuration in a config file.
2. Playground designers should be able to easily organize tournaments and monitor the status of the tournament (enroll, registration, running, complete state)
3. The avatar teams should be able to view different tournament by signing up into the SCG courts only once.
4. The avatar teams should be able to enroll into a tournament using their credentials created during signup. Once enrolled they should be able to access and download the config files and baby avatars for that game.
5. The avatar teams should be able to register their avatar when the tournament is accepting registration.
6. SCG court provides an admin for each game which is responsible for
    a. Running the game i.e. Requesting the avatar to take play their turn and validating the responses.
    b. Ensuring that avatars follow the rules of the game. If not they would be punished.
    c. Updating the reputation of avatars during the course of the game using the rules in Section5.
7. Once the tournament is running, then the avatar teams should be able to monitor the progress of their avatars and their competitors. SCG should provide relevant status information indicating the overall performance of the avatars in the tournament and also the performance of avatars in each game.
8. The SCG courts should also be able to provide relevant history files.
    a. A history file specifying all requests and responses exchanged between admin and avatars in a game.
    b. A Claim specific history file which shows the claim which was opposed and all the responses of the avatars. Also includes the result of the opposition along with reputation updates. The reputation updates should be according to rules in Section5.

3. Nonfunctional Requirements

1. **Usability**: Avatar teams should easily be able to enroll, monitor the status of the tournament and games. The avatar teams should be able to access the game history post completion of tournament. The history files should be structured such that the avatar teams can easily find the claims and responses exchanged between the avatars for a particular claim. This will give the teams an opportunity to learn and improve their avatars.
2. **Extensibility**: The playground designers should be able to add new tournaments by providing configuration file at any time. This should not affect any other ongoing tournaments.
3. **Reliability**: The system should be very stable. But if it crashes, all the tournaments being conducted should be able to be restored to their previous states and reputations of the avatars should be restored.
4. **Scalability**: The system should support 5 simultaneous tournaments and up to 20 avatars at a time. Several hundred signed users should be able to view the status of the tournaments.

4. Use cases

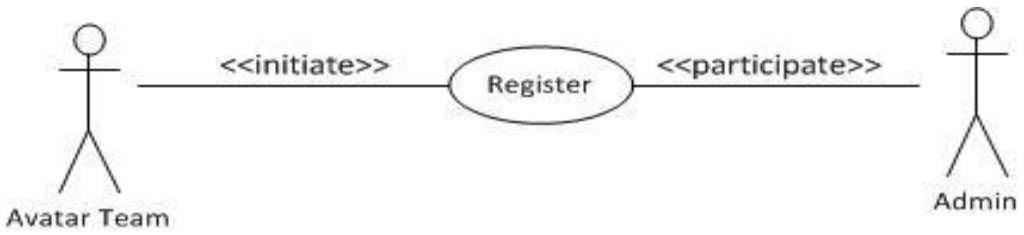
1. Signup

Use case name	Signup
Participating actors	Initiated by Avatar team
Flow of events	1. The Avatar teams enter their username and password in the signup portal. 2. The signup server verifies and validates the input. a. If the username already exists in the system, then the web portal notifies him. b. Else, the username and password is stored at the server.
Entry Conditions	Signup Server is started
Exit Conditions	1. The avatar team should be notified that his username and password can be used to access the system to view and enroll into tournaments, check the tournament status and history. 2. If the username already exists in the system, then the avatar teams should be appropriately notified by the system to re-enter the details.

2. Enroll

Use case name	Enroll
Participating actors	Initiated by Avatar team
Flow of events	1. The Avatar teams use their username and password to sign in to the enrollment portal. 2. The avatar team requests to be enrolled in a particular tournament from the tournament webpage. 3. If the tournament's maximum allowable avatars is not exceeded, then the team is enrolled. The avatar teams can now access and download the baby avatar executable and the config file for this game. 4. Else, the team is notified.
Entry Conditions	1. The avatar team has already signed up. 2. At least one tournament is organized and scheduled in the SCG court 3. Enrollment server is started
Exit Conditions	The avatar team is notified that they are enrolled for the given tournament and informed of the tournament schedule.

3. Register

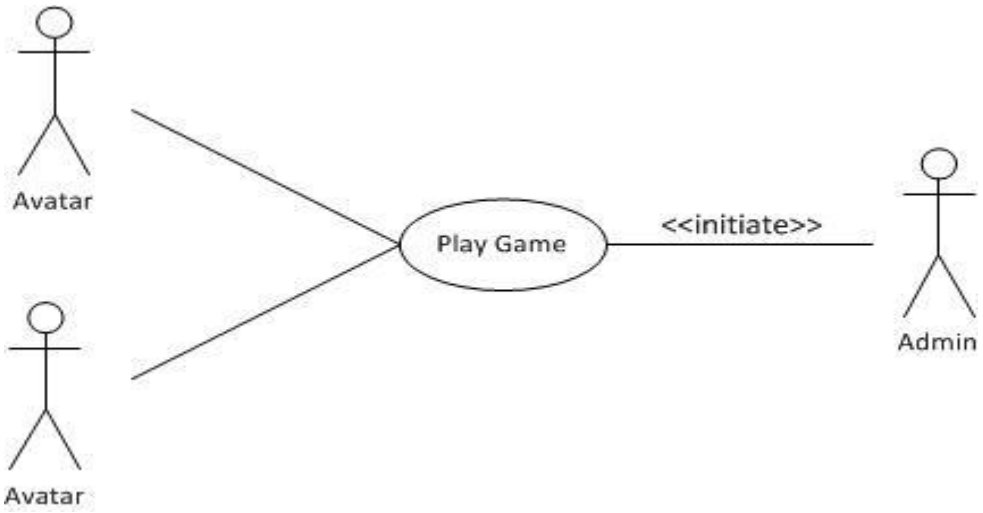


Use case name	Register
Participating actors	Initiated by Avatar team Admin participates
Flow of events	1. The avatar team registers the avatar using their username credentials and the avatar specifications - details of the port number and IP of the system in which avatar is executing. 2. The admin validates the avatar team credentials. 3. If the team credentials are not valid or if an avatar is already registered with that specification, then the team is appropriately notified. 4. Else, the avatar is ready to participate and admin has all required information to communicate with it.
Entry Conditions	1. The avatar team has to be enrolled for the tournament. 2. The current status of the tournament should be registration. 3. Admin should be ready for accepting registration.
Exit Conditions	Avatar is registered and ready to take turns.

4. Define Tournament

Use case name	Define tournament
Participating actors	Initiated by Playground designer
Flow of events	1. The Playground designer provides the config file. 2. The system validates the values specified in the config. a. If valid, creates a new tournament with the start date specified by the playground designer. b. If not, notifies the playground designer.
Entry Conditions	
Exit Conditions	A new tournament is created if the config is valid.

5. Play game



Use case name	Play game
Participating actors	Initiated by Admin Registered Avatars participate
Flow of events	1. Admin picks pair of avatars from the registered avatars based on the tournament style. 2. Admin chooses one of the the avatar to take the turn. 3. Admin creates a request based on config and protocol steps, sends the request and waits for the response from the avatar. The wait period is specified in the config (turnDuration). 4. The avatar receives the request from Admin and creates its response. a. If the admin receives the response from the avatar within the specified wait period, then it processes the response. b. If the response is not received within the wait period: i. The avatar is kicked off with the reason specified. ii. The reputation of the avatars are updated. iii. The game ends. 5. Admin validates the response with its corresponding request. a. If the response is valid, then it updates the avatar's reputation if needed and step 3 to 5 is repeated for the other avatar. These steps carry on for number of rounds specified in the config. b. If invalid, Steps 4(b) (i) to (iii) is carried out.
Entry Conditions	Tournament has to be in running state. Ie. Registration state has completed.
Exit Conditions	The game ends and reputation of avatars are updated.

5. Reputation Rules

When the game starts the reputations of the scholars/avatar are equal. This initial reputation can be configured by the playground designer using the configuration file constant.

Reputation is zero sum. i.e. if one loses the other gains.

In this documentation we are assuming that the person making the claim is Alice. Bob is expected to respond to Alice's claim – refute, strengthen or agree.

1. Refutation of a claim

Alice makes a claim C with quality q and confidence cf. Bob refutes C.

There are 2 scenarios for this:

- Bob successfully refutes the claim. So Bob wins reputation and Alice loses.
- Alice successfully defends her claim. So Alice wins reputation and Bob loses.

The reputation is updated as below:

Alice's new reputation = Alice's current reputation + (cf * result)

Bob's new reputation = Bob's current reputation - (cf * result)

Note:

result: value between -1 and +1.

1 indicates refutation fails i.e. Alice wins.

-1 indicates refutation succeeds. Bob wins.

Any value between -1 and 1 indicates that the partial success of the refutation

Example:

Consider a game using Secret Negative Protocol.

1. Alice claims C with quality q and confidence cf.
2. Bob refutes C.
3. Alice provides instance I_A with secret solution S_A.
4. Bob solves I_A and provides solution S_B.

If S_B >= q1 * S_A, then refutation is successful. [result = -1]

Else Alice successfully defends his strengthening. [result = 1]

Reputation Update: Alice's new reputation = Alice's current reputation + (cf * result)

Bob's new reputation = Bob's current reputation - (cf * result)

2. Strengthening of a claim

Alice makes a claim C with quality q and confidence cf.

Bob strengthens claim C with quality q1 and cf1 where cf1 >=cf. Alice refutes this strengthening.

There are 2 scenarios for this:

- Bob successfully defends his strengthened claim.
  - Bob's new reputation = Bob's current reputation + (cf* absolute(q-q1))
  - Alice's new reputation = Alice' current reputation - (cf * absolute(q-q1))
- Bob fails to defend his strengthened claim. So Alice wins.
  - Alice's new reputation = Alice's current reputation + cf1
  - Bob's new reputation = Bob's current reputation – cf1

Example:

Alice claims C with quality q and confidence cf

Bob strengthens the claim with q1.

Carry out refutation protocol with Bob as claimer and Alice as refuter.

If refutation fails (Bob defends and result between 0 and 1),

Reputation Update: Bob's reputation = Bob's reputation - (cf * absolute(q-q1))

Alice's reputation = Alice's reputation + (cf * absolute(q-q1))

Else refutation succeeds (Bob fails to defend his strengthening and result between 0 and -1)

Reputation Update: Bob's reputation = Bob's reputation - cf1

Alice's reputation = Alice's reputation + cf1

3. Agreement of a claim

Alice makes a claim C with quality q and confidence cf.

When Bob agrees on claim C with Alice, the following conditions should hold true.

- Bob must defend C against Alice.
- Bob must refute C' (C minimally strengthened along quality dimension using the configuration file constant) with Alice as defender.

If Bob fails to satisfy any one of the above condition, then Bob loses.
Similarly Alice must satisfy the following conditions:

- Alice must defend C against Bob.
- Alice must refute C' with Bob as defender.

If Alice fails to satisfy any one of the above condition, then Alice loses.

If both Alice and Bob satisfy all their conditions then the reputations remain unaffected and the

Claim goes into the social welfare set (Claim repository).

Example:

Alice makes a claim C with quality q and confidence cf and Bob agrees. Then the following steps indicate the agreement protocol.

1. Bob must defend C against Alice. Carry out refutation protocol with Bob as claimer and Alice as refuter.
   If refutation fails, refutation result is between 0 and -1 then, proceed to step2.
   Else, Bob loses.
2. Bob must refute C' (C minimally strengthened along quality dimension using the configuration file constant) with Alice as defender.
   Carry out refutation protocol with Alice as claimer and Bob as refuter.
   If refutation succeeds, refutation result is between 0 and 1 then, proceed to step3.
   Else, Bob loses.
3. Alice must defend C against Bob.
   Carry out refutation protocol with Alice as claimer and Bob as refuter.
   If refutation fails, refutation result is between 0 and -1 then, proceed to step4.
   Else, Alice loses.
4. Alice must refute C' with Bob as defender.
   Carry out refutation protocol with Bob as claimer and Alice as refuter.
   If refutation succeeds, refutation result is between 0 and 1 then, the claim C goes into the claim repository.
   Else, Alice loses.
   If Bob loses,
   - Reputation Update: Bob's reputation = Bob's reputation – cf
   - Alice's reputation = Alice's reputation + cf

   If Alice loses,
   - Reputation Update: Alice's reputation = Alice's reputation – cf
   - Bob's reputation = Bob's reputation + cf