Using Mobile Phones to Control Desktop Multiplayer Games

Silvano Maneck Malfatti
malfatti@catolica-to.edu.br

Fernando Ferreira dos Santos
fernandofsan@gmail.com

Núcleo de Tecnologia da Informação Aplicada ao Desenvolvimento de Jogos – NTIGames
Faculdade Católica do Tocantins – FACTO
Av. Teotônio Segurado 1402 Sul Cj.01, 77061-002, Palmas/TO, Brazil

Selan Rodrigues dos Santos
selan@dimap.ufrn.br

Departamento de Informática e Matemática Aplicada
Universidade Federal do Rio Grande do Norte
Campus Lagoa Nova, 59072-970, Natal/RN, Brazil

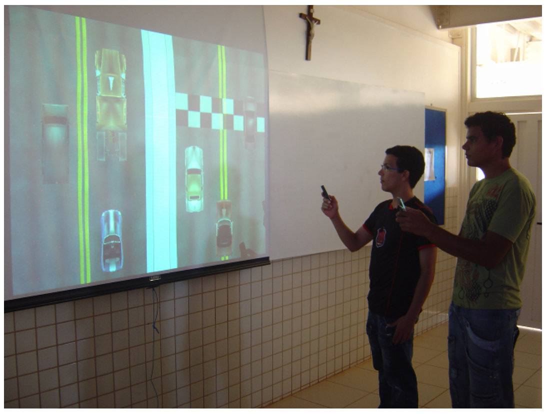


Figure 1: Users playing a multiplayer car race game controlled by mobile phones.

Abstract

This work presents a multiplatform architecture to support the design and implementation of desktop multiplayer games controlled by mobile phones with Bluetooth capability. Our main objective was to demonstrate that by harnessing technologies such as J2ME, J2SE, and Bluetooth communication it is possible to transform any mobile phone that support these technologies into a wireless application-independent remote controller. To demonstrate the flexibility of this approach, we have focused on employing mobile phones as game controllers. We developed four games in different genres/styles that support various input modes among players. The results gathered from our testbed games are twofold: i) in terms of overall game performance there was no noticeable communication delays, and; ii) in terms of gameplay, we have observed that the nature of the game interaction supported by this communication architecture has enhanced the social aspect of games—each game offered a entertaining environment in which a group of people could engage in.

Keywords: Bluetooth, desktop games, multiplayer games, mobile phones.

1 Introduction

The steady growth and advance of the video game industry is considered one of the most remarkable business development in decades, having surpassed the movie industry in gross sales since 2007. Part of this success is due to the evolution of graphics cards, which has allowed game companies to create visually rich and increasingly realistic scenes.

However, since the excellent graphics quality of today's games is no longer a novelty, game developers and designers have started to look for new ways to attract and entertain players. One recent trend is to provide new forms of interaction, replacing the traditional joystick for more elaborate devices, such as dance pads, guitars, drums, pointing devices, etc. Figure 2 shows three examples of currently available unconventional interaction devices.

Figure 2: New interaction devices, from left to right: dance pad, guitar, and the Wiimote.

Nintendo was one of the first companies to invest on a remote pointing device with motion sensing capabilities, called the Wiimote. This device enables players to interact with games by means of their natural body movements, gesture recognition and pointing [Wingrave et al. 2010]. The Wiimote-based games recognise rotations about the device's X, Y, and Z axes done by players holding the Wiimote, thereby tracking their hand movements. At the heart of this innovative controller are accelerometers and optical sensor technologies, which have been employed in Virtual Reality for decades [Burdea and Coiffet 2003][Sherman and Craig 2002]. Similarly, Sony corporation has announced its own version of motion capture controller for PlayStation$^{\text{TM}}$during this year's E3 press conference.

Following a slightly different approach, Microsoft has launched a system called Kinect, which employs cameras and sensors to capture the player's body movement and recognise gestures. This is a major improvement in terms of game interaction, since they have eliminated the need for any external controller—the natural body movement is captured and used as input data, thereby providing an intuitive game control [Microsoft 2010].

It has become clear that the major video game manufactures are interested in researching ways of providing new and innovative interaction paradigms for games. Therefore, the level of originality for such an user interface might be a determining factor for the actual success (or failure) of any new products.

This work proposes and evaluates a framework that enables ordinary phones with Java and Bluetooth support to function as wireless game controller for multiplayer desktop-based games. To demonstrate the flexibility of this approach we developed four multiplayer games that support distinct input mode. The result is a scalable framework that fosters great social interaction, and offers a natural interaction mechanism based on a pervasive technology, the mobile phone.

This paper is divided as follows. Section 2 covers some related work and Section 3 provides some background on Bluetooth communication and how to integrate it into Java applications. The following section describes the proposed framework, while in Section 5 we presented some prototypes developed to validate our framework. Finally, in the last section we offer our conclusions and mention some directions for future work.

2 Related Work

Likewise the game industry, the academy is also researching new interaction paradigms for game development. This is the case, for instance, when we consider the application of serious games to help rehabilitation or to function as complementary therapy [Golomb et al. 2010][Laikari 2009][Burke et al. 2009].

Another promising research topic is to use gesture recognition to receive user input for game interaction. Generally speaking, it is possible to point out two broad approaches to accomplish gesture recognition:

- *based on sensors*, in this case the computer system may employ specialised sensors such as data gloves, or general ones, such as motion trackers. The former are wearable gloves that can have something between 5 to 22 sensors assigned to key parts of a human hand. The latter are special sensors that measure the relative motion of objects (for instance the user's arm or head) in relation to a fixed system of coordinates [Machado 2010] [Teichrieb and Figueiredo 2010].

- *based on computer vision*, by employing regular cameras coupled with computer vision algorithms to recognise fiduciary markers or any parts of the user body such as hands, face, or fingers [Stenger et al. 2010].

The work done by Figueiredo *et al.* (2009), for instance, has introduced a framework based on gesture recognition via colour tracking to control real time guitar-based rhythm games. In their work, the player must wear a pair of coloured gloves to obtain a suitable contrast with the surrounding environment. The system continually captures images of the user playing a virtual guitar and estimates parameters such as position of fingers and speed of hand movements. These informations are then processed to generate a corresponding sound. Although their work has presented good results in terms of easiness of use and robustness, we believe that their framework has two possible drawbacks. Firstly, their gesture-based interface eliminates the need for the prop guitar, which might be a disadvantage since for some people it feels more natural to play holdings something that feels real than moving their hands in the air. Secondly, the processing time necessary for the gesture recognition might impose a limit on the number of simultaneous participants, depending on the processing capacity of the host computer.

Another project that has focused on an inventive way to promote player-game interaction is the CamSpace, developed by Israeli company CEO [Cam-Trax Tech. 2010]. In this project the developers have used computer vision algorithms to transform almost any object into a potential control. After a period of recognition and calibration it is possible, for instance, to use a fruit or even a soda can to interact with the game. One of the great advantages of their approach is to allow games to be controlled by ordinary objects. However, their system has a limitation typical of vision-based solutions: the players must be within the camera's field of view. This restriction might limit the range of motion for the players.

Mobile phones are another example of recent interaction modality that has been drawing interest from the game community. Although most of the popular mobile phones does not have enough processing power to run games with good graphics quality, they offer a great potential to function as a wireless remote (game) control. Mobile phones rely on Wi-Fi or Bluetooth technologies to send and receive data to applications. This exchange of information can be designed to follow distributed network architectures such as the centralised server-based or the peer-to-peer (P2P) models.

In Brazil, the number of mobile phones has been growing steadily and the government has predicted that by the end of 2011 the number of mobile phones will reach the total population of the country [Zmoginski 2010]. Should this prediction confirm, the game industry will have at their disposal a source of potential users who will be happy to extend their mobile phone's functionality by connecting them to digital TV set top boxes, personal computers, or even game consoles.

The Poppet framework [Vajk et al. 2008] is an example of the potential use of mobile phones as game controllers. Through this framework it is possible to develop games that run on large public display and receive input from the on-board sensors present in the player's mobile phones. The phone-game communication is done via Bluetooth. Their testbed game display on a large screen a shared game environment in which the players navigate their individual avatars with their own mobile phones. The movements applied to their phones were captured by the on-board accelerometer and sent back to the game controller central server.

Poppet is based on a client-server model. The client application was implemented using C++ programming language and the Symbian S60 3rd Edition SDK from Nokia. This SDK is necessary to grant access to 3D motion sensors, and is similar in functionality to the J2ME's Mobile Sensor API (JSR-256). Additionally, they run a Java application on the client side to establish the communication between a mobile phone and the server application. Figure 3 shows an overview of the Poppet's architecture. To reduce compatibility issues among different Bluetooth stacks, the server application was developed using the Frason Bluetooth SDK for C#.

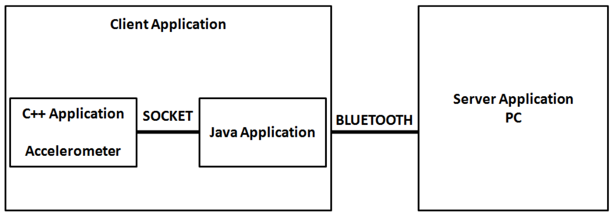


Figure 3: The Poppet's architecture, showing the server and client components.

Although functional, the Poppet architecture has two limitations: *i)* both the client and the server applications are platform dependent; on the client side, only Symbian based devices are able to run the application, while the server application run only on Windows-based machines; and *ii)* the mobile phones work only as input devices; the mobile phone's inherent potential to display information is overlooked in their framework.

The lessons learnt from this review have helped us to define some requirements that guided the design of a novel communication framework architecture called BlueWave. The requirements were:

- Improved accessibility: to achieve improved accessibility for both client and server applications, the framework should be implemented entirely with the Java language. This simple strategy enables anyone who owns a mobile phone with Bluetooth and Java support to participate in games.
- Platform independence: the server should be able to run on a variety of operational systems, such as Windows, Mac OS, Linux, etc.

- Two-way communicability: the mobile phone should transmit data both ways. This means that a mobile phone can also receive information, store it locally and display user-specific information on the mobile screen (e.g. the player score or game status).
- Scalability: the architecture should afford the design of multiplayer games.

The next section briefly describes how Bluetooth communications works, which will help to understand a more detailed view of the BlueWave architecture that follows.

3 The Bluetooth Technology Standard

Bluetooth is an international standard for open wireless communication. It has the (theoretical) maximum data transfer speed varying from 1 to 24 Mbit/s, depending on the version. Bluetooth utilises short length radio waves to exchange voice or data over short distances among different categories of devices [Thompson et al. 2008]. The complete specification for the Java Application Programming Interface (API) for Bluetooth can be found in the JSR-82 [Milikich 2010].

Two valuable features of Bluetooth devices are its reduced power consumption and relatively low technology cost, when compared to Wi-Fi or infrared, for instance. These features are factors that encourage the inclusion of Bluetooth support in mobile phones, whose design is usually concerned with power management issues and cost reduction of its components.

Another advantage of Bluetooth over other wireless communication technologies is the ability to create dynamic networks without the need for an access point nearby. A Bluetooth network operates under the concept of master-slave: one device acts as master and manages the connection with up to seven slave devices [Huang and Rudolph 2007].

The implementation of Bluetooth communication is based on a protocol stack. The protocol stack provides a standard approach to the design of system architectures for radio communication among Bluetooth devices. According to the Bluetooth specification, the protocol stack is divided into two levels: the upper stack and the lower stack [Huang and Rudolph 2007].

The lower stack controls the physical functionality of the device, and it is composed of the radio, the baseband, and the Link Manager (LM) and Link Controller (LC) layers. The lower stack determines how to manage the low level operations such as the conversion of the data from the upper stack into radio signal and vice versa [Kammer et al. 2002].

Considering the programmer's perspective, the upper stack define higher-level protocols that perform services such as the Service Discovery Protocol (SDP), which allows devices to gather information about the capabilities of other Bluetooth devices present in the area of connection; and the exchange of data or objects (OBEX) such as files, videos and pictures among connected devices. Hence, to create a Bluetooth enabled application it is necessary to utilise an API that implements the protocol stack, or at least part of it.

Programming a Bluetooth based application targeted for mobile phones may fall into two possibilities. The first option would be to use the development libraries provided by the manufacturer of the device. The advantage of this approach is to have full access to the resources offered by the device, though this might restrict the application in such a way that it would run only on a particular brand or model.

The second option is to use the Java language through the Java 2 Platform Micro Edition (J2ME). Although the J2ME does not implement the complete Bluetooth protocol stack, the applications implemented within the J2ME environment will have access to the most common Bluetooth features. As a bonus, J2ME based application ensure a standard and portable way to run across devices that support both the Bluetooth and the Java Virtual Machine (JVM).

Despite the Bluetooth programming support granted by the J2ME, the same does not hold true for its desktop counterpart, the Java 2 Platform Standard Edition (J2SE) [Goelzer 2006]. J2SE does not natively implement the Bluetooth protocol stack, therefore to develop a Bluetooth compliant application it is necessary to resort to an external API.

To overcome the aforementioned issue with the J2SE, there exist various API that implement the JSR-82 specification for particular operational systems. Nonetheless, just a few of them are cross-platform and open source libraries, as shown in Table 1.

Table 1: List of APIs that implement the JSR-82 for j2SE.

API	Supported OS	Licensing
Avetana	Win, Mac, Linux, Pocket PC	Open source[1]
Bluecove	Win, Mac, Linux	free (LGPL)
JavaBluetooth.org	Win, Linux, QNX	free
Harald	any system where Java runs	free

1: only linux.

To implement the BlueWave framework, we have chosen the Bluecove API for several reasons: it is a free API, maintained by a community of developers, has extensive documentation, is a mature API (firs released on 2004), supports the OBEX protocol, and does not depend on the Java Communication API (javax.comm).

The Bluecove is a JSR-82 JSE implementation that interfaces with various platforms, namely the Mac OS X, WIDCOMM, BlueSoleil, and Microsoft Bluetooth stack. The Bluecove works as an intermediate layer between the J2SE applications and the operational system's native Bluetooth protocol stack, as illustrated in Figure 4.

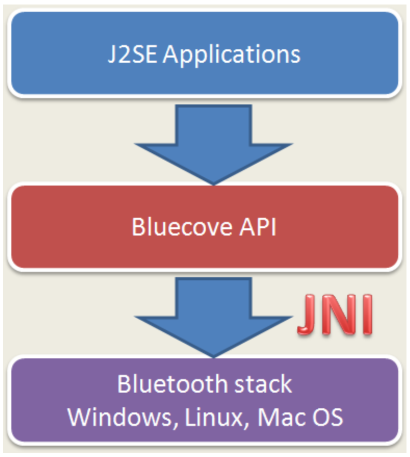


Figure 4: Schematic representation of a J2SE application that access Bluetooth communication via Bluecove API.

4 The BlueWave framework

This section describes the BlueWave, a generic framework aimed at designing cross-platform multiplayer games based on the Client-Server model for distributed applications. The server application runs on a host computer, whereas the client side may run on mobile phones or any portable device that have Bluetooth communication capabilities and run Java based applications.

The BlueWave design has been guided by the requirements listed at the end of Section 2, and shall be described in detail in the next subsections.

4.1 The structure of the client application

Before we describe how the BlueWave client application is organised, it is important to understand how the communication process among Bluetooth devices actually happens.

Because most of the devices that utilises Bluetooth are mobile, Bluetooth networks are often ad-hoc and also mobile. At any given time, a Bluetooth device might detect a new device entering or leaving its covering area. When a group of Bluetooth devices are linked, they form a computer network known as *piconet*. A piconet can interconnect a user group of up to eight devices. One device plays the role of *master*, while the others are considered *slave* devices. The devices can switch roles, by agreement, thereby allowing any slave to become the master. At any given time, data can be transferred between the master and any other slave device. Finally, the Bluetooth specification defines the concept of a *scatternet* as the interconnection among two or more piconets. To interconnect two piconets, a device (either the master or one of the slaves) shall simultaneously act as slave in another separate piconet.

The first step to establish a JSR-82 compliant piconet is to search for possible connections within the range of the master device. This process is called `Inquirity`.

The discovery process also depends on the discovery mode set for each Bluetooth device within the reachable range of the master device. Each Bluetooth device may assume three mutually exclusive discovery modes, namely *i)* `LIAC` (limited inquiry access code), this grants a temporary access mode that will revert back to previous mode after 1 minute; *ii)* `GIAC` (general inquiry access code), this makes the device visible to any inquiring device; and, or `NOT_DISCOVERABLE`, which disables the device visibility to any other device. In the BlueWave framework, the host on which the server will run shall be set to the GIAC mode, enabling the clients to discover it at any time and request a connection.

After the discovery process has been finished, it is necessary to provide a mechanism to allow the clients to choose which of the discovered devices they want to establish a connection to. Besides defining to whom the clients want to connect, they also have to define which service they wish to utilise. This is necessary because a device may offer several services simultaneously, such as data exchange or voice transmission. The entire process can be compared to establishing a connection between two host using sockets: the device name is equivalent to the IP address, while the list of services available are equivalent to the port numbers accepting a communication link.

The client application of our framework is based on three major classes that represent some of the entities and concepts discussed so far in this section. The threes classes are: `CBBlueClient`, `CDeviceList`, and `CClientCanvas`.

The first class represents the MIDlet[2] application and provides the methods `startApp()`, `pauseApp()`, and `destroyApp()`, typical of a J2ME application. This class also keeps a reference to the device display that manages the content to be shown on the client's screen.

The second class, `CDeviceList`, is responsible for displaying a list with the names of the devices that have been found during the inquiry stage, and allowing the user to select one of them to be the server (master). For that reason, this class implements two interfaces: `List` and `DiscoveryListenner`. The first interface defines methods to display an option list on the device display, whereas the second one defines methods to search for devices within the range of connection.

After the searching stage is completed, a `CDeviceList` object exhibits the list of device names found (see Figure 5 for an example). It is up to the user to choose the device the client application will connect to. Because of the secure nature of Bluetooth connections, it is no possible to establish a connection without the user explicitly determining which device he or she wants to connect to.

Once the server device has been chosen, the client application shows on its screen a `CClientCanvas` object. This class has three purposes: to receive and identify keyboard or touchscreen events, to display the graphical user interface to the player, and to manage the communication channel for message exchanging between the client and server applications.

Lastly, the `CClientCanvas` object remains active on the device display during the entire session, unless the user decides to quit the application or the network link is lost.

4.2 The structure of the server application

The server application is organised in two modules. The first is the message server, whose main task is to manage all the client connections. The graphics manager is the second module, whose attribution is to draw all the game elements.

The message server is composed by two classes: `CServerBluetooth` and `CClientConnection`. The `CServerBluetooth` accepts connection requests sent by the clients within range. The message server also maintains a reference to a queue of messages, in which messages sent by all clients linked to the server are stored to be latter processed.

[2]A MIDlet is an application that uses the Mobile Information Device Profile (MIDP) of the Connected Limited Device Configuration (CLDC) for the Java ME environment.

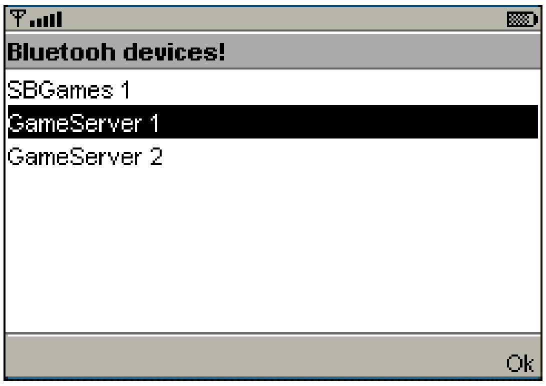

Bluetooh devices!
SBGames 1
GameServer 1
GameServer 2
Ok

Figure 5: Example of a device screen displaying a list of three devices found during the discovery stage.

For each new established connection, the `CServerBluetooth` object instantiates a `CClientConnection` object that, in turn, encapsulates a thread responsible for managing that connection.

The main attribution of a `CClientConnection` object is to constantly listen to the communication link, waiting for the arrival of new messages sent by the corresponding client. As stated before, all the messages received are inserted into the server's message queue. The `CClientConnection` may also forward messages from the graphics manager to any specific client. This is the case, for example, when the game need to update personal information (game status) on the screen of the player device.

4.3 The BlueWave communication protocol

We have decided for a simple communication protocol because of the following reasons: (1) the intended client devices are mobile phones, which are limited devices in terms of processing power and graphics capabilities; (2) we wanted a generic solution that would work with a great variety of game genres, eliminating the need to adapt the protocol for each new game application; and (3) we wanted to keep to a minimum the amount of information to be sent through the communication channel.

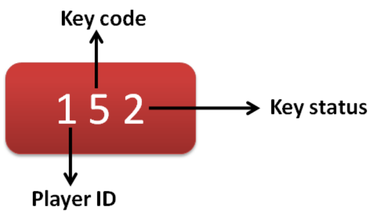
Therefore, our protocol has two types of message packages: the message sent by a client to the server, called `cli2ser` message type, which requires the setting of three data fields for each message package sent; and, the message sent by the server to a client, called `ser2cli` message type, which comprises two data fields. We shall describe them in turns next.

Every new client that connects to the main application receives a unique identification. The first data field of a `cli2ser` message type is always the client `ID`. The next data field is the code of a key (alphanumeric keyboard) or an user interface component (touch screen). The last data field is a status value denoting whether the key or component has been pressed or released (see Figure 6a). In this example, the message indicates that the player with `ID` = '1' (value of the first field) has sent a `keycode` = '5' with the `status` = '2', which means the key has been released.

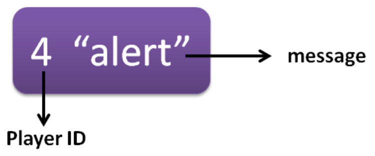
Here we shall consider a practical issue concerning the generation of `cli2ser` messages. Let us assume that the player has pressed and kept hold a key on his/her device. To avoid overloading the communication channel by repeatedly sending redundant messages telling that the same key remains pressed, we have defined a simple strategy: once a key on the client side has been pressed ('key pressed' message sent, or `keystatus` = '1'), the graphics manager will regard it as pressed until a corresponding 'key released' (i.e. `keystatus` = '2') message arrives. In other words, while a key is kept pressed, only a single message is sent from the client to the server.

The `ser2cli` type message has two fields (see Figure 6b). The first stores the destination address of a single client or a code indicating that this is a broadcast message that must be sent to all registered clients. The second field is a string that stores a generic information that needs to be sent to the receiver of the message.

In this case, the client is responsible for parsing the received string and use this information to do any necessary update to keep the application synchronised.



(a) The three data fields of a `cli2ser` message type. This example can be read as follows: the client '1' has released (indicated by the code = '2' in the last data field) the key with code '5'.



(b) The two data fields of a `ser2cli` message type. In this example, the server '4' has sent the string "alert" to the client.

Figure 6: Graphical representation of the two types of message packages.

The function of the graphics manager is to create and control the desktop game application or the server application. This module is organised into a set of classes, each of which responsible for activities such as time management, scene graph management, sound control e playback, and the drawing of special effects, layers and game characters. This architecture is based on a previous work on game engines [Malfatti et al. 2008].

The graphics manager organisation allows for the design of both singleplayer or multiplayer games. For the latter case, it is necessary to create the message queue that has to be passed as reference to the constructor method of the Bluetooth server. Notice that the message queue is needed so that the server application can store all the messages sent by the clients, remove each message, parse it, and take the suitable action, according to the message content.

In Figure 7 we present an overview of a generic application developed with the BlueWave framework. Notice in the image that there are two black arrows between the J2SE and J2ME based components, indicating that the clients are able to both send and receive information.

From the image in Figure 7, it is possible to observe that for an application to support multiple players it only needs to instantiate an object of the class `Server` and pass a reference to the message queue.

5 Case Studies

In this section we present four case studies, in which we have demonstrated some of the features of the BlueWave framework. All case studies involved a desktop based game remotely controlled by players with mobile phones.

The first game was inspired on the classic Strikers 1945, a vertical scrolling air plane shooter arcade game [Psikyo 1995]. As soon as a participant connects to the game server, she or he receives an air plane (with 3 lives) controlled by a set of keys on the mobile phone. All players fly on the same arena and may collaborate to destroy as many enemies as possible, although there is nothing to stop a player from attacking any of the other players. To win the game, a player must obtain the highest score among the participants.

Even though the game may be considered as originally collaborative, we observed that usually there was a lot of competition among players to find out who would obtain the highest score. It is worth noting that the information on remaining lives, number of points, medals, special guns, etc., are stored in and presented

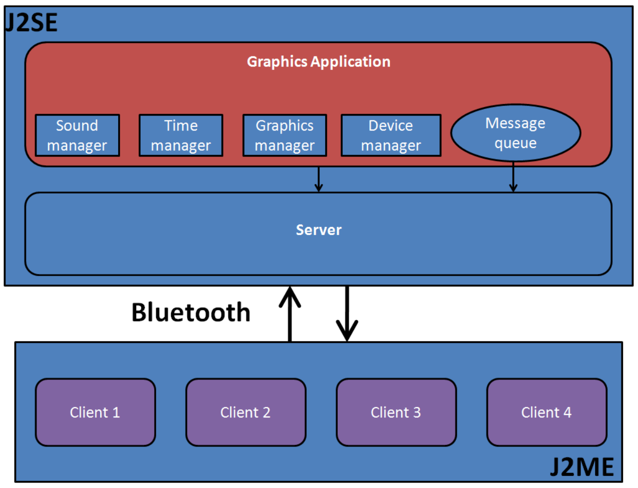


Figure 7: Overview of an application built on top of the Blue-Wave framework. Notice the two black arrows between the J2SE and J2ME components, indicating that the clients are able to send and receive information.

through the player's mobile phone. Thus, a player may hide her score, so that others would not know how she is doing. This fact has improved the social aspect of the game. Figure 8 shows a screenshot of the game and an example of the graphical user interface for two players.

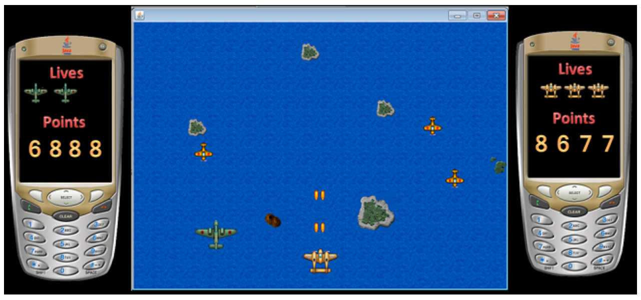


Figure 8: Screenshot of the Strikers 1945 clone, and the graphical user interface for each mobile phone connected to an instance of the game.

We also developed competitive games, as the prototype from Figure 1. This is a vertical scrolling car race game, in which the screen is divided in two lanes, one for each player. Each race car is controlled by the player's mobile phone. The left, right, up, and down arrow keys are mapped, respectively, to left shift, right shift, acceleration, and breaking of the car.

Win the game the player who reaches the finish line first. During the race, the players have to avoid several obstacles, such as other cars, trucks, and oil on the road. Each of these obstacles may slow down the competitor if they come into contact with the player's car.

The behaviour of the volunteers playing this game showed us a frequent situation: often one competitor would hit obstacles because she was distracted, observing the performance of the other competitor, running beside her. For us, this is evidence of the social aspect of the game in addition to a high level of fun afforded by presence games.

The two games described so far have in common the fact that each participant has to have his or her own mobile phone. The next game works differently. It is a quiz based game in which the players are assigned to groups and share a mobile phone (per group).

The serve application, written in J2SE, randomly chooses the questions and shows the green light for players to press a key indicating their wish to answer the question first. The fastest group to press the answer key has the right to provide an answer to the quiz. The chosen group then proceeds to select an answer out of a list of five alternatives. If the group gets it right they earn some points, otherwise their score drops.

The design of the quiz game is such that the set of quizzes and answers are read from input file. Hence, it is viable to employ this game as a supporting tool for instructive purposes in schools.

The last game prototype introduces a mix of collaborative and competitive aspects. It is a battle tank game in which each tank has to be controlled by a crew of two players. One player, the navigator, steers the tank, while the other, the gunner, controls the pivoting gun turret and actually fires the gun. The screenshot in Figure 9 shows that there are obstacles on the battle field that may block the shell. As a result, the tank crew has to work together navigate through the battlefield and fire at the opponent.

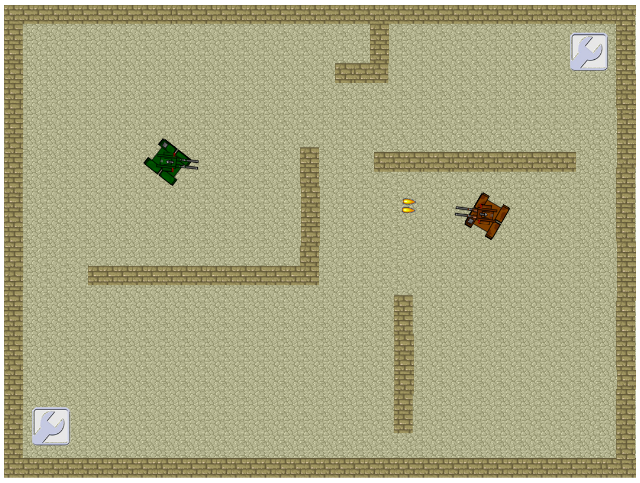


Figure 9: Screenshot of a battle tank game, in which each tank is controlled by a crew of two players. The two mobiles of a crew either steers the tank or controls the pivoting gun turret.

To make the game even more challenging and interesting, each member of the tank crew receives different information on their mobile phones, depending on their role (navigator or gunner). The navigator, for instance, have access to information regarding the tank current damage status and battlefield map. To avoid complete destruction of the tank, the navigator has to occasionally look for a tool icon on the battlefield. These icons improves the health for the tank.

The gunner, on the other hand, has to worry about the gun temperature, which tends to rise if the gunner fires repeatedly in a short time interval. If the gun reaches the threshold temperature, it stops shooting for a while, until the temperature drops to acceptable levels.

The main remark from this game was the social aspect. Even though each player had their own mobile phone, they had to constantly talk to each other to successfully command the tank.

6 Conclusion and future work

The ubiquitous nature of mobile phones affords a wide range of user interaction applications, specially in the game industry. To use a mobile phone as a control device, for instance, gives an opportunity for people who are fairly resistant to game consoles to try (and possibly enjoy) gaming with a tool they are accustomed to using in their daily lives.

In our view, the use of mobile phones as remote controllers in distributed desktop based games suggests the following advantages:

- *It allows for a more natural way of interaction.* Modern mobile phones are equipped with accelerometers, touch screen, Wi-Fi and Bluetooth communication. When these features are properly combined, mobile phones may be transformed into a powerful wireless gesture based controller.

- *It enables casual players to have access to high quality game even if they do not have a modern game console.* In this case, the casual player only needs a regular mobile phone with Bluetooth and Java support, and a desktop computer to run the game application.

- *It eliminates the need to buy extra proprietary game controllers.* Most modern video game consoles are sold with only one game controller. If a group of players want to play together they would have to buy all the extra controllers. Conversely, everyone having a Bluetooth enabled mobile phone would have the potential to use it as a game controller.

- *It promotes application interoperability among different manufacturers, platforms, processing power, and display capabilities.* The use of generic games controlled by mobile phones frees users from specific video game consoles, and widens the game's target audience, since most people have access to mobile phones nowadays. This is a valuable feature for the game market, which often tries to reach as many potential customers as possible.

Considering the aforementioned advantages, we set out to design, develop, and test a multiplayer game framework, named BlueWave. We focused on a distributed architecture in which mobile phones play a central role as wireless game controllers. At the beginning of our project we have defined a set of requirements to guide our design. These requirements were defined based on our analysis of related work concerned with novel forms of user interaction, rather than improved graphics quality.

To evaluate our proposal we have developed four distinct games built on top of the BlueWave framework. We tried to apply as many features as possible, so that most of the available functionality was tested. We then tested our games with several volunteers and gathered some observations to be analysed in terms of whether it has fulfilled the project requirements (c.f. Section 2).

The results were quite promising. In terms of accessibility and platform independence, the applications implemented with Java have granted access to a large variety of devices and mobile phones manufacturers. We have successfully tested our games with Linux, Windows, and Mac OS X. The client application has been also test on different mobile phones models and brands, such as Nokia, Samsung, Sony Ericsson, Motorola, and LG. With respect to the communicability, our battle tank game demonstrated that the client can work both as input device and as a mini display. The games we developed were scalable in the sense that they are multiplayer. We were able to run the battle tank game with up to 5 tanks, which means 10 simultaneous players distributed in two piconets.

Finally, the next step will be to support the on-board accelerometers of mobile phones. This would facilitate a more natural way to control 3D games, for example. We also intend to implement the support for the OBEX communication protocol, thereby enabling applications built with BlueWave to send and receive more complex information, such as a music or image files.

References

BERNARDES JR., J. L., NAKAMURA, R., AND TORI, R. 2009. Design and implementation of a flexible hand gesture command interface for games based on computer vision. In *Proceedings of the VIII Brazilian Symposium on Computer Games and Digital Entertainment (SBGames 2009)*, IEEE Computer Society, Los Alamitos, CA, USA, 64–73.

BURDEA, G. C., AND COIFFET, P. 2003. *Virtual Reality Technology*, 2nd ed. Wiley-IEEE Press, June.

BURKE, J. W., MCNEILL, M., CHARLES, D., MORROW, P., CROSBIE, J., AND MCDONOUGH, S. 2009. Serious games for upper limb rehabilitation following stroke. In *VS-GAMES '09: Proceedings of the 2009 Conference in Games and Virtual Worlds for Serious Applications*, IEEE Computer Society, Washington, DC, USA, 103–110.

CAM-TRAX TECH., 2010. CamSpace from cam-trax technologies. `http://camspace.com`. last access on August.

FIGUEIREDO, L. S., TEIXEIRA, J. M. X. N., CAVALCANTI, A. S., TEICHRIEB, V., AND KELNER, J. 2009. An open-source framework for air guitar games. In *Proceedings of the VIII Brazilian Symposium on Computer Games and Digital Entertainment (SBGames 2009)*, IEEE Computer Society, Los Alamitos, CA, USA, 74–82.

GOELZER, M., 2006. Bluecove: Comunicando aplicativos J2ME com J2SE através de Bluetooth. Revista Web Mobile, edição 8, ano 02, DevMedia Group.

GOLOMB, M. R., MCDONALD, B. C., WARDEN, S. J., YONKMAN, J., SAYKIN, S. J., SHIRLEY, B., HUBER, M., RABIN, B., ABDELBAKY, M., NWOSU, M. E., BARKAT-MASIH, M., AND BURDEA, G. C. 2010. In-home virtual reality videogame telerehabilitation in adolescents with hemiplegic cerebral palsy. *Archives of Physical Medicine and Rehabilitation 91*, 1 (January), 1–8.

HUANG, A. S., AND RUDOLPH, L. 2007. *Bluetooth Essentials for Programmers*, 1st ed. Cambridge University Press.

KAMMER, D., MCNUTT, G., AND SENESE, B. 2002. *Bluetooth Application Developers Guide: The Short Range Interconnect Solution*. Syngress Publishing, Rockland, MA, USA.

LAIKARI, A. 2009. Exergaming - gaming for health: A bridge between real world and virtual communities. In *Proceedings of the 13th IEEE International Symposium on Consumer Electronics (ISCE2009)*, IEEE Press, Kyoto, Japan, 665–668.

MACHADO, L. S. 2010. *Interação em realidade virtual e realidade aumentada*, vol. 1 of *Livros de minicursos do Brazilian Symposium on Virtual and Augmented Reality 2010 (SVR2010)*. SBC, Canal6 editora, May, ch. 2 – Dispositivos Não-Convencionais para Interação e Imersão em Realidade Virtual e Aumentada, 23–34.

MALFATTI, S. M., DOS SANTOS, S. R., FRAGA, L. M., OLIVEIRA, J. C., AND JUSTEL, C. 2008. The design of a graphics engine for the development of virtual reality applications. *Revista de Informática Teórica e Aplicada XV*, 3 (December), 26–45.

MICROSOFT, C., 2010. Introducing kinect for xbox 360. `http://www.xbox.com/en-US/kinect`, last visited, August.

MILIKICH, M., 2010. Jsr-000082 java^{TM}APIs for bluetooth. `http://www.jcp.org/aboutJava/communityprocess/final/jsr082/`, last access on August.

PSIKYO, 1995. Strikers 1945. `http://www.world-of-arcades.net/1945/Strikers%201945/Strikers1945.htm`, last access on August 2010.

SHERMAN, W. R., AND CRAIG, A. B. 2002. *Understanding Virtual Reality: Interface, Application, and Design*, 1st ed. Morgan Kaufmann, September.

STENGER, B., WOODLEY, T., AND CIPOLLA, R. 2010. *Computer Vision Detection, Recognition and Reconstruction*, vol. 285 of *Studies in Computational Intelligence*. Springer-Verlag, Berlin, Germany, ch. 6 – A Vision-Based Remote Control, 233–262.

TEICHRIEB, V., AND FIGUEIREDO, L. S. 2010. *Interação em realidade virtual e realidade aumentada*, vol. 1 of *Livros de minicursos do Brazilian Symposium on Virtual and Augmented Reality 2010 (SVR2010)*. SBC, Canal6 editora, May, ch. 1 – Interação Natural, 9–22.

THOMPSON, T. J., KUMAR, C. B., AND KLINE, P. J. 2008. *Bluetooth Application Programming with the Java APIs*, essentials ed. The Morgan Kaufmann Series in Networking. Morgan Kaufmann.

VAJK, T., COULTON, P., BAMFORD, W., AND EDWARDS, R. 2008. Using a mobile phone as a “wii-like” controller for playing games on a large public display. *International Journal of Computer Games Technology 2008*, 1–6.

WINGRAVE, C. A., WILLIAMSON, B., VARCHOLIK, P. D., ROSE, J., MILLER, A., CHARBONNEAU, E., BOTT, J., AND JR., J. J. L. 2010. The wiimote and beyond: Spatially convenient devices for 3D user interfaces. *IEEE Computer Graphics and Applications 30*, 71–85.

ZMOGINSKI, F., 2010. Brasil fica perto de 1 celular por pessoa. INFO Online, `http://info.abril.com.br/noticias/mercado/brasil-tem-175-mi-celulares-vivo-lidera-22022010-29.shl`, February, 22nd. last access on August.