A participatory framework to support inclusive multi-playing for gamers in disadvantaged conditions

STEFANO CACCIAGUERRA, SILVIA MIRRI, MATTEO ROFFILLI, PAOLA SALOMONI
Department of Computer Science
University of Bologna
Via Mura Anteo Zamboni, 7
ITALY

Abstract: - As a consequence of the increasing mobile and multimodal devices diffusion, on-line gaming is more and more often considered as a dimension of an anytime, anywhere and anyone spaces. Providing accessible games is one of game industry and academic research purposes, in order to comply with the anyone issue in spite of disability. Such an issue assumes a crucial aspect when people with disabilities participate in a multi-player game. Players with disabilities access the game by using assistive technologies with alternative interfaces, which require more operational time than the standard ones, so as to become a new barrier to the participation. Gamers with mobile devices have to face a similar situation, by playing over unreliable networks, promoting the anywhere issue. Typically, small mobile devices impose the use of unconventional interfaces and such a context can compromise the chance of each player to justly take part in the game. We have developed a system that takes advantage of intelligent agents in supporting inclusive playability, even if disadvantaged gaming conditions occur.

Key-Words: - Multi-player Games, Pervasive Entertainment, Agent-based Entertainment, Intelligent Interactive Games, Interactive Entertainment for Special Populations (handicapped, children, elderly)

1 Introduction

On-line entertainment can be intended as a central quality of life issue. More general, game playing is the basis for discovery, learning and growing. Furthermore, playing with other constantly engages people in mutual communication [8]. Accessibility is a well-known issue, supported by different technical specifications and also regulated by law in several countries. From a technical point of view, users with disabilities have to be considered as people accessing in unconventional conditions. Users with disabilities can need specific hardware technologies to interact with a computer, which range from customized keyboards and pointing systems to eye tracking. In other cases they use specific software, such as vocal input interfaces (e.g. devices based on speech recognition mechanisms to control the action) or screen readers (e.g. programs that collect screen information and read it). These assistive technologies can help users with disabilities to join in on-line games, which however require providing them additional support in order to effectively allow them to play [6]. Once in the game, further problems can occur: the difficulty level may not be controllable, making it impossible for a person with mobility impairments to play; crucial information may be given without closed captioning, making it impossible for hearing impaired persons to succeed in the game, whilst others could be provided only through graphical interfaces creating a barrier impossible to be overcome by blind users. Typically, the use of unconventional hardware and software can require additional operational time, if compared with the commonly used interaction styles. Furthermore, some disabilities are characterized by slow movements. Considering multi-player game context, only few games truly permit to users with disabilities to equally participate in a contest where other players are using conventional hardware and software devices. Indeed, this could prevent people with disabilities from playing.

The creation of accessible applications typically generate, as side effects, benefits for different groups of users, including people using small devices or slow Internet connections. Such benefits are often referred to as “curb cuts”. The advantages of this new and fast increasing market, which are usually considered limited to a small group of users, could become a lever to spread game accessibility. The rising availability of network connectivity and the improvement in hardware miniaturization are promoting the expansion of mobile entertainment. Players wandering around desire to participate in multi-player games through applications that should guarantee high playability standards. They are using unconventional devices with small screens and limited keyboards that slow down the interaction. In this sense, the playing conditions of such users share the same limitations that affect people with disabilities. The network itself can further contribute to create an additional disadvantage. In multi-player games, wandering users may experience wireless
communication problems, which create a complex scenario to investigate. For instance, what happens when the mobile gamer wants to continue playing while she/he is moving from one access point to the other? In this case, unexpected delays and packet losses are caused by handovers, transmission errors and temporary link outages. At a higher level, a user may not be able to send or may send with a significant delay her/his actions to the Internet game system, losing interactivity.

In the above mentioned scenarios users would like to participate in a multi-player game but they are operating under difficult conditions that limit their competitiveness and prevent them from actively participating in the session. In this paper, we will present a new approach to support inclusive playability in multi-player games by using a software prototype to enhance interactivity and coherence when disadvantaged condition occurs. We have taken into account two circumstances:

- users can lose a move and this could be due to a network outage;
- users can be late, due to the network delay or to a slower control of the game, when using unconventional devices (either an assistive technology or a mobile terminal). Frequently, in multi-player networked game this condition is considered as a set of lost moves.

Our software prototype uses an intelligent agent that overcomes the above-mentioned limitations. In particular, when fault conditions occur preventing users from controlling the game, our system architecture provide at server side a ghost mechanism that seamlessly substitutes the user. While a user is playing, an intelligent agent is bound to her/him. The intelligent agent has been previously programmed to run pre-fixed algorithms in order to achieve some goals during the game. The agent continuously monitors the state of the user waiting for delays or lacks of control. If they occur, it starts to control the user’s character avoiding game interruptions and slowdowns. When the user regains control and is able to send moves, the agent immediately shifts in the background leaving the control.

The reminder of this paper is organized as follows. Section 2 illustrates specific classes of multi-player games engaged by users in disadvantaged conditions. Section 3 highlights design issues to support playability for all gamers. Section 4 describes the overall architecture based on a multi-agent intelligent system. Section 5 presents experimental results and, finally, Section 6 concludes the work.

2 Multi-Player Games

Many phenomena such as network troubles, limited device capabilities and user difficulties can affect the game, causing the loss of one or more actions and thus disadvantaging the player. All these problems might cause a short or prolonged interruption. For this reason, users in these disadvantaged conditions can be temporarily left out of the game - from a few seconds up to ten minutes or more - causing them to lose one or more consecutive moves. All the disadvantaged conditions discussed above can be considered from the system point of view by analyzing the effects they produce. In particular, we can distinguish two main interruption classes that are differently managed in different games:

- short interruptions, i.e. brief connection failures that cause the user to lose a few moves (actions or turns, depending on the game). The conditions that can produce such an effect could be, for example, vertical or horizontal handovers, transmission errors and temporary link outages, hard-to-use input hardware interface of small devices or user disabilities. This kind of effects could occur several times during a single match since some of the original causes are unavoidable;
- long interruptions, i.e. long failures that cause the loss of a sequence (sometimes very long) of moves. This effect could be produced, for example, by disconnection or errors in the device or in the application that suddenly suspend the game session. These effects are unpredictable and critical for game playability, but they occur with a very low frequency and the original causes could be removed by overcoming the fault.

Obviously, these classes do not reflect the complex and varied set of conditions that disadvantaged players may be faced with, and specifically users with disabilities. A subset of these persons can be effectively represented by the two effects of the disadvantages listed above.

3 Design Issues

Our intention is to offer an approach that allows users to play fast-paced multi-player games even if their conditions cause one of the problems mentioned in the above section. Many games do not take into account if a user is in a critical condition and is not able to send an action to the system. Hence, the game goes on evolving and cutting out on an a priori basis all users in disadvantaged conditions. A possible way to avoid similar
situations is to adopt turn-based games, where each player can only act sequentially at his/her turn. This mechanism is at the basis of inclusive gaming participation. In order to sustain a high number of players and to support a fast-paced evolution of the game, it is necessary to adopt a real-time turn-based approach. Using this approach, we enhance playability by guaranteeing interactivity and coherence for all gamers [3].

Our idea is to place the game system on the Internet allowing users to participate in it from any device (e.g. assistive and mobile technologies) and anywhere (e.g. exploiting wireless adapters). The interface should allow players to interact with it, by adopting a special framework that implements a playing session between the user and her/his avatar. Thus the framework can distinguish whether the game system is not reachable or whether it is reachable with additional delay. Based on these conditions, at the user side, the framework decides whether to wait for another gaming acknowledgment from the system or to recover the playing session. On the game side, the framework is able to detect if the user (on her/his side) is in disadvantaged conditions. If so, it activates an ad-hoc module, called ghost player, which takes the control of the player’s avatar until the problems are solved.

In this way, the system is not affected by single player faults and the game continues its evolution. This allows the game to maintain a fast paced evolution that is independent from latencies and losses arising from disadvantages of single players. Hence, the game can maintain the interactivity under a sensorial perceptivity threshold, providing a realistic look and feel on the final interface. In this sense, for example, unpleasant experiences for players such as low quality scenes with grainy video or jumpy sounds on the user interface are as limited as possible. In this situation, interactivity is related to the latency between action generation on the user device and its realization on her/his output interface via game system communication. A key point of the proposed framework relies on deciding which is the best action ghost player has to play when it is in charge of the orphan avatar. Two competing needs have to taken into account: from the player’s perspective the ghost player should choose the most predictable action while from the others’ perspective the ghost player should mimic the user’s behavior [1].

The first need can be clearly satisfied imposing to choose a default action that is natural for the current state of the game and can be easily predicted by the gamer. For example, the dead reckoning is a typical instance of this situation: an avatar continues to move in the same trajectory even if no direct control is available. This kind of choice can be successfully adopted if the loss of control is limited to a low percentage of the whole actions. In other cases, an avatar which keeps a hard-coded default action for a long period will be recognized as driven by software very quickly by other players. The advantages of this approach are easy implementation, using a look-up table of state-action pairs, and easy forecast of a user’s action. The second need is related to game coherence: if a player is disconnected for a long period, her/his avatar will probably start to show non-human behavior. A continuous up-and-down participation of a large percentage of players will cause the same situation. An approach to keep game coherence over a playability threshold should take into account the stochastic/strategic behavior of human beings. Historically, this consideration prompted game developers to increase the complexity of the algorithms that drive software opponents. The use of more complex algorithms will make it more difficult to recognize a prefixed set of behaviors. This approach can be a valuable tool for software opponents, who are clearly non-human, but it is not adequate to drive an avatar. The rationale is that the increased complexity can only delay the time required by other human players to recognize a disconnection. Another difficulty is related to the development of such algorithms which have to take into account and react to a huge combination of events and situations in a way that must be easily forecasted by the disconnected user.

We promoted an idea to overcome all these issues by reproducing the strategy of each user when she/he is not able to send actions to her/his own avatar [4]. The idea is to monitor each user during a normal game session in order to recognize the typical pattern of her/his behavior and to instruct the ghost player on her/his strategy. In doing so, not only the ghost player will be able to behave as a general human player but also as the disconnected user being monitored. The avatars of two different disconnected users will thus behave differently in the same situation. From the user’s perspective, it will be clear to forecast how the ghost player will act during the disconnection, while for other participants it will be difficult to recognize a disconnection by analyzing the avatar behavior. Obviously, some difficulties have to be solved to reach this level of mimesis.

First of all, in order to capture the essence of the user’s strategy, the driving algorithm must be adaptive. At the moment, machine learning techniques [5] are good candidates for this task. This
A particular class of algorithms is able to solve a problem by analyzing a collection of task pairs (instance, solution) while not knowing the dynamics of the solution (i.e., without formalizing the algorithm). In this sense, we talk about algorithms that learn (or are trained) to mimic a given task solver. If we consider the player as a solver who makes a decision (action) in a given situation (environment), it is clear how to implement the above idea. Each player will show a bias in choosing a particular solution amongst the admissible ones, one that is strictly correlated to her/his own strategy/ability. By collecting a sufficient number of game situations, the ghost player provided with machine learning capabilities will be able to resemble the user’s strategy/ability with a good level of detail. Another important feature of machine learning is generalization, i.e., the ability to produce good responses also in unknown situations.

A second difficulty is related to the computational effort required to achieve a good action. The ghost player is requested to produce an action in a short time, obviously shorter than the sensorial perceptivity threshold. Complex and adaptive systems often waste a lot of time searching for the best solution, since they need to compare the current situation to all those previously learned (i.e., the memory). Fortunately, current state-of-the-art machine learning algorithms, like as Support Vector Machines [9], are able to produce a compact representation of the knowledge model previously learned thus reducing drastically the time required to produce a solution.

A third difficulty is associated to the way in which the ghost player produces a model of knowledge. Two main strategies are available. In the first one, users play an off-line game session devoted to collect data in order to train the ghost player. This procedure is widely accepted as a customization phase. In the second strategy, the training is performed on-line during a normal game session. The advantage of this approach is that users can directly join the game avoiding boring training sessions. The off-line session often achieves better results than the on-line one, since the learning algorithm can access to the whole set of instances during the learning phase. Indeed, a hybrid strategy can be adopted, limiting the off-line train session only to very common situations and then continuing the training on-line. If the mimicking mechanism works well, the coherence of the entire game is maintained and the other users will not be able to tell who is playing. In this sense, coherence is related to the uniformity of the evolution of the game rather than to the behavior of each user.

4 System Architecture
In order to promote the participation of disadvantaged users in a game environment, we developed a platform that integrates a Multi-Agent System with a special Participatory Framework. In this sense, the Multi-Agent System SPADES (System for Parallel Agent Discrete Event Simulation [7]) implements the game environment by setting up the rules of the synthetic world and programming the behavior of its agents. On the other hand, the Participatory Framework (PF) implements a playing session layer on the stack TCP/IP between the user and her/his avatar (Fig. 1). In particular, we implemented a PF that manages the interaction between players and their avatars independently from their disadvantaged conditions. As the PF integrated in SPADES evolves adopting a simultaneous movement turn-based approach, the time spent by the game system to collect all actions of the agents and to deliver the next scene to all players is shorter than the sensorial perceptivity threshold. It is important to notice that the actions sent by remote players might increase turn duration time to an unacceptable degree for the human sensorial perceptivity [2]. As consequence, the evolution of the entire game is slowed down. Hence, if a short interruption occurs because the player is not able to deliver an action to her/his avatar within this time threshold, the PF guarantees the coherence of the game evolution forcing the ghost player to control the slowed avatar. Or even worse, if a long interruption occurs, the PF must recover the playing-session allowing users to resume the control of her/his avatar. While the PF waits for user’s re-connection, the ghost player will control the avatar as during a short interruption.

In order to manage a single player, the PF relies on two coupled modules residing both on the user side (called Shell Participatory Framework; SPF) and the game side (called Avatar Participatory Framework; APF). The APF is in charge either of managing prolonged failures due to any kind of disconnection, or of handling brief temporary problems due to short interruptions. It checks for a TCP connection to its SPF, while the ghost player controls the avatar until it receives an input from its player. Then it accepts a connection to an avatar from a SPF, or it eventually recovers a previously instantiated playing-session if a long interruption had occurred. If the connection is active, the APF controls both the Action timeout and the TCP timeout. The former is used to prevent low interactivity from slowing down the evolution of the game under the sensorial perceptivity threshold. In
this way, it monitors the responsiveness of its SPF within a given Action timeout limit. Obviously, if the timeout limit is exceeded, the APF turns to the ghost player to control the avatar for the user. If the number of consecutive action timeouts exceeds a maximum value (i.e. the TCP timeout), it sets the state of a communication as “broken”. In this case, it shuts down the connection and invokes the listening phase ready to recover the playing-session.

On the other side, the SPF checks if its avatar is reachable or if a new connection is necessary. The first time, the SPF connects its APF to start a new playing-session through the session level protocol. This protocol makes sure that, on each turn, the APF sends a RequestAction event to the SPF. The SPF checks if it is possible to interact with its APF, waiting for a RequestAction event within a given Action timeout. If the timeout expires, it buffers the action, waiting for the next RequestAction event to arrive. If, in the meantime, the player generates another action, the buffer will hold the last one so as not to deliver an old action related to a previous situation in the game. After an amount of time equal to the value of the maximum admissible number of Action timeout (i.e. the TCP timeout) has passed without receiving any playing-session acknowledgements from the APF, the SPF sets the state of the communication as “broken”. In this case, it shuts down the connection and tries to recover the previously instantiated playing-session by reconnecting to its own avatar.

5 Tron: a Case Study
As case study, we realized a multiplayer game freely inspired to Tron (Fig. 2). In this game, it is possible to ride a vehicle around a synthetic arena by making sharp turns while a wall constantly builds up after its passage. The goal of the game is to make the opponents crash into the walls. In our prototype, some vehicles are standard software opponents while others are avatars driven by users. We tested the effectiveness of the PF over an emulated networked scenario where mobile players participate in the game, when two link outages occur. We compared the performances of the game evolution when five different managements of the timeouts are adopted.

![Figure 1: PF on SPADES architecture](image)

On the other side, the SPF checks if its avatar is reachable or if a new connection is necessary. The first time, the SPF connects its APF to start a new playing-session through the session level protocol. This protocol makes sure that, on each turn, the APF sends a RequestAction event to the SPF. The SPF checks if it is possible to interact with its APF, waiting for a RequestAction event within a given Action timeout. If the timeout expires, it buffers the action, waiting for the next RequestAction event to arrive. If, in the meantime, the player generates another action, the buffer will hold the last one so as not to deliver an old action related to a previous situation in the game. After an amount of time equal to the value of the maximum admissible number of Action timeout (i.e. the TCP timeout) has passed without receiving any playing-session acknowledgements from the APF, the SPF sets the state of the communication as “broken”. In this case, it shuts down the connection and tries to recover the previously instantiated playing-session by reconnecting to its own avatar.

5 Tron: a Case Study
As case study, we realized a multiplayer game freely inspired to Tron (Fig. 2). In this game, it is possible to ride a vehicle around a synthetic arena by making sharp turns while a wall constantly builds up after its passage. The goal of the game is to make the opponents crash into the walls. In our prototype, some vehicles are standard software opponents while others are avatars driven by users. We tested the effectiveness of the PF over an emulated networked scenario where mobile players participate in the game, when two link outages occur. We compared the performances of the game evolution when five different managements of the timeouts are adopted.

![Figure 3: timelines (A): time to send an action](image)

![Figure 4: timelines (B): time to send an action](image)

![Figure 5: timelines (B): time to send an action](image)

<table>
<thead>
<tr>
<th>Run</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>% of increased performance</td>
<td>0%</td>
<td>13%</td>
<td>16%</td>
</tr>
<tr>
<td>% of actions played by the ghost player</td>
<td>0%</td>
<td>29%</td>
<td>46%</td>
</tr>
</tbody>
</table>
In Figures 3-5, A represents the standard case we used as term of comparison for all other policies; no mechanism is involved. B adopts a static Action timeout. In simple words, if an action is not notified at the game side within the Action timeout period, the APF controls its agent. C, instead, dynamically changes the Action timeout based on previous statistics. The timelines of Figures 3-5 represent the time (in millisecond) each user took to play an action, while Table 1 reports how the use of these more appropriate timeout managements increase (in percentage) the number of turns scheduled by the game system with respect to case A (within the same time period). Hence, the number of turns per seconds of the game system, which we use as a performance benchmark, is increased up to 16% (see Table 1). In case A, “agent 1” forces the entire system to wait for its actions when the two consecutive outages occur. Graph B shows how the management of a static Action timeout prevents the problems deriving from late actions. This solution creates a sort of filter that discards late actions independently from the disadvantaged conditions. Graph C reports how to adopt a more adaptive management of the Action timeouts.

This approach takes into account the trend of other peer storing the statistics of the communication. In simpler words, it dynamically changes the timeout value depending on whether the communication with a player is having some problems or it is going on quickly. Furthermore, in case C number of the actions played by APF is greater than in case B. This is due to the dynamic Action timeout that tries to make the entire system more reactive by reducing the timeout value of all APFs. Unfortunately, this (not centralized) approach allows one or more APFs to play the actions instead of their players even if there are other APFs with higher timeout values that are waiting for a reply.

6 Conclusion

We have developed a software architecture able to support an inclusive playability for gamers in disadvantaged conditions. In essence, this approach prevents problems coming from unreliable networks, hard-to-use hardware interface and user disabilities that can involve the players in unpleasant gaming experiences. The interactivity of a single user and the speed of the evolution of the whole game can be improve by coupling an improved network framework with the idea to exploit a ghost player that tries to reproduce the behavior of a user when this is not able to interact with the game system. One of the main question emerged during the discussion on this topic was if a disadvantaged user that exploits our approach, really enjoys her/his-self in playing the game. Probably this effect occurs only over a certain number of actions played by the ghost player and in relation with their frequencies. Unfortunately, which is the number and the frequency still remains an open question and it is main topic of our future works. Sometimes, in particular cases, a disadvantaged user might consider invasive our approach. In spite of these final considerations, we believe that the possibility to include the widest number of disadvantaged users as possible represents a success.

References: