P-2-P GAMES FOR COMPUTER SCIENCE

Alessandro Amoroso and Gustavo Marfia

University of Bologna
Department of Computer Science
Mura Anteo Zamboni 7, 40127 Bologna Italy
Email: amoroso, marfia@cs.unibo.it

ABSTRACT

The main idea behind this paper is to take advantage of game design and implementation to learn several principles of distributed systems. This approach represents the other end of the spectrum with respect to the usual meaning of *edutainment*. The educational part is the building of a game, instead of playing that game.

This paper shows how several of the main principles of distributed systems have a "natural" mapping in distributed games, thus inducing their deep comprehension in the developers of the game.

We discuss an experimentation that lasted several years, involving students of computer science, at the forth year of their curricula in our University. The course in distributed systems requires the design and implementation of a distributed game, and the students are requested to perform that task in small groups. The students are requested to design and implement a Web based game between peers, avoiding any form of centralization during the game playing. Moreover, the game has to be tolerant to some kind of faults, keeping coherent behavior for the correct players.

Keywords— Distributed systems principles, educational entertainment, workgroup, homework.

1. INTRODUCTION

One of the most intriguing issues in teaching is to "keep alive" the topics of the course in order to capture students interest. This problem exacerbates when designing the homework, that neither has to be boring nor seem useless to the students. Requiring the design and implementation of games could be an excellent answer to these needs. A game has clearly stated goals and rules, nevertheless, it involves ludic aspects. The playfulness of this approach lightens the effort to accomplish the homework, and increases the mindfulness on the underlying principles [1].

A course in *Distributing Systems* presents several abstract concepts, that could be difficult to introduce into the synthetic scenario of an homework without resulting *artificial*. As an example, consider the *event ordering* and the *fault tolerance* concepts. On the contrary, the architecture of a distributed game *naturally* contains several of these aspects.

As noted above, a game has clear and well known rules and goals. Therefore, the corresponding architecture and functionalities expose relevant problems in the field of distributed systems. As an example, consider that a board game needs, at least, the followings: a reliable communication scheme, a shared common state, a leader election mechanism, and an ordering of the players. Moreover, any effective game deployed over a large scale network has to manage players that possibly leave the game due to a disconnection, or to a slow network response.

Since the goals of the homework are clear, even the debug and test phase of the distributed application, that usually is very tedious and obscure, results plain to design and perform. Moreover, this part could involve some genuine entertainment.

To make the homework more interesting to them, the students could choose the game to implement, providing the satisfaction of some general constraints. The strongest requirement is that the architecture has to be a kind of *peer-to-peer*, avoiding any form of *centralization* during the play. This requirement keeps away form a trivial centralized solutions, forcing the students to focus on distributed architectures.

This paper proceeds as follows. Section §2 outlines the main requirements for the game, that involve both the design and the implementation. The Section §3, shows how some of the fundamental principles of distributed systems maps into games. Section §4 discusses some of the most common trends in resulting games. The Section §5 concludes the paper.

2. THE GENERAL IDEA

The homework has an expected working time between 3 to 4 weeks of work for each student. In order to realize effective games, the students are encouraged to join together in group of three. Bigger groups facilitate lazy behaviors of some components, while smaller groups sources too much workload for its members.

This work was partially supported by the Italian FIRB DAMASCO project and by a grant of the Computer Science Department, University of Bologna.

2.1. Homework requirements

We present the main requirements of the homework and show their justification. The first requirements focus on *what* kind of system, the last ones focus on *how* to implement it.

Timed asynchronous system model: both the *synchronous* and *asynchronous* models are abstractions that rarely captures the reality of a distributed system. We adopted the *Timed Asynchronous Distributed System Model* [2]. The students have to set some reasonable deadlines in order to adjust the system responsiveness;

Shared common state: it strictly depends on the kind of game, and represent the local view of each player. It shows the same data to any player, and it has to be strictly coherent [3];

Reliable communications: to explicitly avoid the impossibility of consensus [4]. Moreover, the communications uses standard TCP protocol, excluding UDP. The latter is usually blocked by firewalls, resulting scarcely usable over the Internet [5];

Simple fault model: the only kind of processes fault is the *player crash*, *i.e.* a player that suddenly do not responds to any message. All the other components of the system are reliable. This condition is likely enough to introduce fault tolerance issues, and does not introduce too complex faults to manage. Note that, in the case of gaming, arbitrary faults are equivalent to cheating, and, for the aim of the homework, they would be useless complex to manage;

Tolerant to multiple faults: sometimes, managing a single fault could be tricky and misleading with respect to a general approach. Therefore, the game must correctly continue even if all but one players crashes;

At least four players: requires an effective distributed architecture to manage the game;

Peer architecture: as discussed above, this requirement avoid trivial centralized solutions. The only form of "authorized" centralized architecture is the registration of the players into a Web service, *e.g.* a server at a well known address. This first phase not influences the play, and has the sole scope of setting a new game, where the players are aware of each other;

Java programming language and RMI: the Java programming language has several primitives for fast implementation of effective prototypes of a distributed system. Usually, the students have learned Java during previous courses, but they are not familiar with techniques to implement distributed applications, such as *remote method invocation* and *distributed mutual exclusion*. Moreover, the students are familiar with communication by means of sockets and tend to use them in any case;

The above requirements do not depict a "real" scenario for a distributed application, but they represent a "likely" subset of conditions.

Imposing reliable communications allow to focus on processes fault solely, and keeps affordable the complexity of the system.

Some apparently obvious topics, such as *security* and other kinds of communication protocols, are explicitly excluded because they are either out of the scope of the course, or because they are dealt in other courses. Moreover, several of these issues are too much complex and expensive to design and implement with respect to the "core" topics.

The adherence of the distributed game to the original set of rules is not required. Usually, the original rules does not consider distributed issues, or they could be useless complex to implement. The students can modify the original rules and define a more simple set.

3. MAPPING CONCEPTS INTO GAMES

In this section we discuss the mapping of some basic concepts of distributes systems with respect to gaming.

3.1. Main concepts

The following are the main issues presented by the homework.

3.1.1. Not centralized server

The *client–server* paradigm offers an easy solution to on–line gaming. A centralized server could store and manage the common state of the game, answering to the client moves. This kind of architecture is not interesting from a *distributed* point of view, therefore it is forbidden.

The absence of a game server, commonly leads to a *ring* architecture. This architecture is a "natural" choice because of its good match with respect to a game scenario, that usually considers turns and rounds. Fig. 1 shows an example of such an architecture.

3.1.2. Shared global state

In general, the shared state in a board game is the board, comprehensive of the position of any pawn and any other object.

In a card game, the shared state is the "table", including the played cards and any other object, such as the amount of the bets.

3.1.3. Message passing

The players could communicate each other solely by exchange messages. The absence of both a central server and a shared memory, forces this kind of communication.

In a board game a message could contain the final move of the pawn representing the playing user. In some cases, for a more fine grain representation of the game, any single dice

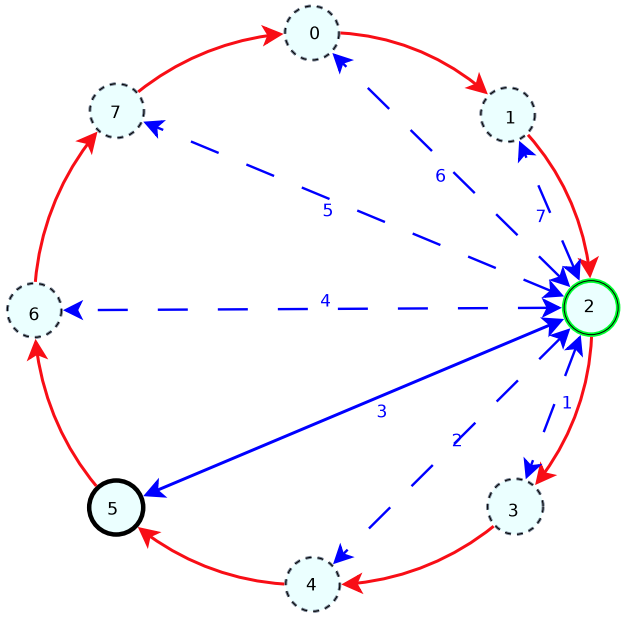

Fig. 1. Example of communication scheme. In this drawing player 2 interacts in turn with each one the players; specifically, it is interacting with player 5. The labels on the edges enumerates all the available communication channels of player 2. The solid edges implements the ring communication scheme useful both to broadcast the game evolvement to all the players, and to keep track of rounds.

throw could be broadcast to all the players to inform them on the game evolution.

In a card game a message could contain the card thrown by the player and, eventually intermediate moves.

3.1.4. Time and event ordering

Suddenly, the student realizes that clock synchronization could be very hard to obtain and basically useless. Instead, a *causal ordering* of events makes much more sense. Any move has to be delivered to the players preserving its order. Usually, the adoption of broadcast methods solves that problem.

Sometimes, the students implemented a textual *chat* service between the players. The messages are off–line with respect to the game, and could be used any time by the players. This case of asynchronous communication explicitly do not considers any message ordering.

3.1.5. Fault tolerance

Both the assumptions of reliable communication channels and of timed system model provide a simple *fault detection mechanism*. If a process, *i.e.* a player, does not answer to a *ping* message before a predefined deadline, we can assume that the process has *crashed*. On the other hand, we can assume that a process has crashed if it does not send a periodic *ping* before a predefined deadline. Moreover, the RMI mechanism throws an exception if the receiver of the invocation crashed or became unreachable.

A resilient player *re-joins* a game that it has abandoned because of a crash, while a crashed player usually remain unavailable until the end of the game. Resilient players are permitted, but advised against because they could be very difficult to manage, *e.g. Monopoly* and *Risiko*. In some board games the managing of resilient players could be straightforward, *e.g. snakes and ladders*.

3.1.6. Group membership

This problem derives by the above mentioned fault tolerance. In case of crashes, the system deeds to re-configure both its membership and its logical architecture, and, *i.e.* what are the surviving players, and how they communicate each other.

The initial group membership is defined by the centralized registration server, and therefore it is not an issue.

3.1.7. Reliable multicast

Since the communication scheme is based on TCP, a multicast is a sequence of reliable point-to-point communications. The most usual communication scheme follows a logical ring, therefore each message arrives from the *previous* player in the list, and it is forwarded to the *next* one.

The most subtle problem that could arise is a crash of a player during a broadcast communication. That scenario could prejudice the whole transmission.

3.2. Side effects concepts

The following concepts are marginal, but still significative, in the homework.

3.2.1. Leader election

The player that has the turn could be considered the leader. Usually this player is univocally identified by a kind of *token* exchanged while playing.

In the above discussed architecture, a crash of the leader is not a problem, because it can be simply detected, and managed, by its *predecessor* or its *successor* in the logical ring.

3.2.2. Code portability

The students are requested to present and discuss their homework into a specific laboratory. They prefer to develop their homework outside the facility labs, to avoid both time constraints (the labs are open 10 ours per day during the week days) and crowded environment (the labs offer about 120 computers to about 400 students overall). Almost all the students own a personal computer, and often they own two of them: a desktop and a laptop.

Due to this scenario, the students experience real problems in potability, both across different operating systems (*e.g.* Linux,

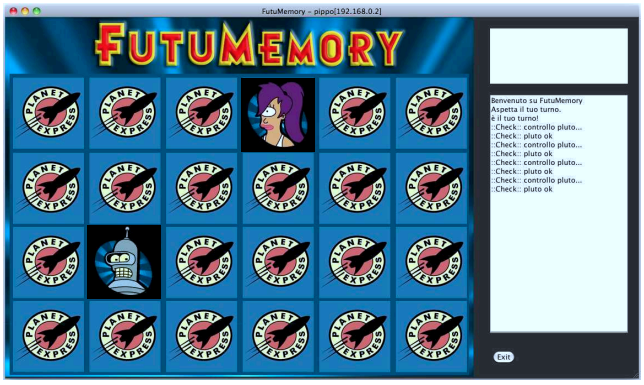

Fig. 2. Example of modified card–based game. A player wins a point when flips, in a single attempt, two cards showing the same picture.

Mac OsX, Windows), and between different flavors of the same OS (typically Linux).

3.2.3. Workgroup and versioning tools

The students need a cooperative development service to keep consistent the several copies of their running development environment.

3.3. Not an issue

The followings do not represent a problem in the homework.

3.3.1. Scalability

This is one of the distributed system principles that are, unfortunately, absent in the homework.

Usually the number of players is eighth at most. Therefore, scalability in terms of local resources and network bandwidth does not represent a problem.

3.3.2. Wireless and mobility

The requirements do not consider any kind of mobility. Nevertheless, some groups tested their games by means of portable devices, such as laptops and smartphones.

4. MOST POPULAR GAMES

The first step of the homework is a discussion with the teacher in order to sketch the initial design of the application. Any group of students proposes both the kind of game to implement, and their original idea of architecture.

The students usually implement pre–existing table games. This choice has the advantage of well-known rules, that could be simplified ad adapted to a distributed environment.

Sometimes, a group of students mix pre–existing games, or modify them introducing variations and new ideas. Fig. 2 is an example of such variations.

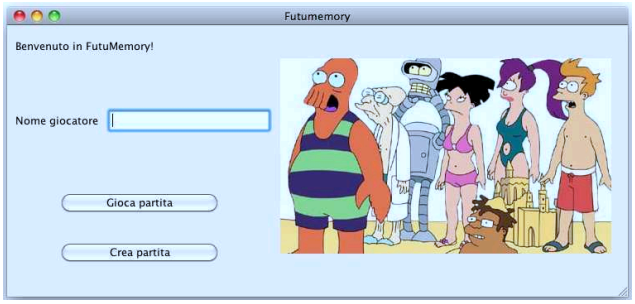

Fig. 3. Example of a game configuration. The player inputs its nikname and chooses between joining an existing group ("Gioca partita") or starting a new group of players ("Crea partita")

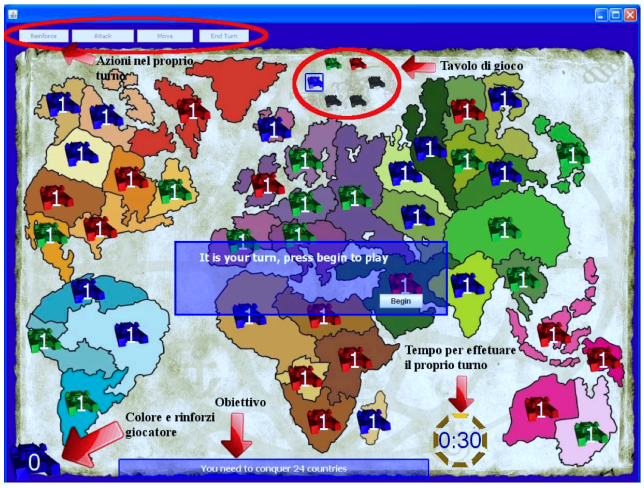

Fig. 4. Example of a board game. The picture shows the beginning of a turn. The top of the window shows buttons to activate the available commands.

To play a distributed game, any player needs to subscribe to a new game by means of a well known server, or Web service. Usually, a new player could join an existing group, that is waiting for the minimum number of players to start playing, or could define a new instance of the game. Fig. 3 show an example of that phase.

We can distinguish at least tho kind of games: board based or card based. The most popular games based on boards are: *snakes and ladders*, *Monopoly*, *Risiko*, *Cluedo*, *Trivial pursuit*.

4.1. Board games

Usually board games requires an initial distribution of resources, and possibly goals. These game evolves in rounds, at each round a designed player makes its moves, modifying the global state. Fig. 4 shows an example of such a board game.

Each player has a position in a circular order, and will eventually receive its turn for move.

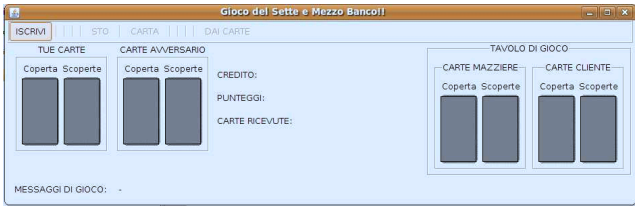

Fig. 5. Example of a card game. Since the interface is in Italian, we translate it in English. The top row contains actions: "Subscribe", "Pass", "Take one more card", "Distribute cards". The rest of the window summarizes the disposition of the cards onto the table with respect to the players.

4.2. Card games

Card games could be categorized in two groups, based on the interaction between the players. In some game, a server player interacts with all the other players on a round-robin basis. In other games, the players moves in turns. Fig. 5 shows an example of the first kind of card game.

5. CONCLUSIONS

In this paper we presented a different approach to educational games: the design and implementation of a game as a mean to learn abstract concepts. In the presented case we analyzed how a course in *Distributes System* could take advantage form such an approach.

We described an ongoing experience with students at their fourth (over five) year of studies in computer science. We adopted this approach several years ago, and the students constantly showed appreciation for this kind of approach. We are planning to continue on this track ad to refine and adapt the requirements year by year.

The proposed approach offers also an effective self–evaluation test to the student in order to check their knowledge. The objectives proposed by the homework are quite clear, and each student could easily verify if she or he meets them.

6. REFERENCES

[1] K. Sung, "Computer games and traditional cs courses," *Communications of the ACM*, vol. 52, no. 12, pp. 74–78, Dec. 2009.

[2] F. Cristian and C. Fetzer, "The timed asynchronous distributed system model," *Transactions on Parallel and Distributed Systems*, vol. 10, no. 6, pp. 642–657, June 1999.

[3] A. S. Tanenbaum and M. Van Steen, *Distributed Sistems. Principles and Paradigms*, chapter 7, pp. 273–317, Pearson, 2007.

[4] J. M. Fisher, N. A. Linch, and M.S. Paterson, "Impossibility of distributed consensus with one faulty process," *JACM*, vol. 32, no. 2, pp. 374–382, April 1985.

[5] V. Hadzilacos and S. Tueg, *Distributed Systems, 2nd ed.*, chapter 5, pp. 97–138, Addison-Wesley, 1993.