DSoak: A Distributed Virtual Water Fight

Requirements

1. Introduction and Context

DSoak is a multi-player, distributed virtual water-fight game, where the players are fully automated processes. Users simply start them up and watch them buy balloons, fill them with water, and them at each other. A player can automatically look for available games and join one that is open but not yet started games. Then, one a game starts, the players compete for shared resources to stay alive and knock other players out of the game. Here are some key features of the system:

- At any time, there may be multiple games available for players to join, as well as other games in already progress.
- As soon as one game finishes, another is created and made available for players to join.
- There can be many active players at any time.
- A player not already in a game can select a game from the list of any available games and join it.
- When a player joins a game, it receives 200 pennies.
- Once enough players have joined a game, the game starts and launches resources managers: Balloon Stores, Water Sources, and Umbrella Suppliers.
- While in a game and resources last, a Player can
    - buy balloons from a Balloon Store
    - fill a balloon from a Water Source
    - throw a water-filled balloon at another Player
    - buy an umbrella from a Umbrella Supplier
    - Raise an umbrella as protection against being hit with a water-filled balloon thrown by another Player
- During a game, if 10 or more water-filled balloons hit a player, then that player will be ejected from the game.
- A game ends when a) there are no more resources, b) there are no more players, and c) there is just one player left. In the latter case, the last player in the game is the winner. In the former cases, the game is draw.
- Once a player is no longer in a game, it is free to join another game.

In DSoak, the players are fully automated processes. An end user may start multiple players concurrently. Besides players, there are five other types of processes in the DSoak system: registry, game manager, balloon store, water source, and umbrella supplier. The registry will manage a list of know processes and allow processes to discover each other. The game manager will manage one or more games and start up balloon stores, water sources, and umbrella suppliers as needed. Balloon store, water sources, and umbrella supplier each hold a limited stock pile of a resource that can be acquired by players and used to play the game.

DSoak Registry

There is always just one Registry running in system at any time, and always uses the same communication end point (IP Address and port). It keeps track of games and active players. It also keeps track of certain game and player statistics, e.g., how many games a player joins and how many it wins.

The Registry will try to determine if players are still functioning by periodically sending them YouAreAlive Requests. If a player does not respond in a timely manner, the Registry will assume that it is no longer functioning. When this occurs, it will forcible log the Player and send a notification to all game managers, so they know to remove the player from any game it might be part of.

The Registry can also provide information to player about available games on demand.

Finally, to handle an increased load, the Registry will start up Game Managers as the number of active player increases.

Game Manager

A Game Manager will create, publish, start, operate, and end games. It will register a newly create game with the DSoak Registry so Players can find available games and join them. It will also let the Registry know when a game has filled and is therefore no longer available, and after a game it will send player and game statistics to the Registry. Once a Game Manager finishes a game, it will create a new one.

Every game requires some number of Players. When enough Players have joined a game, the Game Manager starts the game. This involves launching one or more Balloon Stores, Water Sources, and Umbrella suppliers, and informing the Registry and the Players in the game that it has started.

When there is one only player left in a game, the Game Manager will end the game and notify the Registry of the winner and losers and their scores. It will also notify the winning player that it won. (Not that all of the other players have already been knock out of the game, so they know they lost.) Similarly, if there are no resources left, the Game Manager will notify the Registry of the draw and the player's individual scores, and it will notify the Player that game ended in a draw.

There can be many game managers running at the same time, and therefore many available games or games in progress. As the number of active Players increases, the Registry will start up more Game Servers to handle the load.

Balloon Store

A Balloon Store holds a limited supply of water balloons that can be purchased for one penny each. The balloons in a Balloon Store represents a shared resource and store itself is the resource manager. The Balloon Store will process requests in a first-come, first-serve basis.

Water Server

Like a Balloon Store, a Water Server is also a shared resource manager with a limited amount of a water resources. A player can send a balloon and two pennies to a Water Server and it will return the balloon filled one unit of water. The Water Server will process requests in a first-come, first-serve basis.

Umbrella Supplier

The Umbrella Supplier is another a shared resource manager with a limited amount of umbrella, but unlike the Balloon Store and Water Supplier, it will sell a umbrella periodically to the highest bidder (if there any bidders). A player can bid for an umbrella by sending a *BidForUmbrella* Request with one or more pennies to an Umbrella Server. The Umbrella Supplier will notify the winners and losers of each auction. The pennies sent by the winner are spent and cannot be used again. The pennies sent by the losers not spent and may be used again.

Player

To join in the fun, a *Player* must register itself, i.e. log in, with the *Registry* by sending it a Login Request. The Player can then ask the Registry for information about available games, as long as it remains functioning and logged in.

When a Player is not already participating in a game, it can select one of the available games and try to join it by communicating directly with the Game Manager that is hosting the selected game. However, the player must be fast. Each game has a limit on the number of players, so if a player is not properly obtaining game information from the Registry and efficient, it may get into the selected game.

Successful players must be able to process multiple tasks concurrently. For example, it may need to handle an Alive request from a Registry at the same time it is trying to fill a Balloon with water.

Each player can participate one game at a time, but is free to join another as soon as the first ends or the player is knock out of that game.

Within a game, the shared resources include balloons, water, and umbrellas. Players buy balloons from Balloon Stores, water from Water Sources, and umbrellas from Umbrella Suppliers. When a game server starts a game, it launches a random of these resource managers and gives them a limit number of resources.

Each water balloon can hold one unit of water. Every time a player is hit with a water balloon, it loses a life point. If it's life points become zero, then it is out of the game. Players can bid for umbrellas that can temporarily protect them from incoming water balloons. Umbrella stay active for 10 seconds.

2. Actors and their Goals

Since this application is a fully automated game, there are only two type of actors external to the system: an administrator and an end user. The administrator's goals are as follows:

- Start up the registry
- Review statistics and logs about the system's performance
- See the status of the system at a glance

The end user's goals are as follows:

- Startup game managers – as many as desired
- Startup players – as many as desired
- Watch the players play games with each other
- See the status of the system at a glance
- See scores and other statistics about the individual player processes

3. Functional Requirements (for the Player only)

1. Startup, Initialization, and Quitting
   1.1. After launching, a player needs to set up communications and get register itself with the registry using the Login protocol.
   1.2. When launching, a player may read parameters from a configuration file that govern its game strategy.

1.3. At any time, a player needs to be able to de-register itself with the registry using the Logout protocol.
1.4. If a player receives a Shutdown message from the registry, it should terminate immediately, but gracefully.

2. Staying Alive
   2.1. Whenever a player is logged in (registered) with the registry, it must be able to respond to an AliveRequest message from the registry as part of the Alive protocol.
   2.2. If a player does not respond to an AliveRequest, the registry will determine it to be dead and will not response to any request and will not validate the player for game managers.

3. Choosing a game
   3.1. If not currently in a game, the player should ask the registry for a list of available (not-yet-started) games, using the *Game List* protocol.
   3.2. If there are some available games, the player should choice one of them to join and proceed to do so. See Requirement 3.
   3.3. If there are no games are available, the player should wait some time, e.g. 5 seconds, and ask the registry again for available games.

4. Joining a game and waiting for it to start
   4.1. A player can join an available (not-yet started) game by communicating with the game manager for that game, using the Join Game protocol.
   4.2. If a player successfully joins a game, it then needs to listen for the initiation of a Game Status conversation by the game manager and waiting for a "Start Game" message to be received as part of that conversation. When that message is the received, the player can begin playing. See Requirement 4.

5. Playing
   5.1. When in an active game, the player can decide to buy a balloon, fill a balloon, throw a balloon, buy an umbrella, or raise an umbrella.
      5.1.1. When it decides to take an action, the player performs that action by communicating with the appropriate process.
      5.1.2. To buy a balloon, the player must communication with the Balloon Store using the Buy Balloon protocol
      5.1.3. To fill a balloon (one that was previous purchase but not yet filled), the player must communication with a water source using the Fill Balloon protocol
      5.1.4. To throw a balloon (one that was filled), the player must communication with the game manager using the Throw Balloon protocol.
      5.1.5. To buy an umbrella, the player must communicate with an umbrella supplier using the Umbrella Auction protocol
      5.1.6. To Raise an umbrella, the player must communicate with the game manager using the Raise Umbrella protocol
   5.2. When in an active game, the player needs to keep the Game Status conversation with the game manager going. This is a long-running conversation that supplies the player is information about processes in the game and the games status
      5.2.1. The player should keep track of the balloon stores, water sources, umbrella suppliers, and other players in the game.
      5.2.2. If the game is finished or cancelled, the player should go back to not being in a game.

5.3. When in an active game, the player may receive a Hit message from the game manager according to the Hit protocol, indicating that it was hit by a balloon

5.3.1. When hit, the player should reduce its life points by one. If the player's live points reach zero, it should exit the game and go back to the state of not being in a game. The game manage will consider it as out of the game.

5.4. When it is in an active game and has an umbrella raised, a player should respond to a Lower Umbrella message from the game manager according to the Umbrella Lowered protocol.

5.5. When in game, whether it's started or not, a player can leave the game by communicating with the game server using the Leave Game protocol

4. Non-functional Requirements

1. The system will be built using C#, .Net, Log4Net, Microsoft's Testing Framework. Other third-party libraries may be used as needed.
2. All software-development artifacts, including all documents and test data, will be kept in a Git repository accessible to all team members
3. Team members will commit changes to the repository frequently to a) ensure no artifacts are accidentally lost and b) coordinate with other team members

5. Future Features

1. A graphical display for each player that illustrates balloons being thrown
2. A monitor process that user can run to show current the registry information

6. Glossary

Program An executable piece of software

Process An instance of a program running