Implementation of multiplayer PacMan game with fault tolerance

Tomas Kello
Czech technical university in Prague
Faculty of information technology
kellotom@fit.cvut.cz

Second Author
Institution2
First line of institution2 address
Second line of institution2 address
SecondAuthor@institution2.com

Abstract

1 Introduction

In this paper, we will focus on implementation of distributed application. Communication over TCP/IP protocol, and multi-thread application problems and solutions to keep data consistent. All this techniques we will show on project which goal is to implement multi player PackMan game. For better testing and starting game we implemented main application, which execute command to start program server, client or control them, add delay or close application. In summary we implemented:

- PuppetMaster - main aplication which controll other aplications.
- ProcessCreationService - service which can start application on local computer.
- Server - console application where whole game is maintained, collisions of objects calculated and broadcast new game state to clients
- Client - WindowsForm application. Contain game window with players. Sends instruction to server and receive updated state of game.

Game is based on round timing. Configuration of game contains length of one round and then application start to indexing all rounds as round ID. Round Id is during game hidden from the game perspectivity. Player can get information about game only from log. Where is written round id and state of the game at that specific time.

2 PCS - Process creation service

Is waiting on specific TCP address for Puppet Master to connect and execute commands, which are executed on command line of local computer. With this, it is possible for Puppet Master to start client application on different computers.

3 PuppetMaster

Execute commands from file or keyboard written to console. All commands are executed in parallel on separated thread. So it is important to create critical sections. Application contains list of connected clients, servers and PCS. This list is shared, so every*time new thread want to read. Need get exclusive access to list.

3.1 Added functionality

For better testing we extended application with new command "q", which send command to all application to close them including clients, servers, process creation service and after all,also Puppet Master is closed.

4 Structures used in program

One of the most used class is Game. Contain whole state of the game what mean RoundId. List of the Monsters, Players, coins and obsticles.

4.1 Player

Is derivated from Character and extended by player score.

4.2 Character

Derivated from position, what indicate current location of character. And class add direction of rotation in 2D.

4.3 Obsticle

Contains two object of type Position. First represent left top corner, and second represent width and height of object. All obsticles are represented as rectangles.

4.4 Position

We are playing 2D game, so this class contain coordinates X and Y.

5 Client Server shared functionality

To control game from Puppet Master. All application contains general functionality to influence behavior:

- Insert delay - define delayed one way connection between applications
- Freez - stop executing and sending instructions.
- Unfreez - start to execute operations
- Crash - exit application without letting now to anyone else

and to check state of application:

- Global state - function write general information about his state to the console containing round Id position of monsters, coins, obsticles and position and state of players. All this informations are stored in complex object of type Game.
- Local state - provide same functionality, but you can define round id, when you want to get update. This update is stored in the file of Puppet Master file system.

6 Server

Parametrized program for maintaining state of PackMan game. Possibility to set address, where game will run, number of players in the game and timeout for one round of game.

6.1 Server service

Offer opportunity to register new player for game. If game doesn't start yet, then his address is stored as active player. After enough players are connected, game is initialized and start of the game is broadcast. Second offered functions is SetMove. This function is periodically called by clients during the game. To notify server about required movement.

6.2 Architecture

Code is organized in Round-Rubin architecture with interrupts. They are important to receive instructions from clients. And then periodically are processed and new state of game is broadcasted. This template can be used in more application. And we tried to keep concept as universal as possible. To be able to fit program also for different kind of game.

6.3 Start game

Program initialize position of monsters, coins and client, assign them ID for chat and send them this state.

6.4 Set move

This function is triggered by client, independent on the intern state. Because this function is handled from interrupts, program only store value from client, and leave value for processing later by function Update game

6.5 Update game

Depending on time of timer, this function is triggerend periodicaly and update new position of monsters and players depending of they required direction. After new positions are applied. Program check monsters intersection with obsticles. In that case, monster direction is flipped. Then intersections of players are checked with:

- player: nothing happend. Players can share same position
- monster or obsticle: means Game Over for player. This information is set to him, and his character is removed from gaming place.
- coins: thats the tast of clients, to collect all coins. If intersection happen. Players score increase and coin disappear.

7 Client

Application with game plan and all characters drawn on it. This program read input from keyboard to navigate his character. In additions it is possible to define file, from were will instructions be simulated. Main core is triggered by timer to check keyboard state and send information to the server.

7.1 Client service

Offer multiple control functions for server:

- Game started - notify client about start of the game, configuration and other clients.
- Client disconected - ....
- Client connected - ....
- Game update - indicate new state of game, which should be displayed to the client. If client received request of local state, and ID is same as received Game state. Than this game is serialized and printed in client or Puppet Master application
- Get chat id - this function together with Get vector clocks are used to reconfigure chat vector clocks, after new client is added during game.
- Get vector clocks - return actual state of vector clocks representing number of received messages from every client.

7.2. Architecture

Application is designed has flow organized by events. That case a lot of concurent threads running in the same time. And we need to be very careful to keep data consistend.

7.3. Chat

Is implemented as P2P communication directly between clients. Server is used only to provide contact to other clients in the game.

7.3.1 Causality

P2P communication has problem with causality in asynchronous systems. Because some application can be delayed, then order of messages can be wrong. To deal with this situation. Application has implemented vector clocks. Where is stored sequence number of all clients message. Every client has his own id and number in that vector on position of id represents his number of sended messages. Then it is possible to recognize that some message is missing, and new message needs to be saved in list of pending messages. And append after missing message.

7.3.2 Reconfiguration

Application can deal with a lot of fault tolerance techniques. One of them is also reconnect new client. If new client detect, that the game started early. He ask every other client for his ID and vector clock. Than he can recognize missing ID, which will he use. And his sequence number will initialize to the maximum of all clients. So after he send new message, every other client will recognize his message as new without reseting clocks.

7.4. Language

Helvetica boldface type as in

Figure 1. Example of caption.

Long captions should be set as in

Figure 2. Example of long caption requiring more than one line. It is not typed centered but aligned on both sides and indented with an additional margin on both sides of 1 pica.

Callouts should be 9-point Helvetica, non-boldface type. Initially capitalize only the first word of section titles and first-, second-, and third-order headings.

FIRST-ORDER HEADINGS. (For example, **1. Introduction**) should be Times 12-point boldface, initially capitalized, flush left, with one blank line before, and one blank line after.

SECOND-ORDER HEADINGS. (For example, **1.1. Database elements**) should be Times 11-point boldface, initially capitalized, flush left, with one blank line before, and one after. If you require a third-order heading (we discourage it), use 10-point Times, boldface, initially capitalized, flush left, preceded by one blank line, followed by a period and your text on the same line.

7.5. Footnotes

Please use footnotes sparingly[1] and place them at the bottom of the column on the page on which they are referenced. Use Times 8-point type, single-spaced.

7.6. References

List and number all bibliographical references in 9-point Times, single-spaced, at the end of your paper. When ref-

[1] Or, better still, try to avoid footnotes altogether. To help your readers, avoid using footnotes altogether and include necessary peripheral observations in the text (within parentheses, if you prefer, as in this sentence).

erenced in the text, enclose the citation number in square brackets, for example [1]. Where appropriate, include the name(s) of editors of referenced books.

7.7. Illustrations, graphs, and photographs

All graphics should be centered. Your artwork must be in place in the article (preferably printed as part of the text rather than pasted up). If you are using photographs and are able to have halftones made at a print shop, use a 100- or 110-line screen. If you must use plain photos, they must be pasted onto your manuscript. Use rubber cement to affix the images in place. Black and white, clear, glossy-finish photos are preferable to color. Supply the best quality photographs and illustrations possible. Penciled lines and very fine lines do not reproduce well. Remember, the quality of the book cannot be better than the originals provided. Do NOT use tape on your pages!

7.8. Color

The use of color on interior pages (that is, pages other than the cover) is prohibitively expensive. We publish interior pages in color only when it is specifically requested and budgeted for by the conference organizers. DO NOT SUBMIT COLOR IMAGES IN YOUR PAPERS UNLESS SPECIFICALLY INSTRUCTED TO DO SO.

7.9. Symbols

If your word processor or typewriter cannot produce Greek letters, mathematical symbols, or other graphical elements, please use pressure-sensitive (self-adhesive) rub-on symbols or letters (available in most stationery stores, art stores, or graphics shops).

7.10. Copyright forms

You must include your signed IEEE copyright release form when you submit your finished paper. We MUST have this form before your paper can be published in the proceedings.

7.11. Conclusions

Please direct any questions to the production editor in charge of these proceedings at the IEEE Computer Society Press: Phone (714) 821-8380, or Fax (714) 761-1784.

References

[1] I. M. Author. Some related article I wrote. *Some Fine Journal*, 99(7):1–100, January 1999.

[2] A. N. Expert. *A Book He Wrote*. His Publisher, Erewhon, NC, 1999.