Distributed Systems 1st Homework

Artjom.Lind@ut.ee

October 1, 2015

The deadline for submitting is the 28th of October 2015. You can work in teams of 2. Do not forget to submit the names of your team members.

1 FCF Game "Frogs Catching Flies"

Multiplayer Network game. We have a game field of size NxM cells. We have two kinds of a character: a frog and a fly. The game runs in steps of ~500ms each, and we call it the frame rate of the game (2FPS). Players start in spectator mode - they cannot play but still they see the whole game field also all other players. In order to join the game player needs to specify his class (frog or fly). Then the player becomes active in the field - he can move character to different cell but his visibility is limited by his character's abilities (flies see better than frogs). Each cell is a location that can be occupied by a player character. Cell can not be occupied by two characters of the same class - we call it collision and prohibit movement. If a frog and a fly try to occupy the same cell than we count the fly eaten by the frog and frog earns +1, the fly-player loses all points and becomes the spectator. If the fly is not eaten by frog in 2 minutes the fly earns +1 point. If frog did not eat any flies during 2 minutes - frog dies because of hunger and frog-player loses all points.

Frog and flies have different movement abilities, as follows:

1. The fly can move one cell in any direction except diagonal, or stay in same cell.
2. The frog can move 1 or 2 cells in any direction (except diagonal and curves), or stay in same cell.
3. Spectators have no playable character

Each character has also a visibility area, so the players never see the game field completely:

1. The fly has his visibility area +2 cells in all directions (except diagonal) from its current location (5x5 square with a fly in the middle).
2. The frog has his visibility area +1 cell in all directions (except diagonal) from its current location (3x3 square with a frog in the middle).

3. Spectators see the whole game field

The player should provide the input for the next move in ~500ms otherwise the engine counts the player staying in the same cell. If a player manages to provide multiple different inputs in ~500ms (eg. forward and left) we count the last one.

2 The engine design

The engine of this game is quiet simple:

1. One 2-dim array of size NxM to hold the players positions, each cell may contain a particular players UUID or 0 if not occupied
2. Players array, arrays of player instances, currently in game or spectators. Each instance may contain player's UUID, player's name, class of character, earned points, timestamp of last score, timestamp when joined, player's provided next move (if provided).
3. The engine has just one loop which does the following:
   (a) process player's array, check if new players joined, process character selection, initialize player character into game field
      i. position the player randomly taking into account existing players in the field
   (b) process user input variable (up,down,right,left - fly, frog additionally 2xUp,2xDown,2xLeft,2xRight) and relocate player characters in the matrix
      i. this variable is populated by network loop (processing each player's input) or by bot (computer player thread)
      ii. if variable left unspecified (null) - player did not move the character in ~500ms
      iii. after processing input - zero the input variable (assign to null)
   (c) detect collisions, waste eaten flies, add points to corresponding frogs
   (d) detect survived flies, count the points for flies
   (e) detect frogs that should die because of hunger, waste dead frogs
   (f) sleep ~500ms
4. Engine API exposed to server
   (a) Initialize game (MxN)
      i. Create empty game field of size MxN
      ii. Create empty array for players
   (b) Add player (name)

i. Generate UUID for new player
ii. Create player, set name and UUID,
iii. Declare player spectator
iv. Store player into array of players
v. Return UUID

(c) Player join game (UUID, character)

i. Lookup player by UUID
ii. Set player's character
iii. Initialize player in the game field
iv. Set timestamp when joined, set points 0, set next move variable Staying, set last score timestamp to 0

(d) Player set next move variable (UUID, next move)

i. Lookup player by UUID
ii. Set new value for next move variable

3 Server design

1. Server is first initialized on particular port and creating the Game Engine instance.
2. Server should accept client connections and handle the clients as follows:

(a) When client connects

i. server registers client socket
ii. server reads client's name from the socket
iii. creates new player by name using engine's API
iv. server stores the pair of UUID and client sockets to identify connected clients

(b) When client wants to join the game

i. server receives player's character selection
ii. server sets player's character over engine's API

(c) When client sends input

i. server receives player's input
ii. server sets player's next move variable using engine's API

(d) If client connected irregardless of what did client send, server always sends back:

i. The whole game field if the player is spectator, or:

A. The player's visibility area (the rest of the game field should be shaded) dependent on the character's abilities

B. The player's current score

3. Sever once started should run forever, until explicitly interrupted
   (a) If server needs to be shutdown the clients have to be instructed first

4 The client

1. Client application starts with player's name and server address as command line arguments
2. Client connects to server becoming spectator first
   (a) if connected - server keeps feeding the spectator-client with current state of the game field (whole game field)
   (b) client application should show it properly to the user
3. Client joins the game by selecting his character class
   (a) server declares client active, and initializes him into the game field
   (b) server keeps feeding the client with his explicit visibility area (the rest of the game field is shady)
   (c) client application should show it properly to the user
   (d) client application at the same time collects the uses input (moves) in real-time and sends to server