Test-Bed for Multimodal Games on Mobile Devices

Marcello Coiana[1], Alex Conconi[2], Laurence Nigay[3], and Michael Ortega[3]

[1] Arcadia Design,
Loc. Is Coras – CP. 360 0928 Sestu, Italy
marcello.coiana@arcadiadesign.it
[2] TXT e-solution,
Via Frigia 27 20126 Milano, Italy
alex.conconi@txt.it
[3] Grenoble Informatics Laboratory (LIG), University of Grenoble 1,
38042 Grenoble, France
{laurence.nigay,michael.ortega}@imag.fr

Abstract. We present a test-bed platform for the iterative design of multimodal games on a mobile phone or a PDA. While our test-bed platform is general to multimodal systems, in this paper we focus on games on mobile devices, since games are intrinsically multimodal, enriching thus the gaming experience and potential for interaction innovations. As part of a player-centered design approach, our goal is to be able to quickly test various forms of multimodal interaction before beginning with the development of the final multimodal game on mobile devices. We adopt a micro-approach for our game test-bed, focusing on multimodal interaction for generic tasks, including moving of an object or a character. Our test-bed facilitates the efficient exploration of the multimodal design space and enables controlled experiments of the designed interaction in-lab and/or in-situ.

Keywords: Game, Multimodality, Mobile Device, Test-Bed Platform.

1 Introduction

The first commercialization of many user interface innovations appeared in games, even though they were invented in research environments [7]. Nowadays, these innovative modalities and multimodal interaction techniques, are evident in current high-end platform games (e.g. Nintendo Wii). While in such platform games, innovative modalities and multimodality are becoming more and more popular, multimodality in mobile games is still in its infancy. In the few studies focusing on multimodal mobile games, the considered modalities are mainly reduced to speech and graphics [16] as well as location (i.e., a passive/perceptual modality) as in the pervasive mobile game ARQuake [13]. Moreover except for a few mobile phones that include sensors such as accelerometers, proximity sensors and tactile screen with two-handed interaction as in the recent iPhone [10], multimodality on mobile devices is far from being mature.

P. Markopoulos et al. (Eds.): Fun and Games 2008, LNCS 5294, pp. 75–87, 2008.

However, as the mobile game market tends to follow the more mature console/PC game market and with the recent growing diffusion of multimodal phones, game developers will be more and more interested in following this new trend. Moreover the players will become more demanding in terms of involvement in the mobile games as they are used to interacting with different modalities in PC/console games. Game performance must match or exceed the users' expectations. As a consequence, in order to anticipate the users requirements, and in order to be ready for the market challenge, the game developers need a prototyping platform to experiment on new modalities and forms of multimodality inside their mobile games. In this paper we address this issue by presenting a game test-bed that allows the rapid development of prototypes as part of an iterative user-centered design. Our game test-bed enables the game designers to experiment on new pure or combined modalities inside their games on mobile devices, while overcoming the current interactive limitations of mobile devices. Therefore the test-bed enables the game designers/developers to focus on the experience of the player before developing the final game running on mobile devices. When studying game designer practices, we learned that the experience is key. Since many modalities are available on PC, our test-bed for multimodal mobile games establishes a bridge between two platforms - the PC and the mobile device. The design, development and evaluation of our test-bed have been conducted as part of a multidisciplinary European project, OpenInterface [12] that involves game designers, HCI designers, software developers as well as psychologists and ergonomics.

The structure of the paper is as follows: We first explain the adopted iterative user-centered design approach and the key role of the test-bed in order to fully understand the scope of our mobile game test-bed. We then provide an overview of the underlying software framework used to develop the test-bed, namely OpenInterface, and then present the mobile game test-bed. We finally present an illustrative example of the test-bed, the Funny rabbit game.

2 Player-Centered Design Approach and Test-Bed

Following the ISO standard human-centered design process for interactive systems (ISO 13407 Model, 1999), we adopt an iterative user-centered design approach for designing mobile games. Such an approach for the design of games involves many different types of expertise. Two of the most common groups involved in user interface design are human factors specialists and programmers [1]. For games, graphic and game designers are also involved. The user-centered design approach aims at bringing all these types of expertise together. The four main activities of the design approach, namely 1- Specify the context of use 2- Specify the user requirements 3- Produce design solutions 4- Evaluate designs against requirements, are iterative. For games, the process is truly iterative as shown in Figure 1, and the need for intensive playtest is crucial. The game is evolving from an initial starting point as the designer is designing and balancing the game play of the title [6]. We address this issue of "evaluating designs against requirements" by providing a test-bed for playtesting mobile games. Our test-bed allows the rapid prototyping of the mobile games for performing test with players. Such an approach is also called a player-centered design approach [8] or experience-centered design approach.

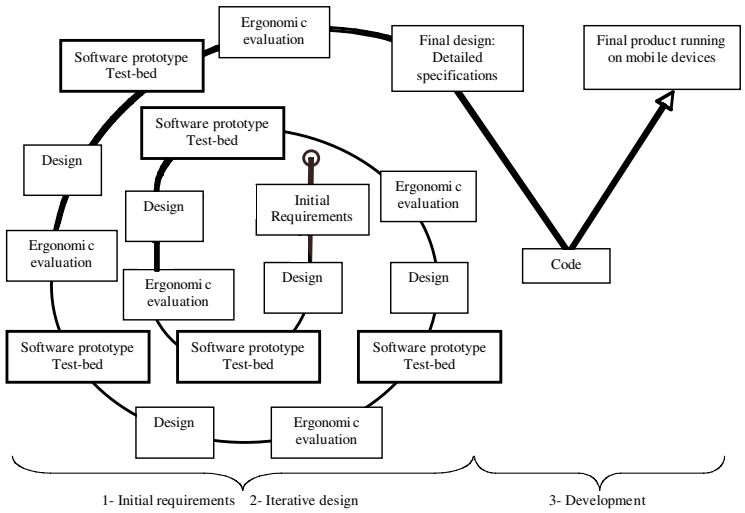


Fig. 1. Overall design and development process. The game test-bed supports the iterative user-centered design. The process comprises three main phases: 1-Initial Requirements 2-Design phase based on prototypes that are iteratively evaluated, modified and refined to lead to detailed specifications. 3- Development based on a more classical approach from software engineering as in the IntuiSign design and development process [14].

Although there is no common agreement on what kind of usability evaluation methods can and should be used to enhance the design of games, player experience will be particularly influenced by input and output modalities and interaction techniques [2]. Multimodality plays a central role in games since the only incentive to use a game is the pleasure derived from the interaction. In addition the flexibility of multimodal interaction enables us to support various player types that also enhance playability [8]. Moreover it has been shown that multimodality in games also enhances the interaction robustness by compensating the weakness of one modality by the strengths of other modalities, making the mobile game more enjoyable in any given environment [11].

To be able to experience a rich set of innovative pure or combined modalities, the game test-bed enables the rapid development of multimodal prototypes. While the test-bed enables us to explore multiple design solutions in terms of multimodal interaction and to experience them with players manipulating a mobile device, the final version of the game will then be developed on mobile devices: this is modeled in Figure 1 by the right branch. The resulting overall design and development process corresponds to the structure of the IntuiSign design and development process [14], an attempt to integrate the well-know user-centered iterative design in a process that fits within the industry constraints.

3 OpenInterface Framework for Multimodal Interaction

For developing the mobile game test-bed, we base ourselves on the OpenInterface framework, an environment for the rapid development of multimodal interactive

systems. The OpenInterface (OI) framework is made of an underlying platform, namely OpenInterface platform, and the OpenInterface Interaction Development Environment (OIDE).

- The OI platform [3] is a component-based platform that handles distributed heterogeneous components based on different technologies (Java, C++, Matlab, Python, .NET). It allows the integration of existing interaction techniques written in different languages. The platform includes a tool for creating new components (i.e., interaction techniques) from existing code without modifying the original code.
- The OIDE is a component-based system built on top of the OI platform. OIDE adds development tools offering access to interaction capabilities at multiple levels of abstraction. The OIDE includes a component repository as well as construction, debugging, and logging tools.

3.1 Component Repository

The OIDE component repository includes modalities both developed ”in-house” (e.g., speech recognition, video “finger tracker”, accelerometer-based gesture recognition) and also accessible via proxies to other component sets (e.g., Phidgets, ARToolkit). Our components include device drivers, interaction techniques, multimodal fusion mechanisms, development services and developer-defined combinations. OIDE supports descriptions, querying and access in an extensible set of description schemas. Within the vast world of possibilities for input modalities (from the user to the system) as well as for outputs (from the system to the user), we distinguish two types of modalities: the active and passive modalities [5]. For inputs, active modalities are used by the user to issue a command to the computer (e.g., a voice command or a gesture recognized by a camera). Passive modalities refer to information that is not explicitly expressed by the user, but automatically captured for enhancing the execution of a task. For example, in the “Put that there” seminal multimodal demonstrator of R. Bolt [4], that combines speech and gesture, eye tracking was used for disambiguating among multiple objects on screen. For games, examples include location-aware games and more generally pervasive mobile games [13]. A huge variety of passive modalities can be used in games including emotion, bio-signal processing and eye-movement.

3.2 Construction and Logging Tools

Figure 2 illustrates the OIDE Graphical Editor in which components accessible via the repository can be inspected, configured, linked to other components and to external services or to application functionality, in the assembly view. The result can be either an individual technique or a fully functional user interface to an application. In the example of Figure 2, a Speech recognition component and a Wiimote component are both connected to a fusion component. The latter combines the received events in a complementary way in order to obtain a complete task “Go there”, combining speech and gesture with the Wiimote. The graphically specified pipe-line of OpenInterface components therefore defines a generic multimodal technique that can be then be connected to an application.

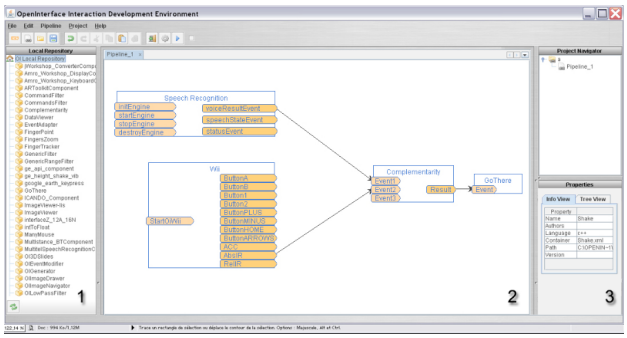


Fig. 2. Overview of the OIDE graphical editor: 1-Palette of components (modalities, devices, fusion and transformation components). 2- Editing zone for assembling the components and defining an OI component pipeline 3- A component information panel.

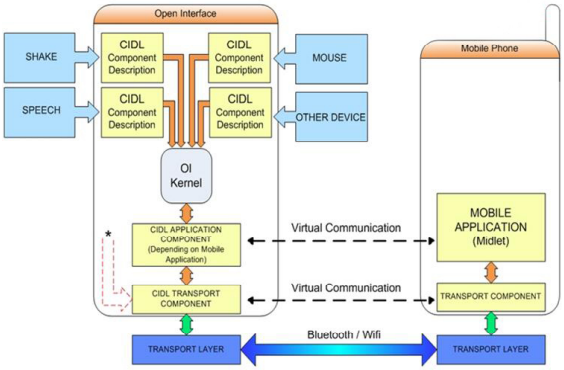
Fig. 3. OIDE Debugging and Logging tools. In the example, the Shake accelerometer events are logged in order to tune the interaction with the mobile game.

The logging tool is responsible for recording multiple component-specific formatted data streams while an OI application is executing [9]. A generic "oscilloscope" display, that can be configured while in use, can be used to display data streams at runtime. In Figure 3, the oscilloscope shows data at the lowest level of abstraction, direct from the device itself (in the example, the SHAKE device) but it is also possible to display application-level data (as shown in Figure 7).

4 Game Test-Bed on Mobile Devices

Focusing on multimodal interaction, the game test-bed is based on the OpenInterface (OI) framework described in the previous section. Therefore the test-bed demands for an architecture that links the OI platform running on a PC with the functional core of mobile games running on devices such as PDAs or Smartphones. Test-bed applications are expected to be flexible and open, to allow fast prototyping for experimentation, but also consistent and adequately performing, in order to address actual use cases. This tradeoff requires careful architectural choices. Besides this main functional requirement (i.e., support the communication between the OI platform and game functional core), some non-functional requirements apply, namely technical constraints (e.g., compatibility with the OI framework and with legacy functional core technology) and business constraints (e.g., support most widespread mobile phones). Furthermore we identify and address quality attributes including usability (provide an easy to use RAD framework for developers of the game functional core and interaction), modifiability (take into account that test-bed scenarios may involve unpredictability) and performance (avoid user interface sluggishness and provide stable and acceptable response times).

Our architecture is based on a layered protocol stack approach, as shown in Figure 4. We designed two additional components running inside the OI platform (also called OI



* Any other component can use the Transport Component to communicate with remote devices using the protocol described in the transport layer.

Fig. 4. Test-bed architecture connecting the OpenInterface platform running on a PC with a game application running on a mobile device

runtime kernel above): the Application Component and the Transport Component. The Application Component acts as a proxy for the mobile application. It is connected with other OI components through the OI platform pipe. Whenever an event from any connected component occurs, the Application Component parses such an event and sends a corresponding message to the mobile application through the Transport Component. The CIDL (i.e. the OpenInterface XML description) of the Application Component provides sinks and sources that manage the communication with the other OI components handling the input and output multimodal interaction. For example Figure 4 presents a setup that includes the SHAKE device, a Speech Recognition engine and a traditional mouse, all plugged into the OI platform. Such devices and modalities have been selected and assembled using the OIDE editor of Figure 2. An additional sink/source pair in the Application Component is connected to the Transport Component.

The Transport Component defines an interface for transferring messages from the OI platform to the mobile game and vice versa. Specific implementations of the Transport Component adopt different physical transport layers, such as Bluetooth or TCP/IP over WiFi. It is worth noting that the abstract transport layer interface also allows non physical connection, when the OI platform and the test-bed functional core are running on the same machine.

The communication protocol between the Application Component and the Mobile Application is message based. Messages are built by the Application Component in accordance with a protocol that the mobile application understands. When designing the message protocol, we identified three main approaches based on the level of abstraction of the exchanged messages:

- Low level: Send raw input from devices and modalities to the mobile application;
- High level: Send application-dependent events to the mobile application. This requires abstracting modality-dependent events in order to obtain application-dependent events (e.g. “Quit Game”, “Move Player Up”, “Open Inventory”);

- Mid level: Send modality-independent and application-independent events to the mobile application. This solution implies abstracting the modality-dependent events into generic meta-events (e.g. “ESC”, “UP”, “START”) creating hence an enhanced virtual gamepad.

On the one hand, the low level approach has the downside of deporting to the mobile application the computational charge of processing the (potentially heavy) raw data flow. On the other hand, the high level approach requires no processing. Representing however each event for every application makes the Application Component become too complex. It also implies a strong interdependence between the Application Component and the mobile application which goes against the modifiabity quality attribute. The mid level approach appears to be the optimal tradeoff: It performs the data flow processing inside the OI platform on the PC, but the internal logic of the OI component pipeline is not too tightly bound to the mobile application functionalities. Nevertheless a mixed approach combining the three above approaches is possible since the OI platform pipe allows stacking events at different levels of abstraction. These levels of abstraction are defined by the power of abstraction (multimodal inputs) or concretization (multimodal outputs) of the function performed by the OI component pipeline. The notion of level of abstraction expresses the degree of transformation that the interpretation and rendering functions realized by the OI component pipeline perform on information.

5 Illustrative Example: Funny Rabbit

From a game designer’s point of view, the development of a game begins with a brilliant idea and ends with the definition of the smallest detail of interaction to increase the player’s experience and engagement in the game. Our test-bed is dedicated to the design of the interaction, after sketches and storyboards that will be used in the beginning of the development.

For designing the interaction, we apply a hierarchical task analysis, compatible with the approach of the game tools that organize a game in missions, objectives, goals, triggers and so on. The game can be described in terms of interactive tasks and subtasks, breaking down the steps of a task performed by a player. Tasks and subtasks describe goals and actions. For example a task can be a main goal and its subtasks the actions needed to reach this goal:

$$Game = \{task1, \ldots taskn\} \quad Taski = \{subtask1, \ldots subtaskn\}$$

For performing each elementary subtask of a given task (abstract task), interaction modalities must be selected that leads us to define concrete tasks. For example an abstract task is <Move a character to the left> and a concrete task <Press left direction key>. In a standard mobile game, each task is performed by the player in a standard way with only one modality namely mod1(Keyboard): *task1->mod1 … taskn->mod1*.

In this way, the design process is more focused on the abstract tasks rather than the concrete tasks. This happens because the way the tasks are performed by the player (concrete tasks) is given for granted for a game designer: for instance the movement of an object or a character is specified using the direction keys or the joypad. On the contrary, with our game test-bed, the design process focuses on both the abstract and

concrete tasks. For an elementary abstract task, several pure or combined modalities can be chosen for defining the corresponding concrete task:

task1 ->mod1 ... task1->modn

Our test-bed allows the designer to explore different modalities for a given task. For performing each elementary task (abstract task), several pure or combined modalities can be designed (concrete task). To do so, the player's experience is always at the centre of the whole design process as shown in Figure 1. Questions raised during the iterative design process include: Is the player satisfied? Is the player engaged in the game? Is it fun? Is it cool? These issues are addressed for each task during the design. Concrete tasks are not given for granted anymore, leading to new unexpected experiences for the player and opening new doors to the creativeness of the game designer. We illustrate this design process and the use of the test-bed in the context of our Funny rabbit game of Figure 5. We show how a task can be extracted from a standard game on mobile devices to study different forms of multimodality for performing it.

In terms of the overall architecture of the test-bed, the functional core (logic of the game) of Funny rabbit is running on the mobile phone. Moreover the graphical outputs are also managed outside the OpenInterface framework, the corresponding code running on the mobile phone. To consider a game as close as possible to the state of the art in games, Funny rabbit is a 3D rendered graphic game. The player controls the movement of a 3D character, chooses which object to interact with, and examines 3D objects with 2 degrees of freedom. These are commonest tasks of a game. Thus, even though we illustrate the test-bed by considering the Funny rabbit game, the designed concrete tasks can be used for another game since they are concrete but generic.

The main task or goal of Funny rabbit is <find the rabbit that is hidden in a chest>. Such high level task is decomposed into three subtasks, illustrated in Figure 6: (1) move (the character), (2) select (the chest), (3) examine (the objects inside the chest).

For the design, we need to proceed further in the hierarchical task analysis and refine the three above tasks in terms of elementary abstract tasks. For example the task <move> is divided into four subtasks: Up, Down, Left and Right. Having identified elementary abstract tasks, we now focus on concrete tasks by considering the modalities to perform such tasks. We start the design of the concrete tasks by considering a first set of interaction devices. For Funny rabbit, we have so far

Fig. 5. The Funny rabbit game

Fig. 6. The three tasks Move, Select and Examine

Fig. 7. Funny Rabbit: Movement of the player with the webcam

Fig. 8. (a) The webcam with Nokia N93 for moving the 3D character. (b) The SHAKE with Nokia N95 for 3D object examination.

considered the following input devices: a microphone, a SHAKE and a webcam. All of these devices are external to the mobile device (Nokia N93). Based on these three devices, different interaction modalities can be designed. We consider speech commands, 3D movements with the SHAKE and 3D movements recognized by the camera coupled with ARToolkit, as shown in Figure 7. In Funny rabbit, an example of use of these three modalities is the webcam with ARToolkit to move the 3D character (Figure 8-a), speech to issue commands like SELECT, and the accelerometer sensor of the SHAKE for 3D object examination (Figure 8-b).

The three concrete tasks have been specified using the OIDE graphical editor of Figure 3. The resulting OI component pipeline is shown in Figure 9: The OI components "SpeechRecognition" and "Shake" correspond to two modalities, respectively speech command and 3D gesture. Two OI components implement the 3D movement using the webcam. One component "ARToolkit" is generic and corresponds to the ARToolkit functionalities (proxy to access the component sets of the ARToolkit). The second component "Adapter" transforms the ARToolkit events in terms of control commands. Since we reuse a component that was developed to control an helicopter in another game, its port is called HeliControl by the developer of the component. For this modality, we did not develop a specific component but reuse existing ones in the palette of components. The OI component "MultiBTComponent" corresponds to the Transport component of Figure 4 and implements a Bluetooth transport layer. The OI component "EventAdapter" plays the role of the Application component of Figure 4.

The pure or combined modalities that define the concrete tasks can be easily adapted by graphically modifying the OI component pipeline of Figure 9. Indeed several other design possibilities for the concrete tasks could be specified and tested for Funny rabbit. The design solution of Figure 9 is based on continuous input for moving the character: this implies that for every single step of the character an input event is required; thus for keeping the character moving up, the player needs to keep

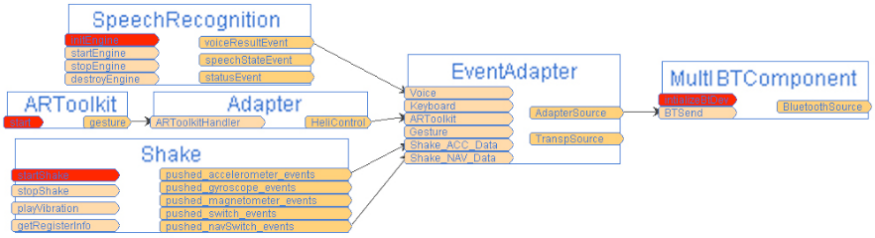


Fig. 9. OI component pipeline that defines the three concrete tasks of Figure 6

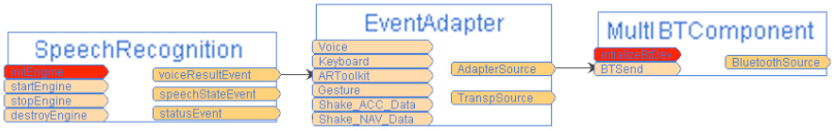


Fig. 10. OI component pipeline that defines the selection of an object by using speech

the webcam in the up position, like a virtual airplane cloche. Another design possibility to explore is discrete input implying to manage different movement states of the character: one event will then modify the state of the character. For instance an event “up” will switch the state into -following the north direction- until the player changes the state by a new event, for instance an event “stop”. In this example, we change the elementary tasks of Funny rabbit and not the main three tasks of Figure 6. While modifications of the three main tasks will require changing the game (the functional core of the game) running on the mobile phone, modifications of the elementary tasks will be done within the OI framework. For example, speech commands are assigned to change the movement states of the character and the navigation switch of the SHAKE is assigned to stop any movement of the character. Since the modality based on the webcam is then no more used for moving the character, it can be assigned to examine 3D objects. Such design solution is specified by graphically assembling OI components and can be then tested with players.

The level of abstraction of the OI components manipulated in the OIDE graphical editor allows the user to focus on the interaction modalities and their combination. For example, the game designer wants to specify the way to select an object from a list: s/he starts with a voice command. The corresponding OI component pipeline specified with the help of a software developer is shown in Figure 10. Coming back home, s/he remembers that s/he has to pay the bills, so s/he opens the drawer where s/he kept them. While s/he is doing this movement, s/he likes the idea of selection associated to that action. So, the following day s/he wants to do the same with the selection task in the game, and adds a 3D gesture recognition engine that recognizes the opening gesture, using the OIDE graphical editor. To do so, in addition to the ready to use OI component for 3D capture (“Shake” component), a new OI component must be developed by programmers for recognizing the opening gesture (“Opening Gesture” component). A generic OI component for gesture recognition, based on an editor to specify by examples the gestures to be recognized, is under construction. Then the game designer enriches the pipeline with the new component

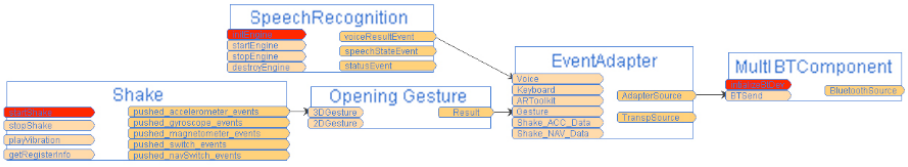


Fig. 11. OI component pipeline (extension of the one of Figure 10) that defines the selection of an object either by using speech or by performing an opening gesture using the SHAKE

as shown in Figure 11. Since the designer likes this way of interacting, s/he also asks the graphical department to make a graphical interface that looks like a drawer. The new prototype will then be evaluated with the help of human factors specialists. Another design solution based on combining the two modalities could also make the interaction more challenging, by combining the opening gesture with a voice command (a magic word for example). Such a design solution simply implies adding a fusion component (Figure 2) before the EventAdapter component in Figure 11.

This simple scenario illustrates the iterative player-centred design of the multimodal interaction of the game and highlights the multidisciplinary character of our design approach as well as the use of our test-bed for quickly developing different prototypes.

6 Conclusion and Future Work

Focusing on multimodal interaction for mobile games, we have presented a test-bed that allows the rapid development of prototypes as a central tool for an iterative player-centered design approach. The key features of our test-bed include:

- The rapid development of design solutions for multimodal interaction, including the combination of existing modalities on mobile devices such as the direction keys of the mobile device keyboard with new innovative modalities,
- The extension of existing mobile games with innovative pure or combined modalities that may not be available today on mobile phones,
- The support for an iterative player-centered design approach involving multiple expertise.

The first informal in-lab evaluation of the Funny rabbit game with a group of 16 users confirms that it is crucial to let players manipulate and experience the game. The evaluated test-bed includes many equivalent modalities for the abstract tasks such as moving the 3D character. For example first feedback was “it is fun” “it is cool”. We plan to perform further experiments based on a game scenario including in-field evaluation, carrying the computer in a backpack.

There are several directions for future work on the test-bed for mobile games. We need to further study the design trade-off in the test-bed that leads to us having some input/output modalities managed in the OpenInterface framework and some running directly on the mobile phone with no link with the framework. For example in the Funny Rabbit game, the 3D graphical display is managed outside the OpenInterface

framework, running on the mobile phone. This design trade-off has direct impact on the interaction design possibilities since modalities not managed within the framework cannot be combined with other modalities for exploring complementary or redundant usage of modalities. In addition this design trade-off is also based on interaction performance, minimizing the exchanged messages between the framework and the mobile application. This is especially important for the case of continuous feedback (tightly coupled interaction as in Shoogle [15]): indeed the mobile application must send messages to the OpenInterface framework for rendering information. That is why we did not manage the graphical display of the game in the framework. Nevertheless other continuous feedback can be managed in the framework. For example let us consider the case where the player moves a character on screen by tilting the mobile phone while obtaining vibrotactile feedback on the movement speed. In order to produce timely haptic responses, the mobile application must send messages to the OI framework and in particular to the OI component dedicated to the vibrotactile display for presenting the current computed speed.

Finally, in order to show the path from research results to industry requirements, a further step ahead needs to be done: the development of a final product from a prototype, as highlighted in our design and development process of Figure 1. The high level of abstraction of the multimodal interaction that the OpenInterface framework offers to developers, allows a smooth transition from a prototype to a well-engineered system that can satisfy the constraints of the current technology in terms of mobile devices and network performances. To show this smooth transition, we developed a game running on commercially available mobile devices, namely a game validator, that is a sub-part of our prototypes validated using our test-bed.

Acknowledgments. This work is funded by the OpenInterface (OI) European FP6 STREP FP6-035182. The authors thank the contribution of their OI colleagues.

References

1. Bailey, G.: Iterative methodology and designer training in human-computer interface design. In: Proc. of CHI 1995, pp. 198–205. ACM Press, New York (1995)
2. Bernhaupt, R., Eckschlager, M., Tscheligi, M.: Methods for Evaluating Games – How to Measure Usability and User Experience in Games? In: Proc. of ACE 2007, pp. 309–310. ACM Press, New York (2007)
3. Benoit, A., et al.: Multimodal Signal Processing and Interaction for a Driving Simulation: Component-based Architecture. Journal on Multimodal User Interfaces 1(1), 49–58 (2006)
4. Bolt, R.A.: Put-that-there: Voice and gesture at the graphics interface. In: Proc. of SIGGRAPH 1980, vol. 14(3), pp. 262–270 (1980)
5. Bouchet, J., Nigay, L.: ICARE: A Component-Based Approach for the Design and Development of Multimodal Interfaces. In: Extended Abstracts of CHI 2004, pp. 1325–1328. ACM Press, New York (2004)
6. Cherny, L., Clanton, C., Ostrom, E.: Entertainment is a human factor: a CHI 1997 workshop on game design and HCI. ACM SIGCHI Bulletin 29(4), 50–54 (1997)
7. Crawford, C.: Lessons from Computer Game Design. In: Laurel, B. (ed.) The Art of human-computer interface design, pp. 103–111. Addison-Wesley Pub. Co., Reading (1990)

8. Ermi, L., Mäyrä, F.: Player-Centred Game Design: Experiences in Using Scenario Study to Inform Mobile Game Design. Game Studies 5(1) (2005), http://gamestudies.org
9. Gray, P., Ramsay, A., Serrano, M.: A Demonstration of the OpenInterface Interaction Development Environment. In: UIST 2007 demonstration (2007)
10. iPhone, http://www.apple.com/iphone
11. Kue-Bum, L., Jung-Hyun, K., Kwang-Seok, H.: An Implementation of Multi-Modal Game Interface Based on PDAs. In: Proc. of ACIS SERA 2007, pp. 759–768. IEEE, Los Alamitos (2007)
12. OpenInterface European project. IST Framework 6 STREP funded by the European Commission (FP6-35182), http://www.oi-project.org
13. Piekarski, W., Thomas, B.: ARQuake: The Outdoor Augmented Reality Gaming System. In: Communications of the ACM, vol. 45(1), pp. 36–38. ACM Press, New York (2002)
14. Schlienger, C., Chatty, S.: An Iterative and Participatory HCI Design Process in the Industry Context: Bringing together Utility, Usability and Innovation... within Budget. In: Interfaces Magazine, vol. 72. HCI Publications (2007)
15. Williamson, J., Murray-Smith, R., Hughes, S.S.: Multimodal Excitatory Interaction on Mobile Devices. In: Proc. of CHI 2007, pp. 121–124. ACM Press, New York (2007)
16. Zyda, M., et al.: Educating the Next Generation of Mobile Game Developers. IEEE Computer Graphics and Applications 27(2), 96, 92–95 (2007)