Multimedia Tools and Applications (2019) 78:32261–32305
https://doi.org/10.1007/s11042-019-07983-6

# Developing pervasive games in interactive spaces: the JUGUEMOS toolkit

Clara Bonillo[1] · Javier Marco[2] · Eva Cerezo[3]

Received: 19 October 2018 / Revised: 11 May 2019 / Accepted: 10 July 2019
Published online: 31 July 2019


**Abstract**
The progress in the development of pervasive games is slowing down because of the multiple challenges that these games brings to developers, due to the great variety of interaction paradigms that this kind of games involve and the difficulties of developing applications where so many innovative technologies converge. In this article we present JUGUEMOS, a toolkit aimed at developers to help them in the creation of pervasive games for Interactive Spaces. The toolkit addresses three challenges that arise when developing pervasive games: the integration of heterogeneous devices, the management of multiple displays and the facilitation of the game coding. The toolkit is based on the TUIML modeling language that allows defining games easily, reducing the impact of the coding between iterations. The toolkit also makes use of the OSC Protocol to interconnect the different devices. Detailed descriptions of the toolkit design decisions, architecture and implementation are presented, together with three different case studies carried out in order to explore the toolkit expressivity, its capability to support collaborative multidisciplinary experiences, and its potential to support interactive experiences outside our Interactive Space. We hope this work would contribute to the spread of pervasive games in interactive spaces.

**Keywords** Pervasive games · Interactive spaces · Toolkits · TUIML

## 1 Introduction

Computer games have some advantages that have make them more popular than traditional games. They intrinsically motivate players by bringing them more fantasy, challenge and

✉ Clara Bonillo
clarabf@unizar.es

Javier Marco
jmarco@esda.es

Eva Cerezo
ecerezo@unizar.es

Extended author information available on the last page of the article

curiosity, which area three main elements contributing the fun in games [25]. However, computer games have often decreased physical activity and social interactions. In the last years, real world is coming back to computer entertainment, with a new gaming genre: Pervasive games. Pervasive games are no longer confined to the virtual domain but integrate physical and social aspects of real world [22]. Usually, the idea in Pervasive Games is that they overcome the limitations of traditional games in the spatial, temporal or social dimensions [28]. However, the term has several definitions that set the focus in different issues related to the games. Last years, the focus has been put in the context, and they are defined as a new gaming experience where the dynamics of the game evolutions by means of the information provided by the context where it is played [1]. Nevertheless, and even though the ambiguity of the term and its use to refer to very different types of games, there are three common characteristics shared by all of them: (1) the mixing of real and virtual world, (2) the support of natural styles of interactions (voice, tangibles, gestures), and (3) their strong social component.

Moreover, a characteristic of this kind of games is that the technologies involved on them have to be seamlessly integrated on the game environment, which is augmented to enable players to interact with the game by physical manipulation of conventional toys, to provide them with useful information via displays or to immerse them in the game world. However, this process of augmentation is not an easy task, since it requires the involved technology to be as "hidden" as possible for the user, in order to not interfere with the game process and to keep the original characteristics of traditional games [48].

Interactive Spaces [14] are a natural place to deploy Pervasive Games, as they both share their Ubiquitous Computing nature. Interactive Spaces (IS) are Distributed User Interfaces supporting several ways of interactions in digitally augmented rooms that combine:

- Multiple interaction techniques: a panoply of related interaction paradigms such as Physical Computing, Context-Aware Computing, Mixed Reality, Wearables and Tangible User Interfaces can be combined and converge in an IS, allowing multiple users to interact, at the same time or in a distributed way, with the different devices involved in the IS.
- An heterogeneous multidisplay output ecosystem: in an IS, providing feedback to let users know what is happening in the environment can be done by different ways: projection walls, screens, and mobiles that can show visual and audio information to the users, haptic technologies that recreate the sense of touch by applying physical reactions to the users such as vibrations or movements, systems that are able to change the users' taste perception, or even based olfactory devices that display different smells to the users depending on the goal of the application displayed in the IS [49].

Initially, ISs have been applied to explore new possibilities of collaborative work and meeting rooms [3, 21] but more recently they have being considered as the ideal environment for the creation of games [11]. As we will see in the state of the art section, during the last years we are seeing an increasing number of projects and prototypes of pervasive games in ISs. Existing systems have greatly contributed to the pervasive games research by emphasizing different natures of ubiquitous interaction. The key to bridge the gap between the design and development of pervasive games is to be able to quickly prototype and tweak game rules as well as the involved user interfaces and interaction devices [44]. However, the number and heterogeneity of display devices that can be found in an IS are raising the multiples challenges that the design

and prototyping of pervasive games involve. Besides, the novelty of the ubiquitous technologies suppose an added complexity to the design and development process, since they have begun to be used just recently and they still lack of stability and homogeneity. Therefore, there is a need of frameworks or toolkits that facilitate, on the one hand, the communication with so many varied input devices and that, on the other hand, put the focus on the prototyping of pervasive games for ISs. Also, as stated by Myers et al. [33], frameworks are tools that aim to make the rapid prototyping of user interfaces easier and enable more iterations during the design process by lowering the threshold of developing the system while retaining a high ceiling of expressiveness. The threshold is the difficulty of learning and using a system, which is connected with the development process, while the ceiling captures the complexity of what can be built using the system, which is directly related with the design process.

The current article presents JUGUEMOS, a toolkit for the prototyping of pervasive games in Interactive Spaces. The toolkit has been designed as a set of tools that help developers with the implementation of hybrid games in indoor ISs, lowering this way the threshold of implementing pervasive games. The toolkit makes use of well-known abstraction modeling languages in order to express the physical interactions of the players in the IS, raising that way the ceiling of expressiveness. Additionally, the toolkit has been designed with a hardware and software architecture that can be adapted to different IS configurations and that it is able to deal with the heterogeneity of devices and ubiquitous technologies in an indoor IS. The toolkit tries to mitigate the lack of tools in this area and aims to contribute to the spread of hybrid games in interactive spaces. The work proposes ways to tackle the main challenges involved in the development of this kind of games and shows the viability of the proposed solutions.

The structure of the article is as follows: Section 2 presents a state of the art of pervasive games and of frameworks aimed at their development. The design decisions that lead to the development of the JUGUEMOS toolkit and its architecture are explained in Section 4. The detailed explanation of the toolkit implementation is presented in Section 5, while Section 6 shows the three case studies carried out in order to explore the toolkit's expressivity and potential. Section 7 contains a discussion regarding the advantages of out toolkit together with the improvements that remain to be done, and finally Section 8 is devoted to conclusions and future work.

## 2 State of the art

In the next section, first we are going to analyze several examples of pervasive games, focusing on their deployment. After that, we carry out an analysis of frameworks or toolkits aiming to facilitate the creation of pervasive games.

### 2.1 Pervasive games in the literature

It has been mentioned in the introduction that when developing pervasive games it is necessary to augment the game environment to allow the players to interact with the technologies involved in the game. In this first part of the state of the art we categorize the pervasive games analyzed in four categories depending on the augmentation carried out: *objects augmentation* (**smart toys**), *table augmentation* (**tabletops**), *room augmentation* (**Interactive Spaces**), or *world augmentation* (**outdoor games**).

### 2.1.1 Object augmentation

*"Caves & Creatures"* [23] is a tabletop role-playing game that combines a tangible board and physical objects with a computer screen where the actions of the players are shown. Players physically move their characters on the board and use cards augmented with Radio Frequency Identification (RFID) to represent different items that can be traded with other players. RFID technology uses electromagnetic fields to detect and track tags attached to objects. The tags can be detected meters away from the RFID reader and they usually carry a local power source in order to operate. Therefore, this kind of technology is commonly used in ubiquitous computing systems when it is necessary to know the position of certain objects or persons. Among the objects to be used, there is also a magic wand with a built-in accelerometer to recognize the movements that the players make to cast spells.

Martins et al. [27] and Tanenbaum et al. [43] propose storytelling digital augmented environments with *Noon* and the *Reading Glove* respectively. *Noon* is an interactive experience where players have to discover the secrets of a manor by manipulating different objects to trigger the memories they keep stored inside. Players wear a bracer with a built-in RFID reader to detect the different objects and the hand movements. The images and sounds that represent the different memories are displayed in a PDA encased in a hollowed-book, also carried by the players. Similar to the Noon game, the *Reading Glove* uses a similar wearable controller embedding a RFID reader to detect tagged objects. Players have to discover hidden memories inside 12 different objects to find out a story about a spy who has been betrayed. The story is narrated through video and audio feedback on a computer screen.

*"Don't Panic"* [30] is an augmented board game where four players have to collaborate to prevent panic from spreading in certain sectors of a city when random dangerous events take place. The cardboard where the game is played is a traditional physical object but the pieces used in the game are square-shaped augmented artifacts with an integrated LCD screen to show information related to the game. These artifacts have several functionalities: they detect other artifacts by proximity, they can display sounds, and represent the pawns controlled by the players, such as the dice to move the characters through the cardboard and the cards that trigger the game events.

"*BoomChaCha*" [50] is a rhythm-based collaborative game where players have to defeat monsters using different toy weapons. The tracking of the toys and how they are manipulated is carried out integrating in the toys accelerometer sensors wirelessly connected to a computer via an Arduino microcontroller. That way, the computer wirelessly receives sensor data from multiple toys, and responds to players by sending commands to the toys' Arduino microcontrollers to light on LEDs attached to the weapons. Digital feedback is shown in a projection screen.

Park [35] developed the "*Hybrid Monopoly*", a new Monopoly-based hybrid board game that makes use of digital smart phones that can be combined with the non-technological game. In this version of the Monopoly, the playing pieces and board remain as physical elements, but a game calculator replaces the game's bank and the cards become digital elements appearing in the smartphone screen. RFID technology is once again used to detect the position of the playing pieces on the board. Three-multicolor LEDs and a LCD panel provide visual feedback to the player's actions, and audio instructions are given to the player through the smartphone devices.

Finally, Gatti et al. [9] made use of haptic digital augmentation to create a hybrid version of the traditional Foosball game. In this version, goals are replaced by a set of hit sensors situated

on either side of the table. By hitting the sensors with the ball, players can hinder the opponent's movements thanks to a haptic actuator attached to the clamps that applies friction when one of the players' sensors is hit.

#### 2.1.2 Table augmentation

"*PingPongPlus*" [13] is a digitally augmented version of the classical ping-pong game considered one of the pioneers in the field of hybrid games. Digitally augmenting the ping-pong table required to develop a completely new ball tracking hardware, consisting of an audio based sensor capable of triangulating how the ball bounces on the ping-pong table. Graphic feedback is provided by projecting digital images on the ping-pong table, and by generating animated graphics related to the hits of the ball on the table. Several augmented ping-pong game modes were developed, enriching the traditional ping-pong game.

*RoboTable* [20] is a tabletop system that allows users to manipulate robots intuitively. This time, instead of using RFID technology to detect the robot on the table, special printed markers are attached to the robot's base. Digital cameras installed inside the tabletop are in charge of detecting the robot's markers placed on the tabletop surface, and the reacTIVision software framework [15] is used to recognize the markers of the robots. reacTIVision simplifies the tracking of conventional objects on interactive tables attaching ordinary printed markers on the base of the object instead of complex and expensive electronic components. reacTIVision uses the TUIO communication protocol [16], built on the Open Sound Control communication protocol (OSC) [47] in order to allow the transmission of messages which are sent when carrying out manipulations on the tabletop surface.

Other hybrid tabletop games also use reacTIVision. In the "*Farm*" game [4] children manipulate animal toys with reacTIVision markers in different ways: for example, when the child makes the cow and the hen "jump" on the tabletop, the image projected on the tabletop shows milk and eggs respectively, and when the pig toy is moved on the virtual bushes, it collects strawberries. Finally, "*Uprising*" [7] is a digitally augmented board game in which players control zombies and humans playing pieces on the tabletop surface to defend their respective territories. This game uses reacTIVision to detect the playing pieces on the tabletop, and also projects the image of the board on the tabletop surface.

#### 2.1.3 Room augmentation

The *Kidsroom* [2] is an indoor environment that re-creates a child's bedroom where children become the protagonists of a story: "*The Bedroom World*". The children have to go asking the different pieces of furniture of the room in order to discover a magic word that would allow them to travel to other worlds. Visual feedback is provided through wall projections that recreate the room environment and the different worlds the children can travel to, and audio feedback is reproduced via several speakers. The deployment of the game required to use a total of four cameras to track the children's position and movements in different angles, and a detection algorithm was implemented to recognize the actions the children perform while playing.

*Touch-Space* [6] is an interactive room aimed to support mixed-reality based hybrid games where players have to walk around and interact with physical objects. In order to track the players' position in the room, *Touch-Space* uses a Real Time Locating System (RTLS) able to detect the position of the head and hands of the players. RTLS systems make use of wireless

tags attached to objects or persons that send wireless signals to the sensors that compose the RTLS so that the system can triangulate their position. In *Touch-Space*, players also wear a Head-Mounted Display (HMD) to support video-see through augmented reality feedback, and the HMD has a small video-camera to track visual markers attached to different objects of the room. The position of the markers is tracked using the ARToolkit software framework [17]. In addition, the players carry a wand toy that gives visual feedback about the tasks that they have to perform.

*Age Invaders* [18] is an interactive indoor game installation based on the Atari™ *Space Invaders* classic arcade game. In *Age invaders* the spaceship digital sprites were physically replaced by the players themselves. In a room, one team plays the defenders while the other plays the invaders. The floor is digitally augmented with audio and video projection, representing the star background and the missiles that each player shoots. Players use a hand controller to shoot the missiles. When a player is hit, the "explosion" is showed in a LED display that the players wear on their body. The tracking of the players' interactions during the game is provided by installing RFID readers in the players' shoes.

Another mixed reality example is *Treasure* [11], where players have to find objects marked with RTLS tags. In order to gather other physical information related to the objects, such as their orientation or certain changes produced in their surrounding environment, MOTE sensors are attached to certain objects. There are two different kinds of objects: target, the ones that the players have to find to win the game, and supporting, the ones that give hints to the player to find the target objects. *Treasure* has been designed so that it is possible to trigger different digital responses to the player's actions: for example, when they find an object an audio can be played, an image/video can be shown on the wall, or both.

Another game, the *Music Room* [31] is an interactive indoor installation where music can be composed by dancing in the space. In order to detect the dancers' movements, cameras are embedded in the environment and a tracking software was developed by applying digital image algorithms. The information extracted from the camera image analysis is sent to a musical composition digital system using OSC. The composition system analyzes the data from sensors and generates music depending on the way people are dancing (close to each other, far from each other, fast, slow).

### 2.1.4 World augmentation

In outdoor games, players can be widely scattered in places from a campus to a whole city, or even the world. "*Can you see me now?*" [8] is a mobile mixed reality game in which online players play an adapted game of catch with real players (runners). Online players move in a virtual model of the city and runners have to "catch" the online player using a handheld map where the position of other runners and the online players is shown. Runners also wear Walkie-Talkies to communicate with each other. In this case, the whole city becomes the playground where players, online and runners, are collaborating.

Another example of outdoor game is *Treasure* [5]. In this game players have to collect virtual coins in the city with the help of PDAs where a map with the current positions of the coins and the players is displayed. Players compete to collect more coins and they can even steal coins from another nearby player. In both games, GPS sensors are used to track the position of the players, and visual and audio feedback is provided by the use of PDAs.

Finally, "*Save the Safe"* [41] is an adaptation of the traditional game of cops and robbers. In this version, children divided in robbers and cops carry a device with a LED that shows the

color of the team and gives audio instructions. At the beginning of the game, the device of one of the cops vibrates, indicating that he/she has the key that the robbers have to steal to win the game. Only the cop with the key knows it, so the robbers must run after the different cops and approach their devices to the cops' ones to try to guess who has the key, while the cops have to pass the key with their devices to prevent the robbers from stealing it. The devices used in this game embed different hardware components such as an accelerometer sensor to measure movements and a XBee module that allows RF communication to interconnect the different devices and measure distances between devices. Visual and audio feedback is provided by a vibration motor, speakers and LEDs.

### 2.1.5 General conclusions

All the examples that have been explained are summarized in Table 1 in order to show their characteristics more clearly. The second column indicates the way to interact with the respective systems, the third column relates to how the implementation has been carried out, and the fourth column shows the different kinds of outputs found in the games.

After the detailed analysis of the different games, some conclusions can be extracted regarding their common characteristics:

- - The development of hybrid games requires the integration of **a great variety of devices**, and there are some kinds of sensors that are commonly used depending on what to track: games where objects are augmented require RFID technology and visual printed markers that are later tracked by specialized software; when the game needs to "know" the position of the players in the system, RTLS technologies are applied in room augmented games while GPS sensors are used in world augmented applications; and when it is necessary to recognize the player's gestures, accelerometers sensors are integrated. Also, some games make use of actuator hardware, such as *BoomChaCha,* whose toys are reactive to users' manipulations by lighting LEDs in certain colors, and *Augmented Foosball*, where haptic actuators are embedded in the foosball's rods to apply force depending on the player's performance during the game.
- - The hybrid games presented in Table 1 make use of different software toolkits and frameworks in order to carry out certain parts of the implementation, such as reacTIVision and the ARToolkit to process camera digital image to track printed markers. Several programming environments such as Max MSP, Arduino, Processing and Eclipse are used to develop the game logic. In order to communicate distributed hardware components, different communications protocols are used, such as TCP, RPC, Bluetooth and OSC. Very often, the development of hybrid games requires to implement their own detection and tracking algorithms from scratch. Although in some works this could be due to the lack of existing software at the moment (*"The Bedroom World"* and "*PingPongPlus"*), there are relatively recent works that require hardware level coding from scratch too, such as *Robo Table*, *Music Room* and *Augmented Foosball*. In addition, among the 19 works analyzed, in 14 of them **the coding of the game has been made ad hoc**, without using any specific tool or framework that facilitates the task of dealing with the different paradigms involved in the game. The only 5 works that make use of existing frameworks are "*Don't Panic*" with *Anyboard*, "*Caves & Creatures*" with *Pegasus*, "*Farm"* with *ToyVision, "Save the safe"* with *RaPIDO,* and *"Treasure*" with *Sixth-Sense*, which will be analyzed in the next section.

**Table 1** Pervasive games classification based on deployment

| | GAME | INPUT | IMPLEMENTATION | OUTPUT |
|---|---|---|---|---|
| **OBJECT AUGM.** | Caves & Creatures (2006) [23] | Objects<br>Gestural | **Communication**: TCP/IP, Bluetooth<br>**Language:** C<br>**Toolkit:** *Pegasus** | Computer screen<br>Accelerometer |
| | Noon (2009) [27] | Objects | **Communication**: RF | PDA<br>Accelerometer |
| | The Reading Glove (2010) [43] | Objects | **Communication:** RFID<br>**Language**: Max MSP | Computer (audio, images) |
| | Don't Panic (2015) [30] | Objects | **Communication**: RFID, Bluetooth<br>**Toolkit:** *AnyBoard** | LEDs |
| | BoomChaCha (2016) [49] | Objects | **Communication**: TUIO, RF (xBee)<br>**Language**: Processing | LEDs<br>Projection screen<br>Accelerometer |
| | Hybrid Monopoly (2017) [35] | Objects | **Communication**: Bluetooth, RFID<br>**Language:** C++, Java | LCD panel<br>LEDs |
| | Augmented foosball (2017) [9] | Objects | **Communication**: Open loop control (Arduino) | Linear actuator<br>LEDs<br>Impact sensor |
| **TABLE AUGM.** | PingPongPlus (1999) [13] | Sound | **Algorithms**: Hit-detection, sound-based tracking system<br>**Language:** C++ | Projector |
| | RoboTable (2009) [20] | Objects<br>Gestural | **Communication**: Bluetooth<br>**Language**: Java<br>**Toolkit**: reacTIVision | Tabletop surface |
| | Farm (2015) [4] | Objects | **Communication**: TUIO (OSC)<br>**Toolkit**: reacTIVision, *ToyVision** | Tabletop surface<br>Projection screen<br>Speakers |
| | Uprising (2015) [7] | Objects | **Communication**: TUIO (OSC)<br>**Language**: Processing<br>**Toolkit**: reacTIVision | Tabletop surface |
| **ROOM AUGM.** | Kidsroom (1996) [2] | Embodied<br>Gestural | **Communication**: RPC protocol<br>**Algorithms**: Vision-based | Projection screens<br>Speakers<br>Lights |
| | Touch-Space (2002) [6] | Embodied | **Communication**: Markers, RTLS<br>**Toolkit**: ARToolkit | Wand for visual/audio information |
| | Age Invaders (2008) [18] | Embodied | **Communication**: RFID, Bluetooth | LEDs<br>Projection screens<br>Speakers |
| | Treasure (2012) [11] | Embodied | **Communication**: RTLS<br>**Toolkit**: *Sixth-Sense** | Projection screens<br>Speakers<br>MOTE sensors |
| | Music Room (2014) [31] | Embodied | **Communication**: OSC<br>**Algorithms**: Visual tracking | Speakers |
| **WORLD AUGM.** | Can you see me now? (2003) [8] | Embodied | **Communication**: Wi-Fi, GPS | PDA<br>Walkie-talkie |
| | Treasure (2005) [5] | Embodied | **Communication**: UDP, Wi-Fi, GPS | PDA |
| | Save the safe (2013) [41] | Objects<br>Gestural | **Communication**: RF<br>**Language**: Processing, C++<br>**Toolkit**: *RaPIDO** | LED<br>Speaker<br>Accelerometer |

– - With the exception of *Augment foosball*, that does not provide visual output, and *Music Room*, that only uses audio feedback, the rest of the games employ a combination of audio and visual feedback. It can also be observed that there are multiple ways to show visual information (projection screens, controlled lights, tabletop surfaces, PDAs, smart-phones,

monitors), and, unlike traditional games where the player's actions and the feedback obtained take place in the same space, in hybrid games the manipulations that the players carry out to interact with the application do not necessarily happen in the same place where the feedback is provided but in their surroundings, and even in different devices at the same time. Also, giving feedback in different ways means that there are **multiple and heterogeneous displays** in charge of showing some pieces of information in order to make the game evolve.

After the analysis of hybrid games carried out from the point of view of the deployment process, it can be concluded that the creation of hybrid games is not something trivial, since it demands to deal with very different technologies whose software and hardware may greatly differ. For that reason, frameworks or tools able to facilitate the whole development process of hybrid games or, at least, a part of it, are needed, which is what we are going to discuss in the next section.

### 2.2 Frameworks for the development of hybrid games

The design and prototyping of hybrid games arises important technical challenges for interaction and game designers. In literature, we can find several examples of design and implementation processes of hybrid games, as those cited in the previous section. It is common to find that the design process usually starts with paper prototyping to test the game rules [24]. Paper prototypes enable designers of hybrid games to quickly manipulate the game rules, goals, and interactions. However, it would be difficult to evaluate the player experience using only paper prototypes. The common design process relies on interactions with short cycles of implementation of a functional prototype, alternating game design and play-tests to quickly test the effects of game design decisions [29]. As the iterative design process advances, more advanced and high-fidelity prototypes are required. Without a functional prototype and a quick iterative design process, hybrid game experiences cannot be properly evaluated. In this context, a multidisciplinary workgroup emerges: engineers and hardware experts are required to deal with hardware, and software developers to code the application.

Therefore, the prototyping process depends on the communication between designers and developers, but the gap between the design and implementation processes is especially critical in the context of hybrid games [38]: as the prototype becomes technologically more complex, the times between iterations are longer, since changes in the design are translated into critical impacts in the hardware and the code, which also lengthens the implementation process while hindering the iterations [44]. In addition, as outlined in the examples of the previous section, hardware configurations are usually purpose-built for each game and software is written from scratch, from low-level hardware code to high level game logic. Another important question in the context of hybrid games is how existing hardware and software design solutions can be reused, and how higher level prototyping support can be provided to designers so they can contribute to the implementation process, bridging that way the gap between the design and implementation processes. While many rapid prototyping tools are available for the standard platforms like personal computers and smartphones, tools for rapid prototyping of hybrid games are less common.

In the analysis of hybrid games carried out, it was seen that most part of the games had been developed ad hoc, but there were also some of them that used specific frameworks to deal with

the creation of the games. Next, we are going to present these frameworks and focus on the three characteristics that we could extract from the previous section.

Pegasus [23] is a software architecture for developing hybrid games based on tangible interaction. To integrate the different tangible and graphical components, each interaction device is associated with a software proxy in charge of communicating with the other devices to exchange information. PDAs are used to show information related to the game, and the authors have created an XML-based language to store the rules of the game. These rules can be modified in runtime, and the different devices can be connected or disconnected while the game is running.

Sixth-Sense [10] is an infrastructure initially aimed at the development of smart-artifact services but which is also used by the same authors to create hybrid games. Sixth-Sense is based on the OWL ontology to define relationships found in a typical smart home environment (humans, objects, and humans and objects). However, the ontology can be extended by adding other concepts related to games, such as a win/loss status.

AnyBoard [30] is a framework for the development of hybrid board games that facilitates the coding of the game logic and also provides tools to support the design of the games. In order to do so, AnyBoard proposes the Interactive-Token approach, based on the Token+ Constraint framework [45]. The Token+Constraint framework provides designers with an abstraction language to specify any TUI. With this language, designers can define the different interactions of the systems in terms of relationships between elements of the game. This way, AnyBoard reduces the gap between the design and prototyping processes.

ToyVision [26] also facilitates de prototyping of hybrid games for tabletop devices. In this case, ToyVision adopts the Tangible User Interface Modelling Language (TUIML) [39], to provide a common tool between designers and developers. ToyVision provides designers with a GUI assistant that enables designers to draw the TUIML specification of a hybrid game for tabletop devices. This graphic is automatically translated into XML code, facilitating that way the process of coding the logic of the tabletop game.

Finally, RaPIDO [42] is a prototyping platform for outdoor games. In the games developed with RaPIDO, players make use of hand-made devices to interact with the game and the other players. RaPIDO devices are hand controllers which embed a RFID reader sensor in order to detect tagged objects, and an accelerometer sensor to measure movements; feedback is provided by a vibration motor, speakers and LEDS to provide audio and visual feedback. The communication between different controllers is provided by the Arduino microcontroller and the xBEE wireless module which also enables to measure the distance between them. Developers have access to a software API that allows to choose the functionalities of the device that they want to activate, and to program the behavior of the devices using the Arduino or C development environments. Therefore, RaPIDO enables the development of hybrid games in which interaction and feedback are provided by multiple identical hand controllers, which integrate different sensors and feedback hardware to be used in different ways.

Next, we are going to present a summary of the characteristics of the frameworks that have been commented in this section (see Table 2). The first three columns show if the frameworks deal with the three challenges that we have found in the previous section: the integration of different devices (first column), the coding of the application (second column), and the management of displays (third column). The fourth column indicates the end-user the framework is addressed to, and the fifth column shows the category of the games developed by those frameworks: object augmentation (OA), table augmentation (TA), room augmentation (RA) and world augmentation (WA).

**Table 2** Characteristics of frameworks

| Framework | Integration | Coding | Displays | Addressed to | | Category |
|---|---|---|---|---|---|---|
| | | | | Designers | Developers | |
| *Pegasus (2006)* [23] | | ✓ | ✓ | | ✓ | TA |
| *Sixth-Sense (2008)* [10] | | ✓ | | | ✓ | RA |
| *ToyVision (2012)* [26] | | ✓ | ✓ | ✓ | ✓ | TA |
| *AnyBoard (2015)* [30] | | ✓ | ✓ | ✓ | ✓ | OA |
| *RaPIDO (2017)* [42] | | ✓ | ✓ | | ✓ | WA |
| *JUGUEMOS* | ✓ | ✓ | ✓ | | ✓ | RA |

It can be seen that the integration of devices is not considered in any of them. In our case, as we wanted to support pervasive games in Interactive Spaces it was considered a critical point. Besides, display management was also crucial and the support to software coding, a must. These three aims were the ones that guided the design of the JUGUEMOS toolkit. The complexity of the task made us focus on developers. Nevertheless, one additional aim was to lower the gap between designers and developers as we will show in the case of studies section.

## 3 Toolkit architecture

The following section is organized as follows: first, the design decisions that were taken to develop the toolkit are explained; then, the general architecture of the toolkit is described; and finally, we will delve deeper into to the process of integrating different devices and creating a pervasive game.

### 3.1 Toolkit requirements and design decisions

The JUGUEMOS toolkit architecture has been designed with the following requirements in mind, which are related to the challenges extracted from the state of the art:

1. To allow the easy integration of multiple devices.
2. To facilitate the coding of the game application.
3. To support the management of multiple displays.

Regarding the first requirement, solution lies on the use of a standard communication protocol between devices of very different hardware characteristics. In the first part of the state of the art, when analyzing how the pervasive games had been created we realized that, among the works that gave details about the communication protocol that they used, there were some that used the OSC Protocol and the TUIO Protocol (also based on OSC). As we have commented briefly before, OSC is a protocol for networking multimedia devices that has been adopted and incorporated in several development environments aiming physical computing, such as Processing [36], a flexible software aimed to code within the context of the visual arts, Pure Data [37], an open source programming language specialized in visual and sound based applications, or vvvv [46], a programming environment designed to facilitate the handling of large media

environments with physical interfaces. Also, the TUIO protocol was built on OSC in order to allow the transmission of messages between tangible multitouch surfaces. Besides these programming environments, there are many other tools that make use of OSC (and TUIO) to isolate the hardware details of the system from the development of the interactive application, such as ReacTIVision, a toolkit focused on vision based sensors that simplifies the tracking of conventional objects by only requiring attaching a printed marker on the object, and KinectOSC [19], an utility to transmit data from official Microsoft Kinect SDK via OSC, easing the way to deal with the diverse Kinect messages by adopting a more simple format. After seeing that existing frameworks made use of OSC for interconnecting very different applications, we decided to adopt it in our toolkit and to use Processing to implement the whole toolkit due to its facility to implement OSC connections.

Regarding the second requirement, the development of tools that allow an easy and rapid prototyping of applications it is not a trivial matter, due to the low-abstraction level of the data received from the ubiquitous technologies, and the impact in the code when changing the interaction methods. For this reason, the solution to facilitate the prototyping of applications is to allow the development of prototypes in a high-abstraction level, reducing this way the gap between the definition and development of the application. Several works that focus on the definition of ubiquitous technologies in a high-abstraction level can be found in the literature: Shaer et al. [40] present in their work the Token and Constraint (TAC) paradigm for describing and specifying tangible user interfaces, and Hornecker and Buur [12] created a model with four different categories that allow to better understand, define and analyze Tangible interactive applications. However, these last works are focused on helping designers by providing them with means to present and define the physical interactions that are carried out in a Tangible system, but they do not address the development process. But, in order to allow a rapid prototyping of applications, it is necessary a model that facilitates the work of developers. As we have commented before, TUIML addresses this necessity, allowing designers to define the system by using the TAC paradigm, but also proposing a way to translate the application concept into a language understandable by a computer. For that reason, some toolkits have make use of TUIML and the TAC paradigm, such as *ToyVision* and *Anyboard*, which have been already analyzed in the previous section. Until now, TUIML had just been used in the field of Tangible Interaction, but after analyzing it we realized that it could also be extended to other kind of interactions, so we adopted it as a model for our toolkit in order to raise the ceiling of expressiveness in the creation process of pervasive games.

Finally, regarding the third requirement, pervasive games that are played in interactive spaces where multiple interaction paradigms converge require to show information in very different displays, such as projection screens, mobiles, tabletop surfaces, or even the floor itself. For that reason, in order to be able to distribute the game's visual information in multiple displays easily, we decided to ground our work on the Virtual Space topology paradigm [34]. In this paradigm, the interactions between the different displays are limited and determined by a virtual space defined by the system that covers all the physical displays involved in the system. Also, every time that it is necessary to show information related to a specific device, each one of them provides a viewport into this virtual space that corresponds to the actual physical display where the information must appear. In a later section we will explain in more detail how this Virtual Space paradigm was adapted in our toolkit.

### 3.2 Architecture description

Figure 1 sketches the toolkit architecture.

The JUGUEMOS toolkit is based on a centralized network architecture in which a software **Broadcaster** is individually connected with the **Host** application and with each of the hardware devices integrated in the IS (display, sensor or actuator). Each device has associated a software process in charge of dealing with the specific hardware issues:

- Each **display** device has associated a "**Painter**" software process in charge of painting visual information or playing audio streams in the specific display device.
- Each **actuator** device has associated a "**Publisher**" software process in charge of sending commands to the corresponding hardware actuator. Actuator devices are capable of performing physical actions perceptible by the users (e.g., object movements, turning on/off lights, etc)
- Each **sensor** device has also associated a "**Publisher**" software process in charge of processing and interpreting the raw data captured by the hardware sensor.

Communication between the devices and the Broadcaster is based on the OSC Protocol. Therefore, all devices of the IS are connected to a local network by Ethernet or Wi-Fi. The Broadcaster keeps an UDP network socket with each device and with the **Host** (the application in charge of managing the game logic). The Broadcaster serves as a central network distribution node in the IS network: on the one hand, it redirects the commands that are sent by the Host to the displays and actuators; on the other hand, it redirects the data of the sensors to the **Semantic Level**. This last process is in charge of interpreting the data sent by the sensors in a high abstraction level. As previously noted, the

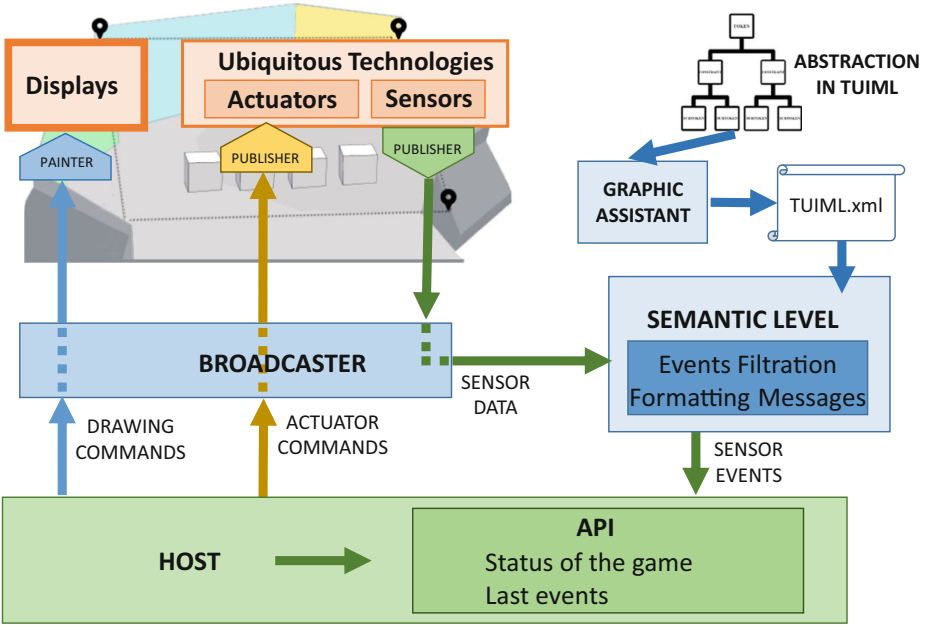



**Fig. 1** Toolkit architecture

JUGUEMOS toolkit uses the TUIML modeling language to **abstract the data** received from the sensors as meaningful physical manipulations of players in the context of a specific hybrid game. In order to do this, the Semantic Level needs to previously "know" the context of the game that is running on the IS. Once the game is being defined in TUIML terms, the **Graphic Assistant** allows to translate this model into an XML file (**TUIML.xml**). The Graphic Assistant is also directly connected to the Broadcaster in order to visually show all the Publishers and Painters connected to it, and also the elements detected by the different ubiquitous technologies in real time, so that it is easier for the developer to define the TUIML.xml file. The data written on this file allows the Semantic Level to filter and process the sensor data in the context of the game, keeping an updated status of all the physical elements involved in the IS during the game.

Game developers will be in charge implementing the Host application; therefore, they need to access this updated status in order to make evolve the game logic (as response of players' physical actions). For that purpose, the developers use an **API** that provides several functions to consult the status kept by the Semantic Level. That way, the development of the game is completely isolated from the hardware details of the IS. Also, it was decided to separate the Semantic Level from the coding of the game so that developers can code the game in any computer language, as far as an OSC socket could be implemented. At this moment, we have an API implemented in the Processing Programming Environment, but the API can be easily translated into other programing environments.

The creation of the JUGUEMOS toolkit was an iterative process in which we incrementally developed and added new tools according to the requirements that have been commented at the beginning of this section. Following we will explain this process in detail.

## 4 JUGUEMOS toolkit development

The creation of the JUGUEMOS toolkit was carried out iteratively in three different stages that correspond to the three requirements explained in the previous section. In each stage, an example of use to show the fulfillment of the requirement was carried out in the JUGUEMOS IS, an IS built in the ETOPIA Arts and Technology Center in the city of Zaragoza (Spain) thanks to a national research project.

### 4.1 Stage 1: Integrating heterogeneous devices

This first stage is related to the first requirement, consisting of carrying out the integration of very different and diverse displays by making use of the OSC Protocol (see Fig. 2).

Each electronic device (sensor, actuators and displays) needs a software process (painters and publishers) implementing a standardized communication protocol with the Broadcaster process, using a common format of OSC messages. The Broadcaster is a software process that runs in a computer with a unique IP address. The Broadcaster opens a network port (i.g: 32000) to listen to the "Publisher" managers. In order to connect with the Broadcaster, each painter and publisher process has to send the Broadcaster an OSC message indicating the kind of device that is connecting (actuator, sensor or display) in the OSC address, and the port number in which the device will listen to the OSC messages coming from the Broadcaster (see Table 3).

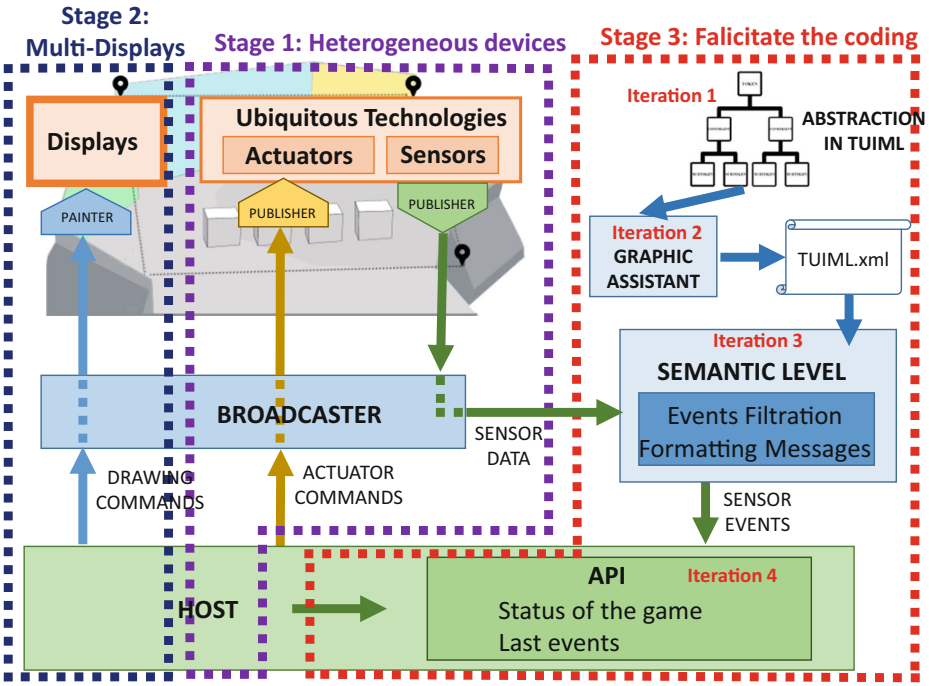



**Fig. 2** Stages to implement the toolkit

That way, the Broadcaster can keep a list of all the devices connected, and it is able to receive and send OSC messages to any particular device, to a group or to all the connected devices (see Fig.3).

Once connection is established, sensor publishers can send OSC messages to the Broadcaster with the data captured by the sensor hardware; actuator publishers can receive OSC messages from the Broadcaster with commands to be performed by the actuator hardware; and painters can receive OSC messages with paint commands to be draw (image displays) or played (sound devices) in the display hardware. All the devices carry out the connection in the same way as showed in Fig. 4. The only difference is the value of the variable **type** that is "sensor", "actuator", "display" or "host" depending on the device that is connecting.

The "Publisher" managers associated to sensor devices store one or several variables that correspond to the physical magnitudes that the sensor tracks. These physical magnitudes are characterized by the number of dimensions that they need in order to represent the numerical value that has been tracked:

- **0D**: the tracked value is binary (1 = true/0 = false); i.g.: a sensor button has two states: "on" and "off".

**Table 3** OSC Message format to connect to the Broadcaster

| Address (string) | Port (int) |
|---|---|
| actuator/connect | value |
| sensor/connect | value |
| display/connect | value |

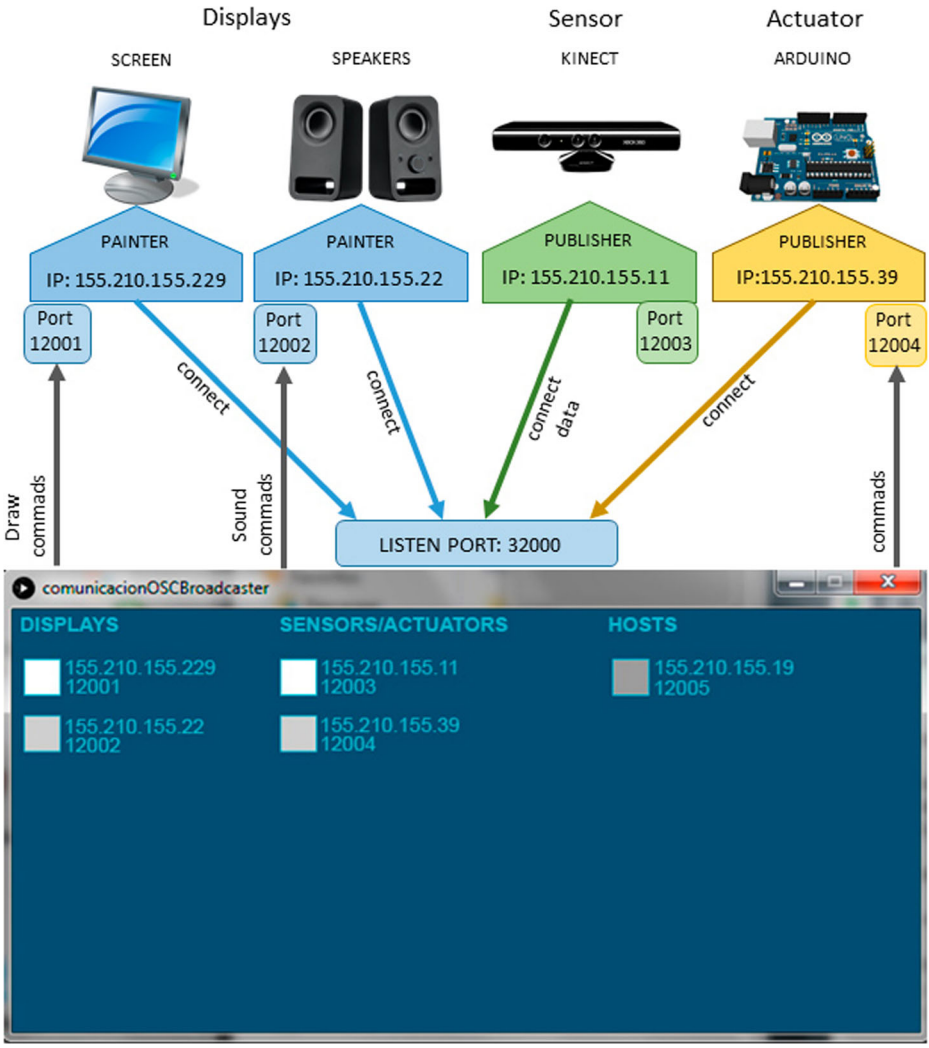



**Fig. 3** List of devices connected in the Broadcaster

```
OscP5 oscP5;
NetAddress myBroadcastLocation;
String remoteIP="155.210.155.1"; // IP address of the Broadcaster
int listenPort=12000;                    // Port in which the "Publisher" manager listens
                                         // to the Broadcaster messages
oscP5 = new OscP5(this,listenPort);
myBroadcastLocation = new NetAddress(remoteIP,32000);
OscMessage m;
String address="/" + type + "/connect";
m = new OscMessage(address,new Object[0]);
m.add(listenPort);
oscP5.send(myMessage, myBroadcastLocation);
```

**Fig. 4** Processing code to connect to the Broadcaster

- **1D**: the sensor tracks a numerical value; i.g.: a micro sensor that captures the audio volume.
- **2D**: the sensor tracks a bi-dimensional vector; i.g.: a sensor that track 2D position of objects on a surface

At this moment, our toolkit is limited to 2D sensors for simplicity, but it can be upgraded to support 3D sensors. Also, each dimension has a certain set of associated manipulations:

- **Add**: a new object has been tracked in the IS (value1 argument is 1), or has stopped being identified (value1 argument is 0).
- **Move**: an object that has been previously identified has moved inside the IS. The value1 argument is the new value of the object if the dimension is 1D, and (value1, value2) is the bi-dimensional vector if the dimension is 2D.
- **Rotate**: an object that has been previously identified has rotated. value1 is the new angle of the object

Sensor categorized as 0D have just the "add" manipulation, since they just allow two different status (button that is "on" or "off", object that is "placed" or "removed"), while 1D-sensors and 2D-sensors have the three of them.

The format of the OSC messages to send messages between the different "Publishers" and the Broadcaster is shown in Table 4. The **address** of the OSC message indicates the name of the device and the dimension of the data that it is going to be sent. The **manipulation** is also indicated as an argument. All numerical values are normalized between 0 and 1 with the exception of the rotate **values**, which are ranged between 0 and 2π radians. Finally, the rest of the arguments are the **ID**, which corresponds to the name of the object that has been manipulated, and **SessionID**, which is the copy number of the object that has been manipulated, enabling this way to track identical objects (same ID) individualized by the "session ID". Also, the structure of the messages that the Broadcaster sends to the "Publisher" managers of the actuators is the same than the one explained in Table 4 changing the first parameter of the address with "actuator" instead of "sensor".

#### 4.1.1 Case of use: "Butterflies" game

In order to prove that our Broadcaster network worked, the "Butterflies" game was developed. In this game, the sensors involved were:

- A Real-Time Localization System (RTLS) to track the children's position on the IS area thanks to small active RF beacons that they wear hanging around their necks, and whose position is triangulated by four sensor receptors placed on the IS corners.
- A microphone, in order to detect when children were shouting.

**Table 4** OSC Message format of the sensors

| Address | Manipulation | ID | SessionID | Value1 | Value2 |
|---|---|---|---|---|---|
| /sensor/0D/device_name | add | – | – | 0/1 | |
| /sensor/1D/device _name | move | – | – | 0 to 1 | |
| | rotate | – | – | 0 to 2π | |
| /sense/2D/device _name | move | – | – | 0 to 1 | 0 to 1 |

Also, regarding the displays involved, a projection screen was in charge of showing an animation with several butterflies flying. While children are quiet, the butterflies fly around until they eventually land on the flowers. However, if children shout, the butterflies that are closer to the children get scared and fly away from the children (see Fig. 5).

In "Butterflies", the child's meaningful manipulations are to walk in front of the projection screen next to the butterflies, and to shout in order to scare them. Table 5 shows the OSC messages that must be generated by the sensors "Publishers" to carry out this game logic.

The "**child**" ID corresponds to the id of the RF beacon that the child is wearing in order to be detected by the RTLS system. Since in our case we do not have more than an RF beacon with the same ID, it is not necessary to differentiate copies of the same ID, so the SessionID field is 0. The same way, the microphone just detects noise but it is unable to differentiate voices, so its IDs and SessionIDs will be always 0, which may difficult the creation of certain games where it is necessary to know when a specific child is speaking or shouting.

Also, to test whether the format of the OSC messages was flexible enough to deal with heterogeneous devices, we developed another version of the "Butterflies" game that used a sensor different from the microphone. In this new version, we used a *Muse* band [32], an EEG sensor that measures the concentration and relaxation of the user. In this case, while the user remains calm or is not concentrating, the butterflies stay on the flowers. However, when the user gets nervous or concentrates, the butterflies run away from him/her (see Fig. 6).

It can be seen in Table 6 that the only difference of changing the sensor that is being used is the name of the different elements, but the way to send the data with the EEG sensor would be the same than with the microphone sensor.

### 4.2 Stage 2: Multi-display management

This second stage is related to the second requirement, consisting of dealing with multiple displays such as monitors, projectors, and Android smartphones and tablets. In order to do so, the most widely used methods consist of (1) using specific hardware for each display, (2) using transmission protocols based on streaming. However, in order to guarantee that our toolkit could work with as many configurations as possible, we could not use these alternatives.

First, because we wanted to keep the homogeneity given by OSC in order to connect all the elements in the system (displays included), since using hardware devoted for each display would imply a specific integration that would go against the first challenge of common integration.

**Fig. 5** Child scaring butterflies in one projection screen

**Table 5** OSC messages that the sensors send for "Butterflies" game

| Address | Manipulation | ID | SessionID | Value1 | Value2 |
|---|---|---|---|---|---|
| RTLS | | | | | |
| /sensor/2D/RTLS | move | RF beacon | 0 | x | y |
| MICROPHONE | | | | | |
| /sensor/1D/micro | move | 0 | 0 | [volume] | – |

And second, because not all devices support streaming, since it greatly depends on hardware and some of the devices that may be used in IS (mobiles, tablets, or even Arduinos) may not have the necessary features to implement it. Also, streaming tends to consume considerable bandwidth, which in our case has to be avoided in order to assure the correct transmission of the messages between the Broadcaster and the devices.

Therefore, our solution to implement the management of displays consisted of treating displays as any other ubiquitous device, in order to keep the homogeneity of the toolkit and guarantee the easy integration of devices. In order to do so, we implemented multiplatform software "Painter" processes associated to each display. A "Painter" process provides the Host application with functions that allow drawing graphic elements (lines, shapes, bitmaps, etc) and reproducing sounds. These processes also listen to OSC messages coming from the Host application, containing commands to paint or reproduce the different elements in the displays. The resources such as images, videos and sounds used in the game are stored in the computer hosting the "Painter" processes, so the Host does not need to send the resource via streaming, but just to indicate the resource that needs to be displayed via making use of some predefined commands.

The way to integrate different displays is to consider each display device as a part of a Virtual Space (VS) that covers them all. Figure 7 shows an example of how to arrange different displays inside the Virtual Space.

By using this technique, it is possible to adapt different displays configurations in the IS, as far as the virtual space area covers all device displays involved in the IS. The arrangement of the displays is defined an XML file (displays.xml). The content of the "displays.xml" is composed of an **ID** that allows to identify the device, the **position** of the display (the (0, 0) coordinate corresponds to the bottom-left corner), and the **dimension** of the display.

**Fig. 6** Butterflies run away when the player concentrates

**Table 6** OSC messages that the sensors send for "Butterflies (EEG)" game

| Address | Manipulation | ID | SessionID | Value1 | Value2 |
|---|---|---|---|---|---|
| RTLS | | | | | |
| /sensor/2D/RTLS | move | RF beacon | 0 | x | y |
| MUSE | | | | | |
| /sensor/1D/Muse | move | muse_id | 0 | [concentration] | – |

Each paint command received by the Broadcaster is in Virtual Space coordinates. Using the information stored in "displays. XML", the Broadcaster can locate which display devices must paint the command, convert its coordinates to local display device coordinates, and send the OSC command to each display device in particular. Table 7 shows some of the OSC messages that the Broadcaster sends in order to draw commands or reproduce sounds.

The address of the OSC message indicates the command, while the arguments of the message are the values needed to carry out the command. The **id** of the display is always sent so that the "Painter" knows where to paint inside the Visual Space. The graphical elements such as text, lines, ellipses and rectangles have three arguments (**r**, **g**, **b**) that represent the color, while the commands to display images or sounds need to send the **name** of the resource. Some of the arguments correspond to the different positions of the elements (**x**, **y**, **×2**, **y2**) and other characteristics such as **content**, **width**, **height**, **size** or **thickness**. Other more specific arguments are the **type** that appears in the "*/display/playVideo*" and "*/display/playAudio*" commands that can be "play" or "loop"; the **pivot** argument used to take the "center" or the "corner" as a reference to draw; and the **frame** in "*/display/jumpVideo*", which indicates the specific frame from where we want to reproduce the video.

This list of commands can be extended by implementing the corresponding function in the "Painter" processes, and creating a new OSC message with the format previously presented. The use of some of these commands will be shown later in the subsection where the coding of the game with the API is explained.

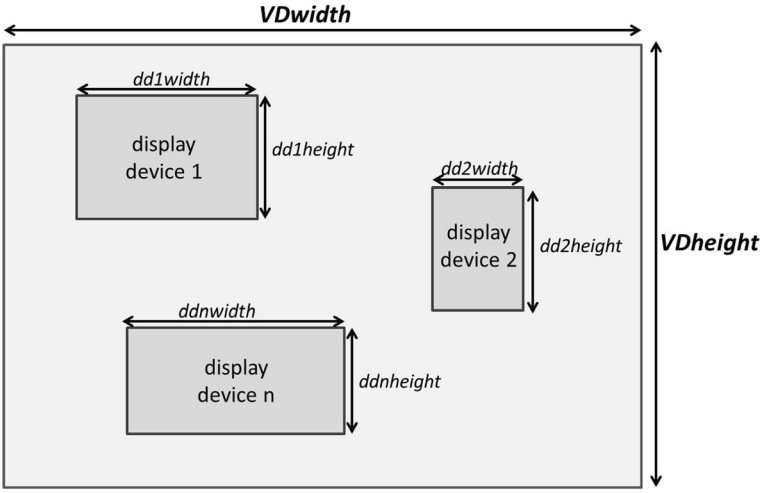



**Fig. 7** Example of integration of displays

**Table 7** OSC Message format of the displays

| Address | Arguments | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| /display/text | id | r | g | b | content | x | y | size | – |
| /display/line | id | r | g | b | ×1 | y2 | ×1 | y2 | thickness |
| /display/ellipse | id | r | g | b | x | y | width | height | thickness |
| /display/rectangle | id | r | g | b | x | y | width | height | thickness |
| /display/drawImage | id | name | x | y | width | height | angle | pivot | – |
| /display/playVideo | id | name | type | x | y | width | height | – | – |
| /display/stopVideo | id | name | – | – | – | – | – | – | – |
| /display/jumpVideo | id | name | frame | – | – | – | – | – | – |
| /display/playAudio | id | name | type | – | – | – | – | – | – |
| /display/stopAudio | id | name | – | – | – | – | – | – | – |

#### 4.2.1 Case of use: Multi-display "butterflies" game

We proceeded to use this technique of integration of displays by re-designing the "Butterflies" game by using the three projections walls instead of just one of them. This way children would be able to walk around the entire IS instead of being forced to just move in front of one single projection screen, clearly improving the game experience.

This time, three entries will appear in the "displays.xml" file, each one of them corresponding to the size of the projection walls (see Fig. 8).

In order to show the different elements in the screen, the Host will use a "draw" function in charge of formatting an OSC message and send it to the "Painter" process. Figure 9 shows the code necessary to draw the background in the different screens.

Finally, Fig. 10 shows the result of the "Butterflies" game being extended to multiple displays, where children can scare butterflies by getting close to the different screens.

### 4.3 Stage 3: Easing the coding of the game

This third stage is related to the third requirement, consisting of easing the coding of the application. This stage has a total of four different iterations: we first adapted the TUIML model to define the pervasive games, afterwards the Semantic Level was added to the toolkit to filter the different messages, a Graphic Assistant was also created to make the definition of the game easier, and finally an API to facilitate the creation of the game logic was developed. As a case of use, and Interactive Space game found in the literature, selected for the variety of the interaction supported, was developed.

```
<xml >
<virtualDisplay width="3840" height="1248">
        <display id="1" x="0" y="0" width="1280" height="768" />
        <display id="2" x="1280" y="0" width="1280" height="768" />
        <display id="3" x="2560" y="0" width="1280" height="768" />
</virtualDisplay>
</xml>
```

**Fig. 8** "displays.xml" file of the "Butterflies" game

```
displays.draw("background_screen1.jpg", 0, 0, 1280, 768, 0, "corner");
displays.draw("background_screen2.jpg", 1280, 0, 1280, 768, 0, "corner");
displays.draw("background_screen3.jpg", 2560, 0, 1280, 768, 0, "corner");

public void draw(String name, float x, float y, int width, int height, float angle, String pivot) {
  idDisplay = "";
  for(int j = 1; j < displayList.length; j++)  {
        // calculation of the display on which the element will be drawn by using the x, y, width, and
        // height arguments (idDisplay)
  }
  myMessage = new OscMessage("/display/draw");
  myMessage.add(idDisplay);       // 1, 2 or 3
  myMessage.add(name);            // name of the element that is going to be drawn
  myMessage.add(x);               // x position of the element
  myMessage.add(y);               // y position of the element
  myMessage.add(width);           // width of the element
  myMessage.add(height);          // height of the element
  myMessage.add(angle);           // rotation angle of the element
  if (pivot.equals("center"))  myMessage.add("center"); // coordinates relatives to center or corner
  else  myMessage.add("corner");                        // of the element
 sendOSCMessage(myMessage);
```

**Fig. 9** Processing code to draw elements in multiple displays

### 4.3.1 Iteration 1: Abstracting the game with the TUIML syntax

Until this point, the toolkit was dealing with the raw data that the sensors sent, which meant that the Host application received all the messages of all the sensors that were connected Broadcaster, even if some of them were not meaningful for the game: for example, in the "Butterflies" game if a tabletop was connected, the Host received its messages even if that device is not used in the game. As the complexity of the game prototypes grew, we decided that it was necessary to filter the messages that the Host received, and also to define them in the context of the game to facilitate the development of the game logic.

In section 3.1 we already discussed the advantages that using TUIML could offer, and that our hypothesis was that a broader interpretation of TUIML would also allow to define the interactions produced in a pervasive game in an interactive space. In [39] a **Token** was any physical element that appeared in a TUI, and a **Constraint** was a physical restriction that a

**Fig. 10** Multi-display version of "Butterflies" game

token imposed to the manipulation of another token, called **Subtoken**, of the first one. Figure 11 shows this hierarchy.

In TUIML, the relations established between two tokens (Token and Subtoken) limited by Constraints are described using TACs. A TAC is a conceptual structure used to describe the status of a subtoken in a specific instant of time; this status is defined as the relation of that subtoken inside the Constraint of a bigger token. The TAC structure can be easily used to describe the current status of a board game.

In pervasive games for IS, this hierarchy remains with the difference that the elements involved on it are not just playing pieces that are manipulated on a board divided in cells. As the tangible interaction design space expands, also the tangible interaction perspective extends. In an IS, users can interact with the game by simply walking on the floor, by manipulating a variety of digital augmented objects, by making certain gestures, or by physical interacting with other player. From the TUIML point of view, the different elements involved in an IS, such as different furniture, interactive walls, and the floor itself, become **Tokens**; the physical restrictions imposed by these Tokens, such as the surface of a table furniture, the surface of a wall and the floor itself, are **Constraints**; and the elements that are manipulated on these Constraints, such as toys and the user who is moving in front of the wall and on the floor, are the **Subtokens**. Hierarchy of tokens and subtokens can have multiple levels: last subtokens can also have associated Constraints: i.e., the user is a Subtoken inside the IS, but at the same time the user can grab a toy, and establish a new level of token and constraint relationship: the toy is a subsubtoken of the user, which is a subtoken of the IS (a token).

In order to describe the states of physical elements and users in a IS at a specific time instant, a **TAC palette** is used, and each TAC of the TAC palette describes a different relationship between two tokens through a constraint. **Digital variables** bounded to a TAC are used to represent physical properties of the subtoken inside the Constraint. Originally, the TAC paradigm defined Constraints as mechanical restrictions to the manipulation of token. Physical manipulations take place in the 3D space, so Constraints have been always defined as regions in the space. However, Contraints can also be classified as 1D, 2D and 3D, and depending on their dimension they will enable a different set of physical manipulations. The 0-Dimension (0D) represents the binary status of an element (on/off, true/false, add/remove) and it is a manipulation contemplated in all the Constraint types (see Fig.12).

This re-interpretation of TUIML also allows describing pervasive games, since it is just necessary to classify the different elements involved in the game with this hierarchy of Token, Constraints and TACs. Nevertheless, we are aware that TUIML has also certain limitations.

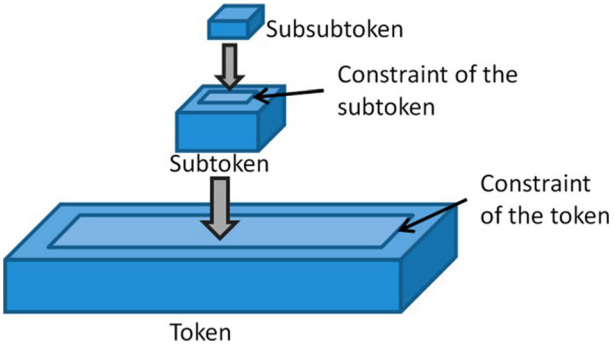



**Fig. 11** Representation of hierarchy between Tokens and Constraints

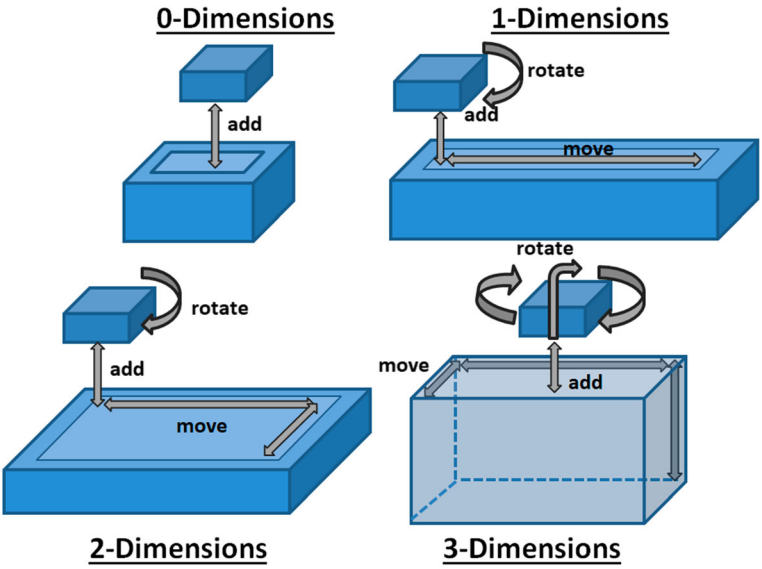



**Fig. 12** Manipulations according to the dimension

For example, TUIML does not allow representing the state of the elements, just relations between them, so this model falls short when trying to define games in which the status of objects is meaningful, limiting that way the range of games to be developed with the toolkit. However, we believe that its capacity to define relations between elements in a rather simple way facilitates the iterations during the development process, as we will show in the last subsection of this section.

#### 4.3.2 Case of use: Designing "The Bedroom World" with TUIML

To assess the potential I the TUIML approach, the "*The Bedroom World*" pervasive game from Bobick et al. [2] was chosen due to the variety of interactions supported. In "*The Bedroom World*" a child enters an interactive bedroom where different pieces of furniture talk when the child approaches them. The child has to go asking around until discovering a magic word that will allow him/her to travel to another world (see Fig. 13). At the end of the game, the voice of the child's mother sounds indicating the child to go to bed. When the child enters the bed a monster appears asking the child to say the magic word. The child does so, the game ends and the next world begins.

The original game did not use any tabletops, but in order to make the most of the devices currently installed in our IS we decided to change the table that the child has to approach for a tabletop device on which the child has to place his/her hand to "make it talk". For this reason, we will call the adapted game "*The Augmented Bedroom World*" (TABW) from now on.

In order to detect the child's manipulations, the following sensors of our IS are used:

- The RTLS to locate where the child is in the IS.
- A tabletop to detect when the child places a hand on the Table.
- A microphone to detect when the child is shouting the magic word.

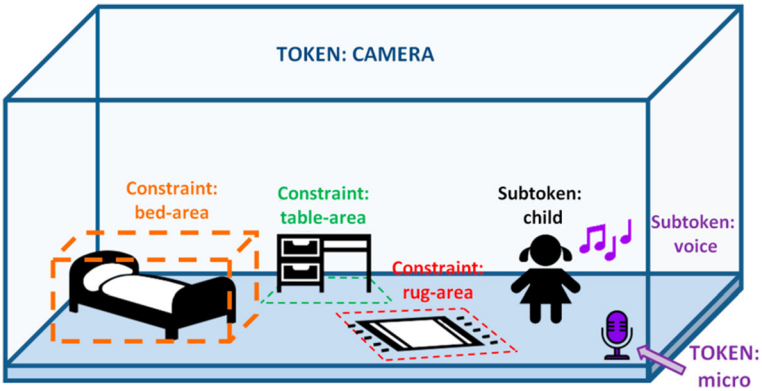



**Fig. 13** "*The Bedroom World*" game

Therefore, the TAC palette of TABW is composed of three different Tokens: (1) the **RTLS Token**, that will detect when the **RF beacon** that the child carries (Subtoken) is close to the rug and the bed (Constraints); (2) the **Tabletop Token**, that will detect when a child places a **finger** (Subtoken) on the tabletop; (3) and the **Microphone Token**, that will detect when someone shouts with a volume inside a certain range (Constraint). Since the microphone could not differentiate the children's voices, a default value = 0 was assigned to the volume detected by it (Subtoken) (see Table 8).

The manipulations that we are going to take into account are "add" instead of "move" because we just need to know when the child enters the rug and the bed areas, and when the finger is placed on the tabletop, but nor their exact position. For the Microphone though we use the "move" manipulation because we need to know the exact value of the voice in order to know if the child is shouting or not. Figure 14 shows the TUIML hierarchy of TABW.

With this example we wanted to prove that expanding the design space of TUIML could cover new interaction modalities in Interactive Spaces, opening the possibility of the creation of a toolkit that supports the building of pervasive games in IS.

### 4.3.3 Iteration 2: Graphic assistant

TUIML allowed us to model the different interactions of the games, but it was necessary to translate them into a language understandable by the computer system. For this reason, we created a Graphic Assistant that allowed (1) to define these interactions by following the hierarchy of Token-Constraint-Subtoken, and (2) to create an XML file containing the game's descriptions in terms of that hierarchy.

**Table 8** TAC palette for TABW game

| Manipulation | Token | Constraint | Subtoken | Values |
|---|---|---|---|---|
| add | RTLS | rug-area (2D) | RF beacon (child) | 0/1 |
| add | RTLS | bed-area (2D) | RF beacon (child) | 0/1 |
| add | tabletop | surface (2D) | finger | 0/1 |
| move | micro | shout-range (1D) | 0 | 0 to 1 |

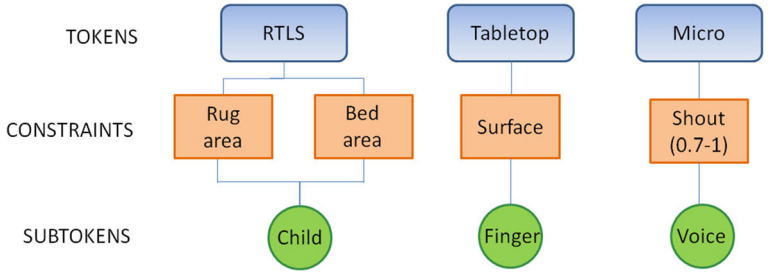



**Fig. 14** TUIML hierarchy of the TABW game

As we have commented before, the Graphic Assistant connects directly with the Broadcaster to show in runtime all the Publishers and Painters connected to it together with the name of the device (sensor, display or host) associated to them (see Fig. 15a). Display, host and actuator devices just show the IP where they are connected. However, when clicking on the name of one of the sensors (Tokens) connected to the Broadcaster, the Graphic Assistant shows the objects (Subtokens) that are currently being detected by the sensor.

The Graphic Assistant translates all these devices into a list of Tokens, the elements detected by those devices into Subtokens, and the areas where manipulating those elements have meaning in the game into Constraints. This hierarchy is then translated into an XML-file. The Graphic Assistant also allows working with a previously created XML file, in case it is necessary to carry out some modifications (see Fig.15b).

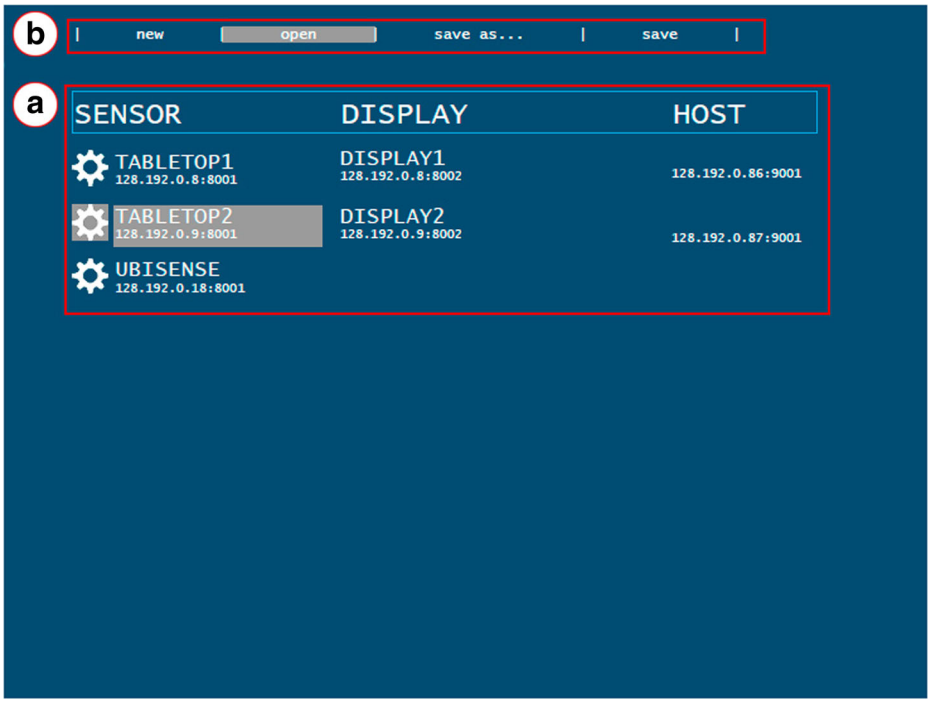



**Fig. 15** Devices connected appear in the Graphic Assistant main screen (a) and also the management of the xml file(b)

### 4.3.4 Example of use: Defining the "The Augmented Bedroom World" with the graphic assistant

In "*The Augmented Bedroom World"* three sensors are involved: the RTLS to detect where the children wearing RF beacons are, a tabletop to detect when someone places a finger on it, and the micro to detect when children are shouting. Therefore, to define this game with the Graphic Assistant, we need to create the constraint and subtokens associated to those sensors.

As we have commented in the previous section, we are going to consider two different constraints associated to the RTLS sensor: when children step on the rug and when they step on the bed. The process to create these two constraints is as follows: after running the Broadcaster, we initiate the Graphic Assistant and click on the RTLS sensor name. When doing so, the Graphic Assistant will show where the RF beacons are in the space. To define the constraint correctly, we situate two RF beacons in the place of the IS where we want to situate the rug and the bed. Those RF beacons will appear on the Graphic Assistant, so we just need to draw a constraint associated to the RTLS sensor around those RF beacons (see Fig. 16). The constraints are created by clicking directly on the screen in order to compose the vertex of the constraint. Once the constraint is created, it is necessary to assign a name to it (BED and RUG in the example) and to select the tokens to be manipulated inside that constraint. The beacons that appear on the image (147 and 255) are the ones that children are going to wear when playing, so we add them as subtokens of the two constraints.

The way to define the constraint of the Tabletop would be the same than the example showed in Fig. 16, with the difference that the constraint defined will cover all the surface.

Finally, to define a constraint associated to a 1D-sensor (such as the micro) we add a new constraint, which at first does not have any range associated. The constraint selected is highlighted in white to differentiate them from the others that have been defined (Fig. 17a). After selecting the desired constraint, we proceed to indicate its range. The first bar indicates

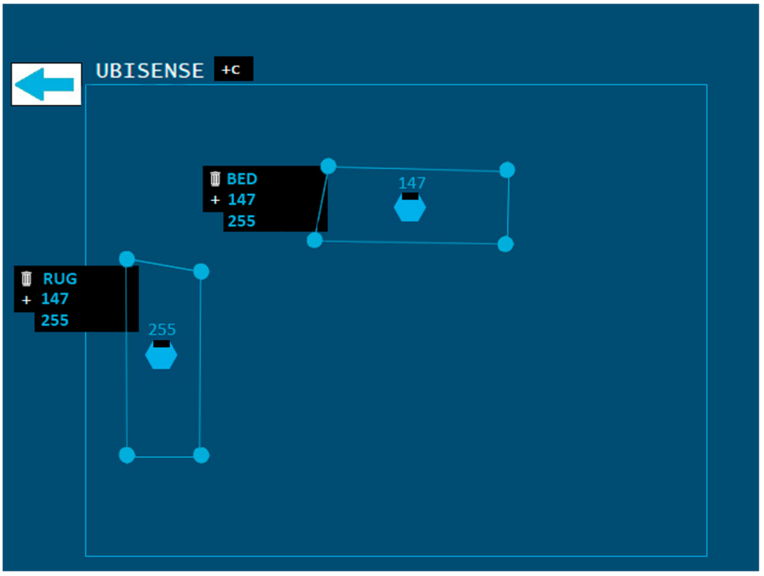



**Fig. 16** Definition of "step on the rug" and "step on the bed" interactions

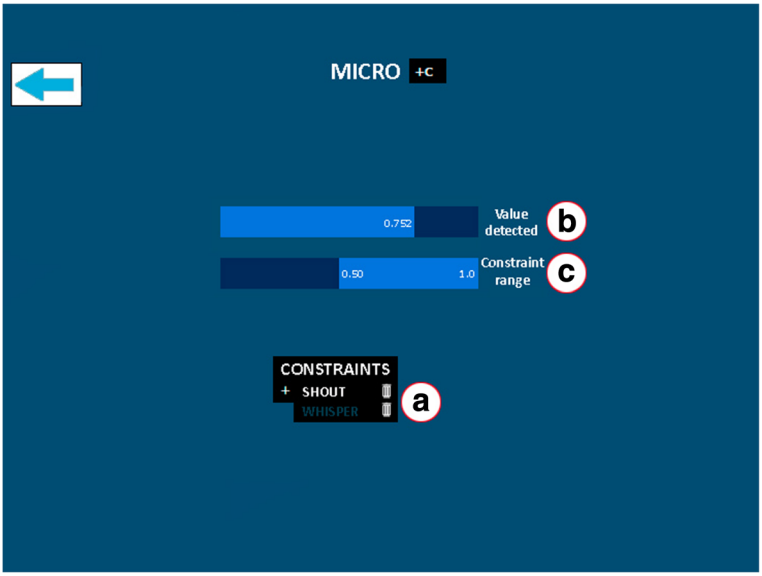



**Fig. 17** Definition of the "shouting" interaction. Constraint selected (a), value detected (b) and constraint range (c)

the values that the micro sensor is detecting in real-time. In Fig. 17b the user is shouting at the micro, which is detecting a normalized value of 0.752. By doing this, the developer can establish in the second bar a range between 0.5 and 1.0 for the constraint, which covers the previously detected value (see Fig. 17c).

#### 4.3.5 Iteration 3: Semantic level

The function of the Semantic Level is to create and keep updated the TAC palette explained in section 4.3, which, as its name indicates, contains a set of TACs describing relations that take place in the IS between two tokens through a constraint.

In order to do this, OSC messages arriving to the Broadcaster from the sensor publishers are processed by the Semantic Level thanks to the "TUIML.xml" that has been created with the Graphic Assistant, where the different elements of the game have been defined in TUIML terms. The Semantic Level uses the **ID** of the message that has been sent by the sensors to identify the element that has been manipulated. If the **ID** of the message that has arrived is defined in the "TUIML.xml" file, the next step is to consult the **type** of the Subtoken (0D, 1D or 2D), and in case of being 1D or 2D, the Semantic Level will extract the position of the Subtoken to know if it has been manipulated inside a Constraint. This way, raw messages from sensor are:

- **Filtered**: OSC messages from tokens not related to the game (not defined in the "TUIML.xml" file) are discarded. OSC messages from Tokens related to the game but that falls out of all the Constraints defined in the "TUIML.xml" are also discarded
- **Abstracted**: by defining Constraints, game events are identified by constraint names. Also, relations between Tokens, Subtokens and Subsubtokens are identified as manipulation inside Constraints.

#### 4.3.6 Case of use: Filtering events for "The Augmented Bedroom World"

Table 9 shows the events related to "*The Augmented Bedroom World*" (TABW) that the Semantic Level will receive from the different sensors.

The Semantic Level receives all the manipulations related to the sensors involved in the game, so it is necessary to filter the ones that have meaning in the context of the game. The constraints do not appear in the messages received because, at this point, the Semantic Level only has the data sent by the sensors.

The next step is to take this data, that in this case belongs to the interactions carried out in TABW game, and check if those interactions are taking place in the constraints that have been defined with the Graphic Assistant in the context of the game. For example, the Semantic Level will receive messages every time that a child with an RF beacon moves, but when consulting the XML generated by the Graphic Assistant it will just take into considerations when a child moves inside the rug area and the bed area.

In order to do this, the Semantic Level reads the XML, where the constraints are defined with their vertex, and calculates if the RF beacon position (2D move manipulation) is inside that constraint. The same way, the Semantic Level filters the values sent by the micro sensor inside the range (0.7–1.0), which was defined with the Graphic Assistant previously.

#### 4.3.7 Iteration 4: API

With the Graphic Assistant, the creation of the "TUIML.xml" file that contained the definition of the game was greatly simplified, easing that way the developer's work. However, it was still necessary to provide developers with functions that easily allowed to consult the status of the TAC palette at any moment, which was the objective of this last iteration.

The Semantic Level keeps updated the whole physical status of the elements involved in the IS as a list of instantiated TACs, which represents all the manipulations taking place at any moment. Each token involved in the pervasive game has its own TAC list, expressing that way its relation with other subtokens through manipulations inside constraints. A TAC list is a table in which each row is an instantiated TAC, while the columns contain the information related with the TACs (see Table 10).

The **first** field is the name of the constraint in which the Manipulation, the **second** field, has taken place. These manipulations are the ones described in subsection 4.1. The **third** and **fourth** fields are the ID and SessionID of the subtoken that has been manipulated inside the Constraint of the Token. Finally, the **fifth** field is the values of the manipulation. Depending on the type of the manipulation there will be one value (0D and 1D) or two values (2D).

**Table 9** TABW events to be filtered

| tokenName | type | manip | ID | SessionID | Value1 | Value2 |
|---|---|---|---|---|---|---|
| RTLS | 0D | add | RF Beacon | 0 | 0/1 | – |
| RTLS | 2D | move | RF Beacon | 0 | 0 to 1 | 0 to 1 |
| Tabletop1 | 0D | add | finger | num | 0/1 | – |
| Tabletop1 | 1D | rotate | finger | num | 0 to 1 | – |
| Tabletop1 | 2D | move | finger | num | 0 to 1 | 0 to 1 |
| Micro | 0D | add | 0 | 0 | 0/1 | – |
| Micro | 1D | move | 0 | 0 | 0 to 1 | – |

**Table 10** TAC list

| TOKEN | | | | | |
|---|---|---|---|---|---|
| Constraint | Constraint_values | Manipulation | ID | SessionID | Values |

The coding of the Host application consists of listening from OSC messages from the Broadcaster, parsing them, evolving the game according a game rules, responding to players composing OSC messages, and sending them to displays and actuators. In order to simplify the parsing and composing of OSC messages with the adequate format the toolkit provides developers with an API. This API is at the moment for the Proccesing programming environment, but it could be easily translated to other programing environments as far it enables OSC implementation.

The Functions of the API are classified in three groups:

1. Functions to **consult the TAC list of each Token**: the API offers different functions that allow to:

- **Consult the status of the game**: the developer can always know the status of the elements that are involved in the pervasive game at any moment.
- **Event of last TAC that has been instantiated in the system**: A callback function can be implemented to be triggered each time a TAC has changed, witch informs of the last manipulation that has taken place in the IS.

The API also provides functions in order to consult the specific attributes of the TAC that have been explained in the Table 8 of section 4.4.1.

2. Functions to **send commands to actuators**: these commands will cause the actuator to react physically. For example, the functions can lit a LED in a specific color.
3. Functions to **send graphic and sound commands to displays**: these commands are the ones explained in the Table 7 of section 4.2.

### 4.3.8 Example of use: Implementation of "The Augmented Bedroom World" with the API

In TABW, the list of TACs that need to be consulted are shown in Table 11, since we need to know: when the child steps on the rug (TAC1), when the child steps on the bed (TAC2), when the child places his/her hand on the tabletop (TAC3) and when the child is shouting the magic word (TAC4).

Next, in Fig. 18 part of the code in Processing language for the TABW Host application is presented. In TABW we make use of the API functions (1), that allow the developer to know the last TAC that has been instantiated, and the API functions (3) in order to draw the background and reproduce sounds.

Finally, Fig. 19 shows different moments of TABW game: when the player steps on the player steps on the rug and when she places a hand on the tabletop.

After explaining in detail the architecture of the toolkit, in the next section some cases studies are presented.

**Table 11** TAC list of TAWB

| Num | Constraint | Constraint_values | Manipulation | ID | SessionID | Values |
|---|---|---|---|---|---|---|
| RTLS | | | | | | |
| 1 | rug-area | [rug-vertex] | add | RF beacon | 0 | 0/1 |
| 2 | bed-area | [bed-vertex] | add | RF beacon | 0 | 0/1 |
| TABLETOP | | | | | | |
| 3 | surface | [surface-vertex] | add | finger | 0 | 0/1 |
| MICROPHONE | | | | | | |
| 4 | shout-range | 0.7–1 | move | 0 | 0 | 0 to 1 |

## 5 Cases studies: Exploring toolkit expressivity and threshold

During the last two years, the JUGUEMOS toolkit has been employed in several projects with students, which has helped us to obtain valuable feedback about its expressivity and threshold. Three of them will be presented in this section. Two of them have used the JUGUEMOS IS while the third one was carried out in the Aragon School of Design (ESDA).

### 5.1 Case study 1: "The Indian World" project

The first case study describes the development of a pervasive game called "The Indian World", aimed to be played in JUGUEMOS IS and devised by two students of the Industrial Design Engineering degree at the University of Zaragoza as part of their final degree project. With this first case study we wanted to explore the expressivity of the toolkit by seeing if it allowed to develop other people's ideas, since up till this point we had just developed games in order to

```
void eventoTAC(TAC lastTAC) {
    // child is on the rug
    if (lastTAC.getTokenName().equals("RTLS") && lastTAC.getConstraintName().equals("rug")
    && lastTAC.getSubtokenName().equals("child") && lastTAC.getManipulation().equals("add")) {
        if (lastTAC.values.get(0) ==1){
           displaysClient.playAudio("rugVoice.mp3");
        }
    }
    // child places the hand on the tabletop
    else if (lastTAC.getTokenName().equals("tabletop")  &&
    lastTAC.getConstraintName().equals("surface") && lastTAC.getSubtokenName().equals("finger"))
    && lastTAC.getManipulation().equals("add") {
        if (lastTAC.values.get(0) ==1) displaysClient.playAudio("tabletopVoice.mp3");
   }
   // child is on the bed
    else if (lastTAC.getTokenName().equals("RTLS") && lastTAC.getConstraintName().equals("bed")
   && lastTAC.getSubtokenName().equals("child") && lastTAC.getManipulation().equals("add")) {
        if (lastTAC.values.get(0) ==1)  childInBed = true;
  }
  // child has shouted while being on the bed
    else if (lastTAC.getTokenName().equals("micro")   &&   lastTAC.getManipulation().equals("move")
    && lastTAC.values.get(0) > 0.7) {
        if (childInBed)  displaysClient.playAudio("endGame.mp3");
     }
}
```

**Fig. 18** Processing code to treat the events for TABW

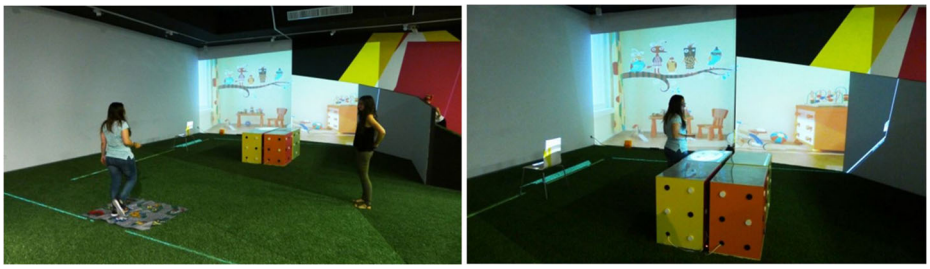


**Fig. 19** "The Augmented Bedroom World" game

test the functionalities of the toolkit. In this case, the students designed the game concept and all the graphical resources (images, animations, and sounds). Figure 20 shows the configuration of JUGUEMOS IS at the moment of creating "The Indian World" and the scheme of the game's world.

In this game, the protagonist, Drippy, meets an Indian Chief who has a key that Drippy needs to travel to other worlds. In order to recover the key, children have to play drums of different colors with drumsticks by following the sequence that the Indian Chief performs. After doing so, the Indian throws the key to the top of a totem that is also composed of different colors. Children have to play drums to destroy the corresponding colors of the totem: for example, if the first totem piece is yellow, the child with the yellow drum must play it. After the totem is destroyed the key falls to the ground, a light appears over it and then children can grab it to approach it to the keyhole that has just appeared to end the game.

The correspondences of the different elements that compose the game concept and the technologies involved in the IS are presented next:

- The different animations are displayed on the three projection walls (see Fig. 20-4) and on the four tabletop surfaces (see Fig. 20-1). This makes a total of seven displays organized in a unique Virtual Space. Also, the sound effects, such as the voices of Drippy and the Indian Chief and the sound of the drumsticks connecting with the drum, are reproduced through the different speakers of the room and the ones inside the tabletops.
- The drums are the tabletop devices. Each tabletop displays a different color on its surface (red, green, yellow and blue) and each child "controls" one tabletop. In order to interact with it, children use drumsticks with small white foam balls attached to their tip, so that reacTIVision can detect when they hit the tabletop's surface as if they were actually playing a drum (see Fig. 21a)

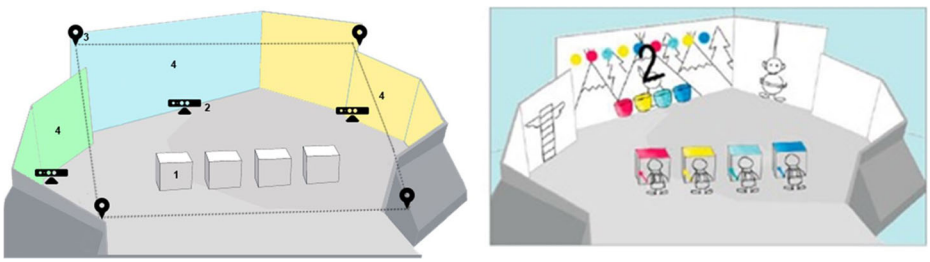



**Fig. 20** Scheme of the Interactive Space and "The Indian World". 1: Four tabletops. 2: Kinect sensors. 3: RTLS. 4: Projection screens

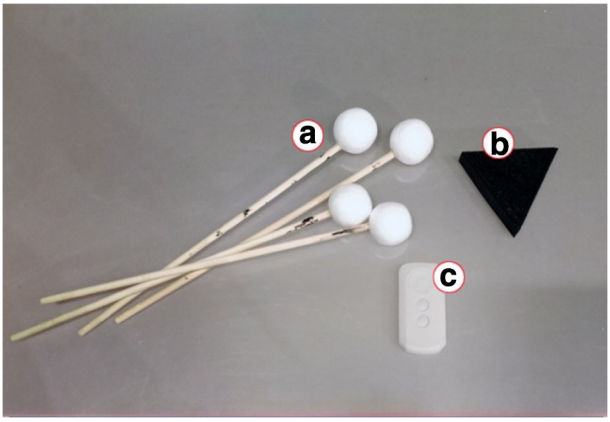



**Fig. 21** Indian World's toys

- The light that appears over the key when children finish breaking the totem is a LED spotlight placed in the corner of the projection screen, which illuminates the physical key that has been on the floor during the game. The LED turns on thanks its connection to an Arduino actuator that has its own Publisher process connected to the Broadcaster.
- The key is a 3D printed object (see Fig. 21b) which contains a RTLS tag inside (see Fig. 21c), to be tracked when children approach it to the keyhole.

The developer (belonging to the research team) had to translate these concepts into TUIML, in terms of tokens and constraints. The Graphic Assistant was used to model a constraint in each tabletop surface, and to define the area of the floor where children would need to approach with the key in order to open the door to the next world. Therefore, in this case the events to be taken into consideration are:

- The drumsticks' tip is placed on the tabletop (when children "play" the drums).
- A child enters an area situated next to the front projection screen.

Thanks to the API functions the codification of the game logic is quite simple (see Fig.22).

Figure 23 shows the Indian World running in the IS, with children interacting with the tabletops.

### 5.2 Case study 2: Collaborative experience between designers and developers

As mentioned before, up till this point the games presented had been developed by the authors, so with this second case study we wanted to see if the toolkit was simple enough to be used by external developers (threshold). In addition we wanted to explore the capability of the toolkit to support collaborative multidisciplinary (designers/developers) experiences.

The multidisciplinary team consisted of 5 Graphic Design students of the 4th year at the School of Arts in the University of Plymouth (UK) and 6 computer engineering students of the 4th year at the School of Engineering and Architecture (EINA) of the University of Zaragoza (Spain). The graphic design students had previous skills on generating multimedia content for multimedia applications, but no experience in the design of interactive applications or

```
void eventoTAC(TAC lastTAC) {
  // tabletop1 has been touched
  if (lastTAC.getTokenName().equals("tabletop1") && lastTAC.getConstraintName().equals("surface")
  && lastTAC.getSubtokenName().equals("finger") && lastTAC.getManipulation().equals("add")) {
    if (lastTAC.values.get(0) ==1){
          // stage one of the game: follow sequence
     if (gameStatus==1) {
           if ((sequence[current]==”red”) {
            current++;
           }
     }
     else if (gameStatus==2) {
           if ((totem[current]==”red”) {
            // the child hasto hit the table four times to break the totem section
            golpes++;
            if (golpes==4) {
             current++;
             golpes=0;
            }
           } // totem=red
    } // gameStatus=2
   } // add =1
  } // lastTac
  …
  //The code for the rest of the Tabletops is the same, but consulting if sequence[current] is “green”,
  // “blue” or “yellow”


  // key is next to the keyhole
  else if (lastTAC.getTokenName().equals("RTLS") && lastTAC.getConstraintName().equals("keyhole")
  && lastTAC.getSubtokenName().equals("143") && lastTAC.getManipulation().equals("add")) {
    if (lastTAC.values.get(0) ==1){
     endgame= true;
   }
  }
}
```

**Fig. 22** Processing code to treat the events for the Indian World

pervasive games. The Computer Engineering students were all taking a User Center Design course, which gives them skills to develop interactive multimedia applications, but they had no previous experience on ubiquitous technologies or on the prototyping of pervasive games.

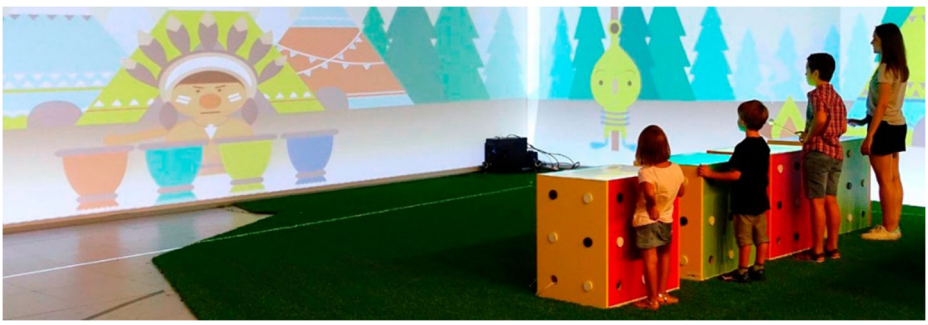


**Fig. 23** The Indians World running the Interactive Space

Before working together, the 6 computer engineering students attended a 3 h practical session in the IS so that they could familiarize with the ubiquitous technologies involved in the IS and JUGUEMOS toolkit. The configuration of the IS at the moment was the same as the previous case study (4 tabletops, RTLS sensor, and 3 projection screens) but this time it also included Kinect sensors. The session was completely practical: each student had to follow a guided exercise in which they had to complete the Butterflies game (section 4.1) by using the Processing Development environment and the JUGUEMOS API. Students were provided with the official Processing documentation and with the JUGUEMOS toolkit documentation. Also, we organised another 2 h session with the 5 graphic design students from the School of Arts of Plymouth, where the students were introduced to the physical affordances of the ubiquitous technologies currently integrated in the IS, in order to bring physical interaction to pervasive games. We also showed them different examples of pervasive games developed for our IS.

After those two different sessions, both computer engineers and design students worked together. The eleven participants were arranged in two groups of 5 and 6 people respectively: 2 designers and 3 developers in one group, and 3 designers and 3 developers in another group. The designers were the ones in charge of thinking about the game concept, to choose the interaction paradigms that were going to be used, and to create the graphic resources of the games. Accordingly, the developers listened to the designers' ideas to decide if they could be developed in the IS and within the three hours of the fourth session. The ideas that the two groups came up with were.

- **Car races (group 1)**: In this game, players use the tabletops to drive their respective cars with two different objects that control the car's speed and direction (see Fig. 24 Left). The designers created the necessary graphical resources for the projection in screens and the tabletops with the technical support of the developers to choose the images resolutions and formats supported by the toolkit, a knowledge that the developers had acquired in the first session.
- **Building a car (group 2)**: The idea was to make a construction game. Players use their hands to "grab" the car pieces that appeared on the projection screen to "put" them on their matching place of the car shape (see Fig.24 Right).

In the "Car races" game, the constraints that needed to be defined with the Graphic Assistant had to cover the whole tabletop surface, since the controls could be placed in any position on the tabletop. However, in the "Building a car" game developers had to define four

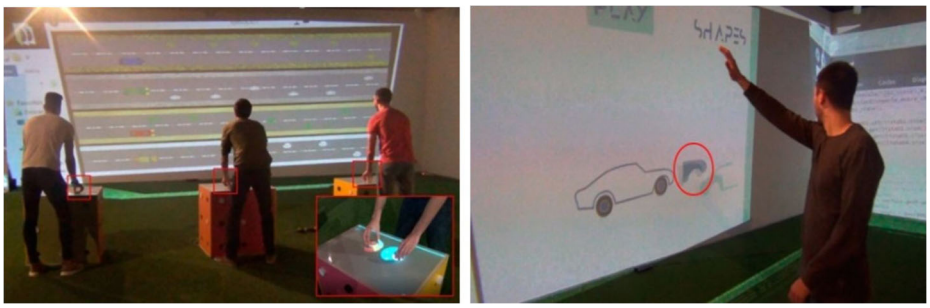


**Fig. 24** Pervasive games created. Left: "Car races". Right: "Building a car"

different constraints corresponding to the four pieces that needed to be placed on the car shape, so that the Semantic Level could filter when the user's hand entered those areas.

Regarding the game logic, developers of the "Car races" game had to consult the messages with the orientation of each controller (rotate manipulation) and translate this data into two different values: one value corresponds to the rail where the car moves (upper, middle, or lower rail), and the other to the speed of the car (slow, normal or fast) (see Fig.25). In the "Building a car" game developers had to consider the messages that came from the Kinect that contained the (x, y) position of the player's hand (move manipulation), to differentiate which car piece the player was selecting with his/her hand (see Fig.26).

Thanks to the toolkit all the developers were able to develop a fully functional prototype based on the ideas provided by the designers in a three hours session. However, this was possible because the ideas presented by the designers were supported by the IS existing hardware: if different physical interactions had been proposed, the complexity would have notably increased since it would have been necessary to carry out the integration of new devices together with the programming of their corresponding Publishers. Nevertheless, regarding the coding of the game, which is independent of the devices installed in the IS, the simplicity of the generated code indicates that it is to be expected that new design iterations can be carried out quite fast, which helps to lower the gap between designers and developers.

### 5.3 Case study 3: Activity in the Aragon School of Design (ESDA)

It was made possible thanks to a collaboration between the ESDA and our research group. The collaboration opened the opportunity for their students to explore the possibilities of advanced interactive installations and for us to continue to explore the expressivity of our toolkit and, besides, its capability to support interactive experiences outside our IS.

```
void eventoTAC(TAC lastTAC) {
  // tabletop1 has been touched
  if (lastTAC.getTokenName().equals("tabletop1") && lastTAC.getConstraintName().equals("surface")
  && lastTAC.getManipulation().equals("rotate")) {
    if (lastTAC.getSubtokenName().equals("control_velocity_id")) {
          // checking if the car goes slow, normal or fast
      double angle=360-degrees(lastTac.getValores().get(0));
      if (angle >= 90 && angle <= 135) veloc=6;                    //fast
          else if (angle > 135 && angle < 220) veloc=4;            //normal
          else if (angle > 220 && angle <= 270) veloc=2;           //slow
    } // velocity
    else if (lastTAC.getSubtokenName().equals("control_rail_id")) {
          // checking if the car is in the upper or the lowe rail
      double angle=360-degrees(lastTac.getValores().get(0));
      if (angle >= 90 && angle <= 135) rail=1;                     //upper
          else if (angle > 135 && angle < 220) rail=2;  //middle
          else if (angle > 220 && angle <= 270) rail=3;            //lower
    } // position
  } // lastTac
  …
  //The code for the rest of the Tabletops would be the same but changing the parameter of the
  // getTokenName() function:" tabletop2", "tabletop3", "tabletop4"
}
```

**Fig. 25** Processing code to treat the events for the "Car Races" game

```
void eventoTAC(TAC lastTAC) {
  // tabletop1 has been touched
  if (lastTAC.getTokenName().equals("Kinect") && lastTAC.getSubtokenName().equals("rightHand")
  && lastTAC.getManipulation().equals("move")) {
    if (lastTAC.getConstraintName().equals("wheel1")) {
          // checking if the car goes slow, normal or fast
      double x=lastTac.getValores().get(0);
          double y=lastTac.getValores().get(1);
      drawPiece(x,y);
    } // velocity
  } // lastTac
  ...
  //The code for the rest of the car pieces would be the same but changing the parameter of the
  // getSubtokenName() function: "wheel2", "door", "carHood"
}
```

**Fig. 26** Processing code to treat the events for the "Building a car" game

First, the ESDA organised a contest opened to all students, inviting to submit ideas for an interactive art installation suitable to be set up in the school hall. No indications were given except that the installation should transmit the idea of "Innovation". Five different concepts of interactive installations were submitted. An external team of professional designers judged the ideas from a pure design perspective (technical aspects were not judged), and one was chosen for its graphic approach to the "innovation" concept.

The idea consisted of a room where visitors can enter to be reflected on the wall as a "cartoon" character. The virtual reflection mimics the visitor movements (see Fig. 27 Left). In addition, at the entrance of the room, a tactile screen lets visitors choose the virtual avatar they want to be reflected in the wall (see Fig. 27 Right).

A group of volunteer students and teachers from the different disciplines of the ESDA was formed with the aim of designing and implementing the final interactive installation; also, a research member of our group was involved to be in charge of the coding of the installation logic.

Several work sessions took place in our interactive space (see Fig. 28 Left). Our toolkit enabled us to quickly prototype the initial concept (see Fig. 28 Right). The tracking of the users' position was provided by a MS Kinect sensor. Two different displays were required: the wall display and the tactile screen. These last ones were solved using a 10′ android Tablet.

TUIML was used to model the different manipulations of the user's body parts (head, hands, feet …), and the different manipulations of touch actions on the tablet screen. This last

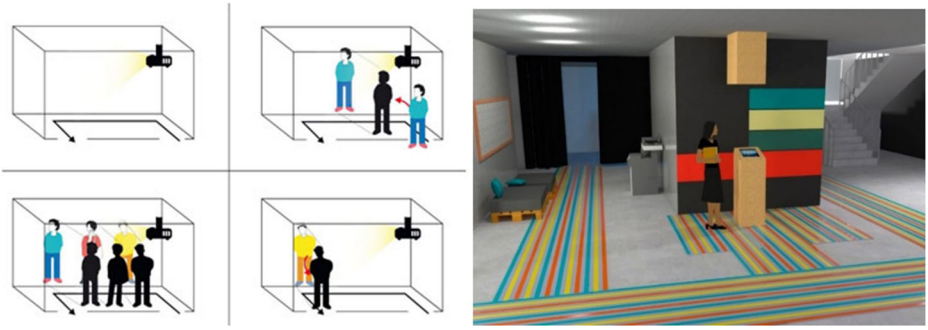


**Fig. 27** Art concept of the installation. Left: Script. Right: Totem with tactile screen

**Fig. 28** A work session in our interactive space. Left: Teachers and students from different design disciplines. Right: Students testing an early prototype of the art installation

one demanded to define several constraints associated with the buttons and sliders included in the graphic interface to be displayed on the tablet screen, which would enable users to choose a virtual avatar (see Fig. 29).

When the initial prototype was finished, the group was divided in different work teams depending on their design disciplines:

- Graphic designers were in charge of creating a collection of virtual avatars, and the graphic interface to be displayed on the tablet.
- Interior designers were in charge of building a dark room in the ESDA hall, and installing the hardware elements: video projector, Kinect sensor, and computers.
- The authors were in charge of implementing the game with the toolkit (see Fig. 30).
- Product designers were in charge of building a totem to accommodate the 10′ Android Tablet (see Fig. 31 Left).

Interconnectivity of the different elements of the installation (projector display, Android tablet, Kinect sensor, Broadcaster, and Host) was provided by installing a conventional Wi-Fi router. Finally, the interactive art installation could be enjoyed by ESDA students, staff and other visitors during a Design Week organized at the ESDA (see Fig. 31 Right).

The students that took part in its development appreciated the experience. One of the students highlighted that the activity showed her a new application of design through new

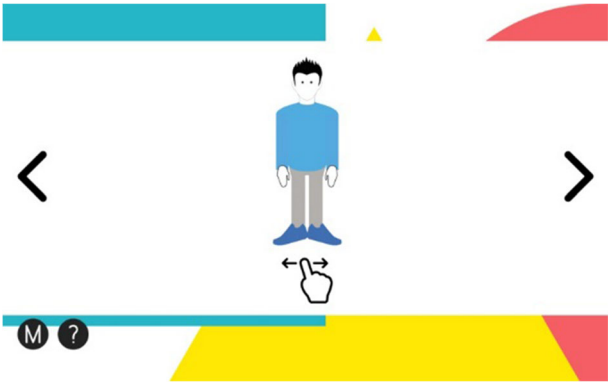



**Fig. 29** Graphic Interface for the tablet screen. Each button and slider controller required to define a constraint to fingers subtokens in TUIML

```
void eventoTAC(TAC lastTAC) {
  // tablet has been touched
  if (lastTAC.getTokenName().equals("tablet") && lastTAC.getSubtokenName().equals("finger")
  && lastTAC.getManipulation().equals("add")) {
    if (lastTAC.getConstraintName().equals("rightArrow")) // user has touched the right arrow to select
    the avatar  }
   }
  if (lastTAC.getTokenName().equals("tablet") && lastTAC.getSubtokenName().equals("finger")
  && lastTAC.getManipulation().equals("add")) {
    if (lastTAC.getConstraintName().equals("characterArea"))  // user has selected an avatar
   }
  … // events for each button of the tablet application
  // user is moving
  if (lastTAC.getTokenName().equals("Kinect") && lastTAC.getSubtokenName().equals("rightHand")
  && lastTAC.getManipulation().equals("move")) {
     // checking if the car goes slow, normal or fast
    double x=lastTac.getValores().get(0);
    double y=lastTac.getValores().get(1);
    drawBodyPart("rightHand", x,y);
  } // lastTac
  …
 //The code for the rest of the body parts would be the same but changing the parameter of the
 // getSubtokenName() and drawBodyPart() function by other body parts
}
```

**Fig. 30** Processing code to treat the events for the ESDA game

technologies. Also, she expressed her interest to learn more about coding for her future design career.

## 6 Discussion

When developing hybrid games for interactive spaces, the usual quick cycles of designing, implementing a prototype and testing become very hard to complete due to the complexity of implementing functional prototypes. This way, the gap between developers and designers gets bigger and gaming experiences are not tested till the game has been almost completely developed. The work presented here wants to contribute to solve these problems proposing a toolkit, aimed at developers, that facilitates the creation of functional prototypes. A strong point of the toolkit is to facilitate the integration of new devices that may support different

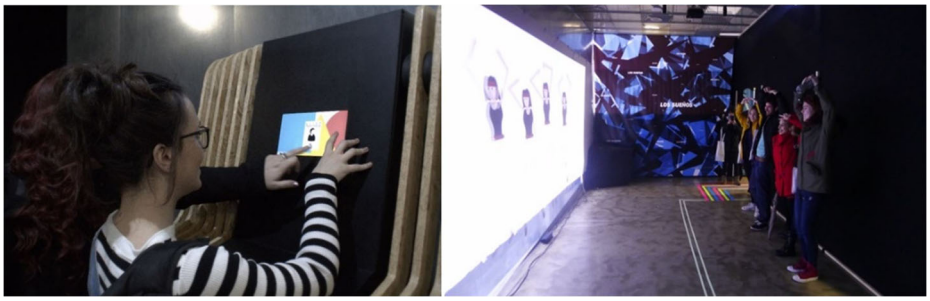


**Fig. 31** ESDA activity. Left: Totem designed to accommodate the Android tactile screen. Right: Interactive installation in the ESDA hall

types of interaction, without a great implementation impact. This is achieved by using TUIML as the common design framework, a Graphic Assistant that makes the devices integration easier and an API that provides a unified way to consult the device events that take place during the gameplay, facilitating the coding of the game logic. This way, the threshold of developing pervasive games in complex Interactive Spaces is lowered.

Moreover, the possibility of integrating diverse devices, including diverse displays, has been able to raise the ceiling of expressiveness of the toolkit. The two first case studies presented illustrate both aspects (threshold and ceiling). They also show that even though the toolkit is oriented to developers, it facilitates the work of multidisciplinary teams decreasing the gap between the two collectives.

Besides, the toolkit does not impose any minimum configuration to work except the necessity of implementing an OSC client in order to connect the different devices, since one of the main objectives when developing it was to deal with heterogeneity of devices. Also, since our toolkit is software based, the hardware requirements are exclusively imposed by the devices involved. Games that make use of a Kinect sensor, such as "Building a car" (Section 5.2), just require to be connected to a PC computer with Windows 10 and USB3.0. The same way, simple sensors such as the microphone used in the "The Augmented Bedroom World" game or the LED spotlight of "The Indian World" project (Section 5.1) can be connected to very low-spec hardware, such as an Arduino microcontroller. However, the RTLS system used in games like "Butterflies" (Section 4.1) and "The Augmented Bedroom World" (Section 4.3.1) requires a high broadband network router and a medium to high desktop PC computer. In summary, configurations with simple hardware will allow the development of simple games, while more complex configurations needing several sensors and displays working at the same time will require better hardware connected to a better network, allowing that way more complex games.

This also applies to possible delays in the interaction with the user. They will depend on the sensors used, on the features of the computer chosen to host the Broadcaster, and, especially, on the characteristics of the router: high-spec routers will reduce the delay of the interchange of OSC packets between the Broadcaster and the different devices, while low-spec routers may cause a delay in the transmissions. Also, the game complexity is a factor to take into consideration: simple games will probably require less OSC packet traffic than more complex games. In the experiences shown in the paper, we noticed some delay when working with the RTLS system due to the implementation of the sensor itself, which was independent of our software toolkit, and also occasional 'blinks' when trying to send too many Draw commands to the display Painters. However, these delays have never significantly affected user interaction, even in the experiences we have carried out using a low-spec router, such as the one taking place in the Aragon School of Design (Section 5.3).

Finally, although the toolkit has grown in parallel with a specific Interactive Space, it is not restricted to it. It is true that the two case studies have been carried out in the Interactive Space, a specific controlled environment. But the third case study shows that the toolkit infrastructure can be translated to more "in-the-wild" experiences: as long as the toolkit has access to a Wi-Fi network, it will be able to connect with the devices involved in the pervasive game.

However, due to the intrinsic complexity of developing this kind of games, the toolkit has some limitations that we are going to comment next.

The choice of using TUIML as the abstraction model implies a few restrictions. TUIML is based on establishing relations between constraints and objects that are manipulated inside those constraints, such as the "add", "move" and "rotate" manipulations associated to the 2D

and 3D physical space. However, TUIML fails short when trying to represent other kinds of manipulations that do not meet this definition. For instance, in a game that required using shape changing objects, made of clay for example, the interactions carried out with the objects could not be represented with the TAC hierarchy, since in this case a token is being manipulated itself, not inside a physical restriction imposed by another token. A more thorough analysis about how TUIML could address this limitation remains to be done, but still we believe that the range of games that can be designed is considerable, as we have tried to prove along the whole article. Moreover, we have adopted the static specification of TUIML that Shaer and Jacob [39] defined, without addressing the dynamic specification in which Tokens, Constraints and Subtokens could change as the game evolves. In this aspect, our toolkit is limited because the TUIML defined by the Graphic Assistant is static, which makes it suitable for games that do not require a very variable game logic during the gameplay. In case it was necessary to consider changes in the gameplay, it would be necessary to divide the game in stages associated to different TUIMLs. This is another issue that will have to be tackled in the future.

Another limitation is that, currently, our toolkit just allows dealing with devices whose data can be transformed into a 2D-dimension, without addressing 3D-devices. This is due to the fact that, until now, the applications that we needed to develop could be carried out by remaining just in the two dimensions, so a new version of the toolkit able to support 3D-data remains to be developed. In order to do this, it would be necessary to update the Graphic Assistant so that it could provide developers with a 3D-virtual space that would allow them to create 3D-constraints (volumes). Also, if 3D-displays were involved, such as holographic displays, the Virtual Space paradigm would need to be updated, allowing placing the different physical displays in a 3D-space.

Another key issue is that our toolkit is aimed at developers, not designers. It would be necessary to simplify even more the process of creating games so that eventually designers could also use the toolkit without the support of developers. This represents a challenge due to the complexity of Interactive Spaces.

Finally, in spite of the use cases and studies carried out the last two years, more lengthy and formal experiences remain to be completed.

## 7 Conclusions and future work

In this paper we have presented JUGUEMOS, a framework that aims to facilitate the creation of pervasive games for Interactive Spaces. This goal was motivated by an analysis of pervasive games and frameworks, which revealed the three challenges that arise when developing these games and the lack of frameworks when meeting all those challenges.

The JUGUEMOS toolkit is based on a centralized network architecture that allows an easy integration of new devices with the OSC Protocol and Processing, and that adopts the Virtual Space display paradigm to deal with the management of different display devices. The toolkit is based on the TUIML model in order to define pervasive games, and provides developers with a WYSIWYG Graphic Assistant, in charge of creating the game file that will be interpreted by the toolkit, and an API to easily access the information related to the devices connected in the IS. We have presented cases of use that aim to illustrate the different stages that composed the development of the JUGUEMOS toolkit, and also three different case studies to show how our toolkit is actually used. Finally, we discussed the limitations that our toolkit currently has. However,

although it is still necessary to improve some aspects of our toolkit, we believe that we have managed to provide developers with a tool that facilitates the creation of pervasive games, and opens the door to interesting design experiences. In fact, further experiences with designers and developers of interactive applications are planned in the near future.

**Acknowledgments** We want to thank Belén Cebrián, Alejandro Navarro, the students and tutors of the School of Arts of Plymouth, the students and tutors of the EINA, the ESDA staff, and the Cesar-Etopía laboratories, for making this work possible. This work has been partly financed by the Spanish Government and the European Union through the contract TIN2015-67149-C3-1R (MINECO/ FEDER) and by the Aragonese Government and the European Union through the FEDER 2014-2020 "Construyendo Europa desde Aragón" action (Group T25_17D).

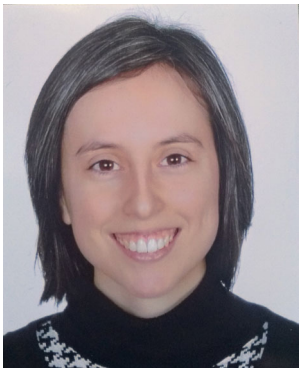
## References

1. Arango J, Gallardo J, Gutiérrez FL, Collazos CA, Amengual E, Valera R, Cerezo E (2017) Pervasive games: giving a meaning based on the player experience. In: proceedings of the XVIII international conference on human computer interaction, p 9, ACM
2. Bobick AF, Intille SS, Davis JW, Baird F, Pinhanez CS, Campbell LW et al (1999) The KidsRoom: a perceptually-based interactive and immersive story environment. Presence 8(4):369–393
3. Borchers J (2006) The Aachen media space: multiple displays in collaborative interactive environments. In Workshop Information Visualization and Interaction Techniques for Collaboration across Multiple Displays in conjunction with CHI (Vol. 6).
4. Cerezo E, Marco J, Baldassarri S (2015) Hybrid games: designing tangible interfaces for very young children and children with special needs. In: More playful user interfaces. Springer, Singapore, pp 17–48
5. Chalmers M, Bell M, Brown B, Hall M, Sherwood S, Tennent P (2005). Gaming on the edge: using seams in ubicomp games. In proceedings of the 2005 ACM SIGCHI international conference on advances in computer entertainment technology, pp 306-309, ACM
6. Cheok AD, Yang X, Ying ZZ, Billinghurst M, Kato H (2002) Touch-space: mixed reality game space based on ubiquitous, tangible, and social computing. Pers Ubiquit Comput 6(5–6):430–442
7. Dionisio M, Gujaran A, Pinto M, Esteves A (2015) Fall of humans: interactive tabletop games and transmedia storytelling. In proceedings of the 2015 international conference on Interactive Tabletops & Surfaces, pp 401-404, ACM
8. Flintham M, Benford S, Anastasi R, Hemmings T, Crabtree A, Greenhalgh C, et al. (2003) Where on-line meets on the streets: experiences with mobile mixed reality games. In proceedings of the SIGCHI conference on human factors in computing systems, pp 569-576, ACM
9. Gatti E, Pittera D, Moya JB, Obrist M (2017). Haptic rules! Augmenting the gaming experience in traditional games: The case of foosball. In World Haptics Conference (WHC), 2017 IEEE, pp 430-435, IEEE
10. Guo B, Satake S, Imai M (2008) Lowering the barriers to participation in the development of human-artifact interaction systems. International Journal of Semantic Computing 2(04):469–502
11. Guo B, Fujimura R, Zhang D, Imai M (2012) Design-in-play: improving the variability of indoor pervasive games. Multimed Tools Appl 59(1):259–277. https://doi.org/10.1007/s11042-010-0711-z
12. Hornecker E, Buur J (2006) Getting a grip on tangible interaction: a framework on physical space and social interaction. In proceedings of the SIGCHI conference on human factors in computing systems, pp 437-446, ACM
13. Ishii H, Wisneski C, Orbanes J, Chun B, Paradiso J (1999) PingPongPlus: design of an athletic-tangible interface for computer-supported cooperative play. In proceedings of the SIGCHI conference on human factors in computing systems, pp 394-401, ACM
14. Jetter HC, Reiterer H, Geyer F (2014) Blended interaction: understanding natural human–computer interaction in post-WIMP interactive spaces. Pers Ubiquit Comput 18(5):1139–1158
15. Kaltenbrunner M, Bencina R (2007) reacTIVision: a computer-vision framework for table-based tangible interaction. In proceedings of the 1st international conference on tangible and embedded interaction, pp 69-74, ACM
16. Kaltenbrunner M, Bencina R (2007) reacTIVision: a computer-vision framework for table-based tangible interaction. In proceedings of the 1st international conference on tangible and embedded interaction, pp 69-74, ACM
17. Kato H (2007) Inside ARToolKit. In 1st IEEE International Workshop on Augmented Reality Toolkit.
18. Khoo ET, Cheok AD (2008) Age invaders: inter-generational mixed reality family game. Int J Virtual Real 5(2):45–50

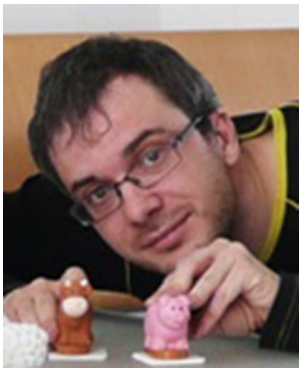
19. KinectOSC: https://github.com/microcosm/KinectV2-OSC Accessed 3 October 2018
20. Krzywinski A, Mi H, Chen W, Sugimoto M (2009). RoboTable: a tabletop framework for tangible interaction with robots in a mixed reality. In proceedings of the international conference on advances in computer Enterntainment technology, pp 107-114, ACM
21. Lahlou S (Ed.) (2009) Designing user friendly augmented work environments. Springer-Verlag London
22. Magerkurth C, Cheok AD, Mandryk RL, Nilsen T (2005) Pervasive games: bringing computer entertainment back to the real world. Computers in Entertainment (CIE) 3(3):4–4
23. Magerkurth C, Engelke T, Grollman D (2006). A component based architecture for distributed, pervasive gaming applications. In proceedings of the 2006 ACM SIGCHI international conference on advances in computer entertainment technology, p 15, ACM
24. Magielse R, Markopoulos P (2009) HeartBeat: an outdoor pervasive game for children. In proceedings of the SIGCHI conference on human factors in computing systems, pp 2181-2184, ACM
25. Malone TW (1981) Toward a theory of intrinsically motivating instruction. Cogn Sci 5(4):333–369
26. Marco J, Cerezo E, Baldassarri S (2012). ToyVision: a toolkit for prototyping tabletop tangible games. In proceedings of the 4th ACM SIGCHI symposium on engineering interactive computing systems, pp 71-80, ACM
27. Martins T, Sommerer C, Mignonneau L, Correia N (2009). Noon: a secret told by objects. In proceedings of the international conference on advances in computer Enterntainment technology, pp 446-446, ACM
28. Montola M (2005) Exploring the edge of the magic circle: defining pervasive games. In: Proceedings of DAC, pp 16–19
29. Montola M, Stenros J, Waern A (2009) Pervasive games: theory and design. Morgan Kaufmann Publishers Inc.
30. Mora S, Fagerbekk T, Di Loreto I, Divitini M (2015) Making interactive board games to learn: reflections on AnyBoard. In Make2Learn@ ICEC, pp 21-26
31. Morreale F, De Angeli A, Masu R, Rota P, Conci N (2014) Collaborative creativity: the music room. Pers Ubiquit Comput 18(5):1187–1199
32. Muse band: http://www.choosemuse.com/ Accessed 3 October 2018
33. Myers B, Hudson SE, Pausch R (2000) Past, present, and future of user interface software tools. ACM Transactions on Computer-Human Interaction (TOCHI) 7(1):3–28
34. Nacenta MA, Aliakseyeu D, Subramanian S, Gutwin C (2005). A comparison of techniques for multi-display reaching. In proceedings of the SIGCHI conference on human factors in computing systems, pp 371-380, ACM
35. Park JW (2017) Hybrid monopoly: a multimedia board game that supports bidirectional communication between a Mobile device and a physical game set. Multimed Tools Appl 76(16):17385–17401. https://doi.org/10.1007/s11042-017-4589-x
36. Processing web: http://www.processing.org Accessed 3 October 2018
37. Pure Data web: https://puredata.info/ Accessed 3 October 2018
38. Satyanarayanan M (2001) Pervasive computing: vision and challenges. IEEE Pers Commun 8(4):10–17
39. Shaer O, Jacob RJ (2009) A specification paradigm for the design and implementation of tangible user interfaces. ACM Transactions on Computer-Human Interaction (TOCHI) 16(4):20
40. Shaer O, Leland N, Calvillo-Gamez EH, Jacob RJ (2004) The TAC paradigm: specifying tangible user interfaces. Pers Ubiquit Comput 8(5):359–369
41. Soute I, Lagerström S, Markopoulos P (2013). Rapid prototyping of outdoor games for children in an iterative design process. In proceedings of the 12th international conference on interaction design and children, pp 74-83, ACM
42. Soute I, Vacaretu T, Wit JD, Markopoulos P (2017) Design and evaluation of RaPIDO, a platform for rapid prototyping of interactive outdoor games. ACM Transactions on Computer-Human Interaction (TOCHI) 24(4):28
43. Tanenbaum J, Tanenbaum K, Antle A (2010) The Reading glove: designing interactions for object-based tangible storytelling. In proceedings of the 1st augmented human international conference, p 19, ACM
44. Ullmer B, Ishii H (2000) Emerging frameworks for tangible user interfaces. IBM systems journal 39(3.4): 915–931
45. Ullmer B, Ishii H, Jacob RJ (2005) Token+ constraint systems for tangible interaction with digital information. ACM Transactions on Computer-Human Interaction (TOCHI) 12(1):81–118
46. Vvvv: multipurpose toolkit (2014) https://vvvv.org/ Las Accessed: April, 2018
47. Wright M (2005) Open sound control: an enabling technology for musical networking. Organised Sound 10(3):193–200
48. Yamabe T, Nakajima T (2013) Playful training with augmented reality games: case studies towards reality-oriented system design. Multimed Tools Appl 62(1):259–286. https://doi.org/10.1007/s11042-011-0979-7

49. Yanagida Y, Kawato S, Noma H, Tomono A, Tesutani N (2004) Projection based olfactory display with nose tracking. In virtual reality, 2004. Proceedings. IEEE, pp 43-50, IEEE
50. Zhu F, Sun W, Zhang C, Ricks R (2016) BoomChaCha: a rhythm-based, physical role-playing game that facilitates cooperation among players. In proceedings of the 2016 CHI conference extended abstracts on human factors in computing systems, pp 184-187, ACM





**Clara Bonillo** is a Ph.D. student in the Doctoral Program in Systems Engineering and Informatics, University of Zaragoza (Spain), in the AffectiveLab Group. Her research area is Tangible Interaction. In 2015, her final project entitled "Development of a tool for the design and running of activities for the elderly with the NIKVision tabletop" received the prize of the best Final Project from the Spanish Human Computer Interaction Association of which she is a member.



**Javier Marco** obtained a Ph.D. in Computer Science Engineering in 2011 at the University of Zaragoza (Spain). His thesis researched the benefits of tangible interfaces for children and their involvement during design and evaluation stages. He was a visiting researcher at the ChiCI Group at the University of Central Lancashire and at the M-ITI at the University of Madeira (Portugal), under a Carnegie Mellon post-doc program. He is in the scientific committee of the Interaccion International Conference and Tangible and Embedded Interaction International Conference. He also has organized several workshops in different international symposiums dealing the design and implementation of tangible applications.

**Eva Cerezo** received a Ph.D. degree in Computer Science in 2002. She is Associate Professor at the School of Engineering and Architecture and Head of the Computer Sciences and Systems Engineering Department at the University of Zaragoza. She leads the AffectiveLab, a research group that focuses on affective multimodal human computer interaction, tangible tabletops and virtual humans. She is author of more than 80 international publications. She is a member of the Executive Board of the ACM SIGCHI Spanish Local Chapter.

## Affiliations

**Clara Bonillo**[1] · **Javier Marco**[2] · **Eva Cerezo**[3]

[1] Advanced Interfaces Group (AffectiveLab), Computer Science Department, Universidad de Zaragoza, Ed. Ada Byron, C/María de Luna n. 1, 50015 Zaragoza, Spain

[2] Escuela Superior de Diseño de Aragón (ESDA), C/ María Zambrano, n. 3, 50018 Zaragoza, Spain

[3] Advanced Interfaces Group (AffectiveLab), Computer Science Department, Engineering Research Institute of Aragon (I3A), Universidad de Zaragoza, Ed. Ada Byron, C/María de Luna n. 1, 50015 Zaragoza, Spain