A game-based approach to the teaching of object-oriented programming languages

José María Rodríguez Corral [a,*], Antón Civit Balcells [b], Arturo Morgado Estévez [a], Gabriel Jiménez Moreno [b], María José Ferreiro Ramos [c]

[a] School of Engineering, University of Cádiz, Chile 1, 11002 Cádiz, Spain
[b] Technical School of Computer Engineering, University of Seville, Reina Mercedes s/n, 41012 Seville, Spain
[c] School of Marine, Nautical and Radioelectronic Engineering, University of Cádiz, República Saharaui s/n, 11510 Puerto Real, Cádiz, Spain

Keywords:

Human–computer interface Programming and programming languages Tangible user interfaces
Teaching/learning strategies
Wireless sensor networks

A B S T R A C T

Students often have difficulties when trying to understand the concepts of object-oriented programming (OOP). This paper presents a contribution to the teaching of OOP languages through a game-oriented approach based on the interaction with tangible user interfaces (TUIs). The use of a specific type of commercial distributed TUI (Sifteo cubes), in which several small physical devices have sensing, wireless communication and user-directed output capabilities, is applied to the teaching of the C# programming language, since the operation of these devices can be controlled by user programs written in C#. For our experiment, we selected a sample of students with a sufficient knowledge about procedural programming, which was divided into two groups: The first one had a standard introductory C# course, whereas the second one had an experimental C# course that included, in addition to the contents of the previous one, two demonstration programs that illustrated some OOP basic concepts using the TUI features. Finally, both groups completed two tests: a multiple-choice exam for evaluating the acquisition of basic OOP concepts and a C# programming exercise. The analysis of the results from the tests indicates that the group of students that attended the course including the TUI demos showed a higher interest level (i.e. they felt more motivated) during the course exposition than the one that attended the standard introductory C# course. Furthermore, the students from the experimental group achieved an overall better mark. Therefore, we can conclude that the technological contribution of Sifteo cubes – used as a distributed TUI by which OOP basic concepts are represented in a tangible and a visible way – to the teaching of the C# language has a positive influence on the learning of this language and such basic concepts.

1. Introduction

Currently, the computer is part of students' life and most of them have some experience in the use of computer games, search engines, social networks, text processors and instant messaging. However, students often have difficulties when trying to understand the abstract concepts of object-oriented programming (e.g. class, object, method, attribute, inheritance), since it is hard for them to find the equivalence of such concepts in real life (Yan, 2009). Also, programming is teached traditionally as the creation of text-based programs, which is neither familiar nor attractive to students. Most of them are more experienced in the use of the mouse and graphical environments (gestural interfaces) than in the use of the command line (textual interfaces).

Object-oriented design is very natural, since in real life we think in terms of objects, which have certain properties and behaviors. However, when writing programs, those students initiated in imperative programming tend to adopt the traditional view consisting of instructions that are executed and control structures that define the control flow of programs (Overmars, 2004). In this sense, programmers

expect computer programs to have a beginning and an end, and therefore one understands a program by reading through it. Nevertheless, understanding an object-oriented program means understanding what objects are and how messages are transferred among them in order to accomplish tasks (Rosson & Carroll, 1990).

There are successful learning experiences based on the creation and the use of computer games (Overmars, 2004; Yan, 2009). A computer game consists of objects (player and enemies that must be avoided or destroyed, walls and obstacles that block the way, things to be taken, weapons, …). Thus, thinking in terms of videogames means thinking in objects and how they react to one another and to the player's inputs.

In this work, we want to take abstract object-oriented programming (OOP) concepts to the "visible world" for students. The use of Sifteo cubes (Sifteo Inc., 2012a) allows to materialize basic OOP concepts as object, attribute and method, and so it stimulates significant learning (Fink, 2003) on students. First, we will develop some demonstration programs in order to illustrate some basic OOP concepts using the features of Sifteo cubes. Next, we will prepare two introductory OOP courses using the C# programming language: One of them will include the software demos previously referred, whereas the other one will be a standard course. Finally, we will design two tests: a multiple-choice exam for evaluating the acquisition of basic OOP concepts, and a programming exercise also based on multiple choices.

Once the two group of students – experimental and control – that take part in the experiment have completed the two exams, we will proceed to evaluate the results and to extract the corresponding conclusions, that will be shown in last sections.

2. Learning theory principles

In this section, we address some important learning theory principles that, in our opinion, underlie the learning process of a programming language, as *constructivism* and *reflective practice*.

When creating a constructivist learning opportunity or lesson, the created situation must have the property of stimulating the learner to solve a problem or take some action, and then to reflect on what has happened, connecting the situated experience to prior knowledge (Lesgold, 2004). In this sense, learning a programming language can be though as a construction process, where the student gradually builds new knowledge structures that connects to his prior knowledge.

For example, when learning an imperative language (Rodríguez, García & Morgado, 2006), once the student has learned the first fundamentals (i.e. constants, variables, basic data types and basic input/output statements), then he or she is ready to study the language structures that controls the program execution flow (e.g. "if", "while", "for") and integrate this new knowledge with that previously achieved. Thus, each new learning is like a piece that must be correctly placed over the previous ones in order to build a logical and integrated comprehensive knowledge structure.

Also, in any learning process, the reflective practice is essential, as it enables the person to learn from his experiences. Kolb developed a theory of experiential learning (Kolb, 1984) called the *Learning Cycle* (Fig. 1). Such cycle comprises four stages: *Concrete Experience*, *Reflective Observation*, *Abstract Conceptualization* and *Active Experimentation*. It can be entered at any point, but all stages should be followed in sequence for achieving a successful learning process.

When trying to solve a programming exercise, the student identifies the specific programming resources he or she needs (e.g. constants, variables, control flow structures, program modules,…), and establishes some criteria and evidences that will let him know about the correctness of his solution to the proposed exercise (*Active Experimentation*). Then, he or she writes and compiles a program and, if there are no compilation errors, the computer runs it (*Action*). When the computer monitor shows the compilation errors or the execution results, the student gathers information and reflects on what has happened (*Reflective Observation*).

Finally, the student extracts conclusions from the analysis of the available information in order to learn from his experience (*Abstract Conceptualization*). He or she uses the previously established criteria and evidences in order to determine the achievement grade of his aim – a correct program execution with a right solution – or not – the existence of compilation errors or a wrong program execution. In this last case, the student will use his conclusions to carry on to a further stage of *Active Experimentation* in order to plan a revised strategy (i.e. make the corresponding corrections to the program, recompile it and, if there are no errors, run it again) that allows him to achieve his aim, so that the learning cycle starts again.

Thus, the Kolb cycle implies that it is not enough to have an experience to learn, but there must be a reflection process after the experience: It is critical to reflect on the experience in order to formulate concepts that can be applied to new environments. Finally, this

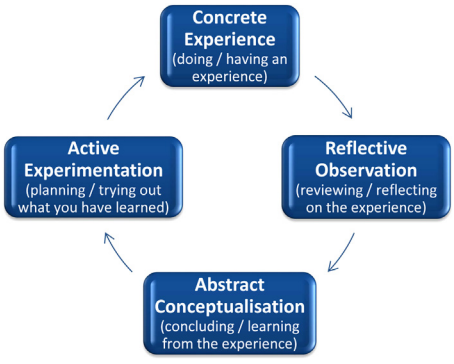

Fig. 1. Kolb learning cycle.

Fig. 2. A game with four Sifteo cubes.

new learning will be tested in new situations and contexts. In this way, theory, action, reflection and practice are linked into a dynamic cycle and complement each other (Shon, 1991).

Finally, with the help of the technological support provided by Sifteo cubes (Sifteo Inc., 2013a), we expect to encourage the student to apply "left handed" qualities as intuition and creativity (Bruner, 1979) to his own learning process. In this case, the study of an object-oriented programming language.

3. Technological foundations

Currently, there is a great deal of activity in wireless sensor networks, in which many computationally-equipped nodes cooperate to perform a wide variety of tasks. In (Merrill, Kalanithi, & Maes, 2007), the application of principles from sensor network technologies to tangible user interface (TUI) research is proposed in order to open up new interaction possibilities.

Sensor networks (Akyildiz, Su, Sankarasubramaniam, & Cayirci, 2002) consist of collections of sensing and communication devices that can be distributed spatially and are capable of exhibiting coordinated behavior. Sensor network nodes can be built with an array of sensing technologies that can be used to build rich models of local interactions.

Tangible user interfaces (Horn, Solovey, Crouser, & Jacob, 2009) and tangible interaction are terms increasingly gaining concurrency within the human–computer interaction community. TUIs utilize physical representation, manipulation of digital data and offer interactive couplings of physical artefacts with computationally mediated digital information (Xie, Antle, & Motamedi, 2008).

The conception of a Sensor Network User Interface (SNUI), used to develop the Siftables (Merrill et al., 2007) and later the Sifteo cubes (Merrill, Sun, & Kalanithi, 2012; Sifteo Inc., 2013a), takes the form of a distributed TUI in which several small physical devices have sensing, wireless communication, and user-directed output capabilities. They are a generic interaction platform that combines the flexible graphical display capabilities of the GUI with the physicality of a TUI, coupled with the capabilities of a sensor network.

Siftables (Merrill et al., 2007) consist of a collection of compact tiles, each with a microcontroller, a color LCD screen, a 3-axis accelerometer, four IrDA infrared transceivers, an onboard rechargeable battery and an RF radio. Sifteo cubes (Merrill et al., 2012; Sifteo Inc., 2013a) are the evolved and commercial version of Siftables (Fig. 2), so each cube replaces the Siftable microcontroller with a 32-bit ARM CPU,

Fig. 3. Sifteo cubes into the charging dock.

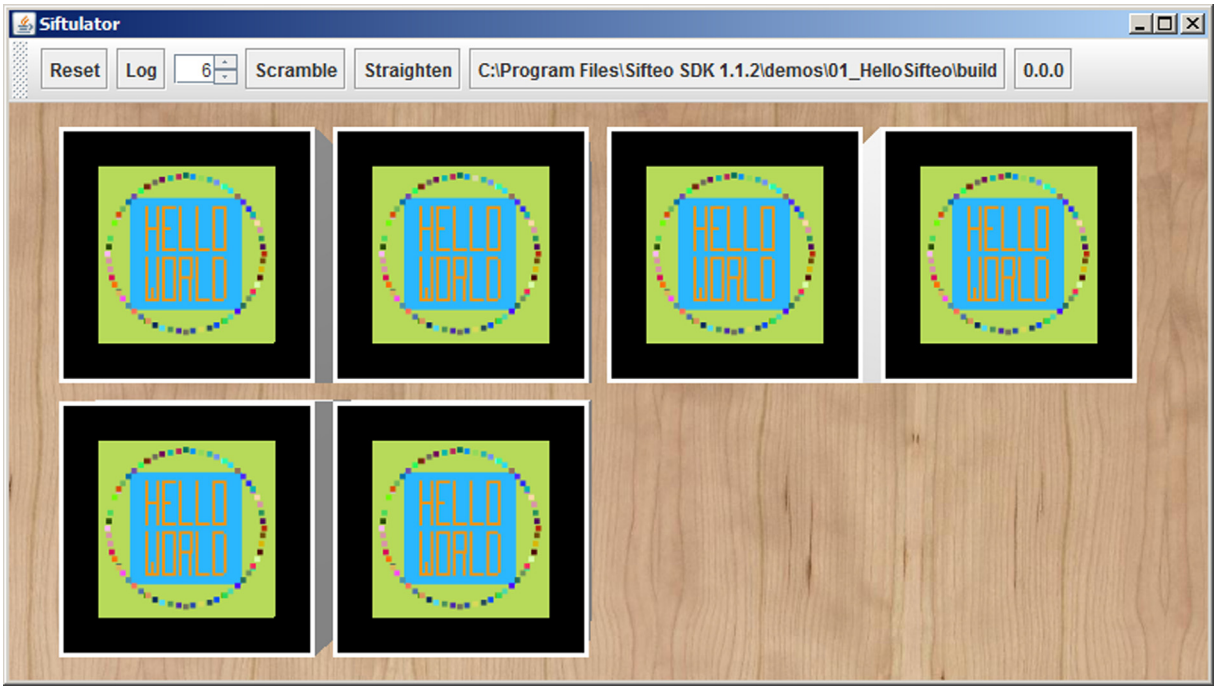

Fig. 4. Application "Hello Sifteo" running in "Siftulator".

includes an 8 MB Flash memory and is powered by a lithium polymer rechargeable battery (Fig. 3). It has a clickable 128 × 128 color TFT screen and uses a 2.4 GHZ wireless radio signal to communicate with the host computer. The cubes can detect each other using the Sifteo near field object sensing technology.

Sensing in the Sifteo cube design is accomplished by the 3-axis accelerometer and the Sifteo object sensing technology. A Sifteo cube can sense its own motion in the plane of a surface as well as the action of being lifted, tilted or shaken. The proximity sensors allow a cube to detect other neighboring cubes at a close range (about 1 cm). These sensing, graphical display and wireless communication capabilities allow Sifteo cubes to behave as a single, coordinated interface to information and media.

For a typical data manipulation task, each cube is provided via radio with an image that represents an instance of the data to be sorted. The physical manipulations performed by the user to the set of cubes are sensed and used as inputs to the specific program that controls the manipulation task. Visual feedback during the task is shown to the user on the cube screens, and auditory feedback can be played by the computer that runs the manipulation task control program.

Applications for Sifteo cubes can be developed using the *Sifteo Software Development Kit* (Sifteo Inc., 2013b), that contains *Siftdev* - the desktop application that provides the runtime support for Sifteo cube applications (*Siftrunner*), with additional tools for developers -, a simulator (*Siftulator*) that allows to test the applications without using the cubes (Fig. 4), and finally, a series of example applications and a reference documentation about the available C# packages and classes in the *Sifteo API*, that allow developers to deal with all the functional features of Sifteo cubes (i.e. sensing technology and TFT display).

In addition to an Apple Mac or a Windows PC with minimum system requirements (Sifteo Inc., 2013b), developers must also have the Mono - an open-source, cross platform C# and .NET environment that supports Linux, Mac OS X and Windows – runtime installed. Mono (Dumbill & Bornstein, 2004; Xamarin, 2012) is a software platform designed to allow developers to easily create cross platform applications. It is an open-source implementation of *Microsoft .Net Framework* (Platt, 2003) for C# and the *Common Language Runtime*.

C# (Michaelis, 2010) is a simple, modern, object-oriented, and type-safe programming language (Microsoft Corp., 2010). It has its roots in the C family of languages and will be immediately familiar to C, C++, and Java programmers. C# is standardized by ECMA International as the *ECMA-334* standard (ECMA International, 2006) and by ISO/IEC as the *ISO/IEC 23270* standard.

C# shares many characteristics with Java, but it has many features that Java does not (Reges, 2002), as *enums*, *reference parameters*, *operator overloading*, *properties*, *iterators* and the *foreach* loop, *delegates* (a sort of type-safe function pointers) and *a more consistent object model*. Some of them have been clearly taken from C++. The language designers had the advantage of knowing about Java weaknesses and this fact has allowed them to address many of such weaknesses.

4. Application of software demos to the study of the C# object-oriented programming language

This section describes two simple C# software demos that use the Sifteo cube technology in order to illustrate some important OOP concepts (Détienne, 1997; Meyer, 1997; Rumbaugh, 1991) as *class*, *object*, *attribute*, *method* and even *inheritance*. Also, some event-based programming concepts as *events* (Faison, 2006) and C# *delegates* (ECMA International, 2006; Michaelis, 2010; Microsoft Corp., 2010) can be explained in the same way.

We have developed a first demo ("Screen Colors") that displays ordered sequences of colors on the screen cubes. Such sequences will progress independently in each cube by pressing its clickable screen (*ButtonEvent*). After the last color, the first one in the sequence is

displayed again. Shaking a cube (*ShakeStartedEvent* and *ShakeStoppedEvent*) restarts the color sequence for this cube, whose screen shows the first color again. Next paragraphs describe the program code step by step:

The *System* assembly contains the core C# APIs, whereas the *Sifteo* assembly includes all the APIs for communicating with the Sifteo cubes.

```
using System;
using Sifteo;
```

The application logic must be written into a class that inherits from the *BaseApp* class. Since the application classes – that are subclasses from *BaseApp* – are meant to be executed inside the runtime environment *Siftrunner*, sometimes, a bootstrapping process is required to start the program manually, outside of *Siftrunner*, for debugging purposes. The *Bootstrap* class is a simple wrapper with a *Main()* method that starts the application: First, we get an instance (object) of the application class, and then we call its *run()* method.

```
public class Bootstrap {
  public static void Main() {
    ScreenColorsApp myScreenColorsApp = new ScreenColorsApp();
    myScreenColorsApp.Run();
  }
}
```

ScreenColorsApp is declared to be a derived class from *BaseApp*. *Siftrunner* will look for this subclass when it launches the application. We have declared an array that will be later set to the colors to be shown on the cube screens. *initialColorSound* and *nextColorSound* variables will be assigned two sound objects to be used when a cube screen shows again the first color of the sequence and when it changes its current color for the next one in the sequence, respectively.

The *Setup()* method is called when the application starts. Thus, all the necessary logic for initializing it should be included into this method. In this case, we initialize the data structure that supports the color sequence, that is an array of a C# *struct*[1] data type (ECMA International, 2006; Michaelis, 2010; Microsoft Corp., 2010) named *Color*, defined in the Sifteo API. Each array element is initialized using the *Color* struct constructor, that accepts a specific color given in RGB format: One value from 0 to 255 for each basic color.

The *colorIndex* variable is set to zero, as it must point to the first color in the sequence. This value will be stored into the *userData* public attribute of each cube object. Such attribute is an object that provides a convenient way for tagging a *Cube* instance with some additional data needed for the application purposes.

The *SoundSet* class is the set of audio assets available to an application. The C# property *BaseApp.Sounds* allows to access the sound set for a given application. *SoundSet* is also a *factory* for creating *Sound* instances. Thus, the invocation of the *CreateSound()* method creates an audio playback instance for a given MP3 or *Pulse Code Modulation* (a Windows ".wav" file) sample: A *Sound* object that represents the audio sample. For this demo, we have used two sound files named "initialcolor.mp3" and "nextcolor.mp3" for the *initialColorSound* and *nextColorSound* objects respectively.

Each instance of the *Cube* class represents a physical Sifteo cube. The *CubeSet* class represents the set of cubes currently connected to the computer through the Sifteo USB radio link. The *BaseApp.CubeSet* property[2] (ECMA International, 2006; Michaelis, 2010; Microsoft Corp., 2010; Reges, 2002) provides an instance of *CubeSet* for developers' use, that can be indexed in order to access a specific *Cube* object: Cubes are stored in the order in which they're connected, so that *CubeSet[0]* would return the first connected cube object.

The *foreach* statement allows us to iterate over all the cubes in *CubeSet* in order to carry out a same set of actions over each cube connected to the system. First, we store the current index value for the color array into the *userData* public attribute of each cube object. Next, we paint each cube entire screen the color from the array that corresponds to the index value, that in this occasion is equal to zero. However, nothing drawn will be actually shown on a cube display until its *paint()* method is called (see the *Tick()* method description in a next paragraph). In the last code lines of the *foreach* statement, we add event handlers[3] (ECMA International, 2006; Faison, 2006; Sifteo Inc., 2013b) to each cube for controlling button and accelerometer actions. For this demo, we consider the events triggered when the user presses and releases the clickable screen of a cube and when he or she shakes it.

[1] A *struct* is a C# structured data type based on the C++ *struct*, that as well as properties and data fields, it may contain methods and constructors. An essential difference between a class and a struct is related to their memory storage: The former is a *reference type* whereas the latter is a *value type*. Also, inheritance is not supported for structs.

[2] C# allows a programmer to have the simplified syntax and preserve encapsulation by defining a *property*: A member that provides access to a characteristic of an object or a class. A property can be accessed using the same syntax as a data field, but the access is translated by the compiler into calls on *get()* and *set()* accessors defined in it. Therefore, properties are a natural extension of data fields.

[3] Objects in the Sifteo API, particularly *BaseApp*, *CubeSet* and *Cube*, trigger events to notify an application of various happenings, including the manipulations that the user performs on the cubes. To listen for an event, a handler method (which specifies the actions to be carried out when the event is triggered) must be added to it.

```
public class ScreenColorsApp : BaseApp {

  Color [] colors;
  Sound initialColorSound, nextColorSound;

  public override void Setup() {

    int colorIndex = 0;

    // colors: red, green, blue, yellow, cyan, magenta and white
    colors = new Color[] { new Color(255, 0, 0), new Color(0, 255, 0),
             new Color(0, 0, 255), new Color(255, 255, 0),
             new Color(0, 255, 255), new Color (255, 0, 255),
             new Color(255, 255, 255) };

    initialColorSound = Sounds.CreateSound("initialcolor");
    nextColorSound = Sounds.CreateSound("nextcolor");

    foreach (Cube cube in CubeSet) {
            cube.userData = colorIndex;
            cube.FillScreen(colors[colorIndex]);
            cube.ButtonEvent += OnButton;
            cube.ShakeStartedEvent += OnShakeStarted;
            cube.ShakeStoppedEvent += OnShakeStopped;
            }

    initialColorSound.Play(1);
  }
```

Finally, the *Play()* method is called on the *initialColorSound* object in order to indicate that the first color in the sequence is displayed for the first time on the screen of all the connected cubes and thus, the application has started. The parameter is a real number that indicates the volume between zero (the lowest) an one (the highest) for the sound to play.

The *override* keyword (ECMA International, 2006; Michaelis, 2010; Microsoft Corp., 2010) allows the original *Tick()* method from the *BaseApp* base class to be replaced in the *ScreenColorsApp* derived class with a more suitable implementation for the application purposes. It is called every 1/20th of a second by default (the *tick rate* is adjustable) and, in this case, it iterates over the object representations of the connected Sifteo cubes in order to call their *paint()* method. This method updates the screen image of a cube, so that all the graphics previously drawn through the corresponding drawing methods are now shown on the cube display.

```
public override void Tick() {
  foreach (Cube cube in CubeSet) cube.Paint();
}
```

This is a handler for the *Button* event, that is triggered when a cube clickable screen is either pressed or released. The *pressed* argument is true in the first case and false in the second one. Obviously, the *cube* argument represents the physical Sifteo cube whose button has been used. The *UniqueID* property returns the *unique identifier* for the corresponding cube object when it is read. The value of such property will remain constant across disconnects and game sessions, so it can always be used to uniquely identify a single cube.

When a cube button is pressed, the sound corresponding to a color change is played, and when it is released, the index value corresponding to the current screen color is retrieved from the *userData* cube object public attribute and it is stored into the *colorIndex* local variable. Next, this variable is incremented by one. If its value reaches the color array length, then the variable is set to zero for indexing again the first color array element, that represents the first color in the sequence. Finally, the *userData* cube object attribute is assigned the updated value of *colorIndex* and the screen cube is painted the indexed color in the array.

```
private void OnButton(Cube cube, bool pressed) {
  Console.Write("Cube: {0}. ", cube.UniqueId);
  if (pressed) {
     Console.WriteLine("Button pressed.");
     nextColorSound.Play(1);
     }
  else {
     Console.WriteLine("Button released.");
     int colorIndex = (int) cube.userData;
     colorIndex++;
     if (colorIndex >= colors.Length)
        colorIndex = 0;
     cube.userData = colorIndex;
     cube.FillScreen(colors[colorIndex]);
     }
}
```

This is a handler for the *ShakeStarted* event, that is triggered when the user starts shaking a cube. The *cube* argument represents the physical Sifteo cube that is being currently shaken, and its unique identifier is displayed on the computer monitor for execution tracing purposes. The specific action to carry out in this case consists of playing the sound corresponding to a new beginning of the color sequence.

```
private void OnShakeStarted(Cube cube) {
  Console.WriteLine("Cube: {0}. Shake start.", cube.UniqueId);
  initialColorSound.Play(1);
}
```

This is a handler for the *ShakeStopped* event, that is triggered when the user stops shaking a cube. The *cube* argument represents the physical Sifteo cube that has just been shaken, and its unique identifier is displayed on the computer monitor for execution tracing purposes. The *duration* argument indicates how long the cube has been shaken and it is measured in milliseconds. First, a local variable named *colorIndex* is declared and set to zero. Next, the *userData* cube object public attribute is assigned this value and the screen cube is painted the first color in the array, in order to restart the color sequence for the shaken cube.

```
private void OnShakeStopped(Cube cube, int duration) {

  int colorIndex = 0;

  Console.Write("Cube: {0}. ", cube.UniqueId);
  Console.WriteLine("Shake stop: {0} seconds.", duration / 1000.0);
  cube.userData = colorIndex;
  cube.FillScreen(colors[colorIndex]);
 }
}
```

The second demo (“Hello Sifteo”) was developed by Sifteo and it is included in the *Software Development Kit* (Sifteo Inc., 2013b). Once the Sifteo SDK has been installed, this demo can be retrieved from the disk folders “/demos” and “/demo-docs”. We will only give a brief description of this second application, since its code contains a large number of explanatory comments.

The program illustrates the basic structure of an application for Sifteo cubes and shows some ways of drawing graphics on a cube. First, each cube screen is entirely painted a previously defined RGB color that is the same for all the connected cubes. Next, a square of another RGB color is drawn in the center of each screen and the text “HELLO WORLD” is printed inside the squares. Finally, the program draws a 4x4 randomly colored dot that describes a circular trajectory around each square. Fig. 4 shows this second demo running in a Sifteo simulator (*Siftulator*) window.

These two demos are useful for illustrating some object-oriented and event-based programming basic concepts for those students that are being introduced to these programming paradigms using the C# language. The main concepts are:

Table 1
Values of indicators and standard deviations for the two groups of students.

Indicator name	Experimental group		Control group	
	Value	Std. deviation	Value	Std. deviation
Clarity level (presentation)	3.57	0.62	3.32	0.69
Interest level (presentation)	4.01	0.65	2.93	0.57
Spent time (study)	134.36	30.81	139.93	15.87
Achieved mark (test 1)	8.24	0.54	8.22	0.31
Achieved mark (test 2)	6.64	1.23	4.20	1.38
Spent time (test 1)	12.07	0.70	13.40	0.48
Spent time (test 2)	12.64	0.81	12.80	1.05
Difficulty level (test 1)	2.43	0.90	2.53	0.83
Difficulty level (test 2)	2.57	0.49	2.81	0.56

- Classes: The *Bootstrap* startup class that launches the application (*ScreenColorsApp*), that is also a class.
- Objects: A Sifteo cube, a color and a sound can be represented and handled as objects.
- Attributes (data fields): *uniqueId* and *userData* are cube attributes.
- Methods: *FillScreen()* and *Paint()* for a cube object, and *Play()* for a sound object.
- Inheritance: The *ScreenColorsApp* application class inherits from the *BaseApp* Sifteo API base class.
- Method overwriting: *Setup()* and *Tick()* are original methods from the *BaseApp* base class that are overridden in the *ScreenColorsApp* derived class.
- Event handlers: *OnButton()*, *OnShakeStarted()* and *OnShakeStopped()*, that allow specific actions to be performed when the corresponding events (i.e. *ButtonEvent*, *ShakeStartedEvent* and *ShakeStoppedEvent*) are triggered.

5. Experimental results and discussion

In the School of Engineering (University of Cádiz), the course *Fundamentals of Computer Science* is imparted in the first year of the Degree in Industrial Technology Engineering (European Commission, 2013). At present, this course consists of two main topics: introduction to Information Technology and introduction to Computer Programming in C language (Kernighan & Ritchie, 1988; Rodríguez Corral & Galindo Gómez, 2006). The first one is an introduction to computer architecture (CPUs, memories, I/O, buses and evolution of computers) (Hennesy & Patterson, 2006) and operating systems (history/evolution, system calls, process scheduling, memory and I/O management, and file system) (Tanenbaum & Woodhull, 2006).

The second topic covers the basic knowledge about programming in C language: basic data types, constants, variables and operators, control flow statements, functions, arrays and files. The pupils of this course are being initiated in the C procedural programming language and, moreover, they do not have any knowledge about OOP. These initial conditions make of them a suitable group for our study. Furthermore, C++, Java and C# are languages based on C to some extent and, therefore, the syntax in general, the basic data types and the control statements are known by these pupils.

We have selected a sample of thirty pupils aged 18–19 for our study: Fifteen pupils (the control group) have attended a standard C# OOP course that includes practical demonstrations (i.e. computer execution) of example programs, without the technological and didactic contribution of Sifteo cubes, and the other fifteen (the experimental group) have attended a course that makes use of such contribution (that is, it includes the demos for Sifteo cubes described in the previous section).

For our study, we have selected those pupils that have demonstrated a high performance level and a positive attitude during the teaching of the course *Fundamentals of Computer Science*, in order to be sure that they are really interested in computer programming.[4] Furthermore, the selected pupils have been given an incentive consisting of improving their final mark in the course, to motivate their participation in the experimental tests.

Once the two courses (the standard course and the one that incorporates the technological contribution of Sifteo cubes as a didactic innovation) have been imparted, both groups of pupils have been asked to indicate their perception of the interest and the clarity level of the exposition, using a scale between one and four for avoiding the central tendency, as well as the spent time for the study of the course contents. Also, they have completed two tests: a multiple-choice exam for evaluating the understanding of basic OOP concepts, and a programming exercise also based on multiple choices in order to reduce the subjectivity in the marking process. From such tests, we have obtained the following data for each pupil: the achieved mark, based on the number of correct answers, the time taken in resolving each test and their subjective perception of the difficulty level.

Due to a human-related error committed in the selection process, an outlier - an observation that deviates significantly from the majority (Barnett & Lewies, 1994; Liu, Shah, & Jiang, 2004) - was detected in the experimental group and, therefore, it has been removed from the sample. So, the experimental group has now fourteen pupils instead of fifteen.

Table 1 shows the values of a set of indicators, calculated as the mean values of the obtained results for each group (experimental and control), along with the corresponding standard deviations. A brief description of these indicators is provided below:

- Subjective perception of the clarity level (from 1 to 4) for the course exposition.
- Subjective perception of the interest level (from 1 to 4) for the course exposition.
- Spent time (expressed in minutes) for the study of the course contents.

[4] Since the sample has been obtained using a pre-established selection criteria instead of a random sampling method, this work represents a pilot study: A starting point from which more detailed studies can be carried out.

Table 2
Mean values of indicators for the two tests.

Indicator name	Experimental group	Control group
Achieved mark	7.44	6.21
Spent time	12.35	13.1
Difficulty level	2.5	2.67

- Achieved mark (from 0 to 10) for tests 1 and 2 respectively.
- Spent time (expressed in minutes) for tests 1 and 2 respectively.
- Subjective perception of the difficulty level (from 1 to 4) for tests 1 and 2 respectively.

Thus, the specific values of the indicators and the standard deviations are shown for each group of students that have been taken part in the experiment:

- Experimental group: Students that have taken part in a C# OOP course using Sifteo Cubes as a technological and didactic resource.
- Control group: Students that have taken part in a standard C# OOP course.

Table 2 shows the mean values of the indicators for both tests: *mean achieved mark*, *mean spent time* and *mean perception of the difficulty level*.

From the obtained results we can observe that, for the *interest level* and *achieved mark (test 2)* indicators (Table 1), the experimental group clearly achieves higher values than the control group. First, a higher interest level for the exposition of a topic is related to a greater motivation in the student's learning process. On the other hand, the C# course for the experimental group includes the two sample software demos using Sifteo cubes, described in section IV. Thus, the achievement of a higher value for the *achieved mark (test 2)* indicator could be explained as the result of using more meaningful and illuminating examples instead of a typical set of standard C# sample codes to be executed in a computer.

Finally, despite that the most important differences between the experimental and the control group results are found in the values of the *interest level* and *achieved mark (test 2)* indicators, it must be pointed out that all the existing differences are in favor of the experimental group. Although this work is a preliminary study, such fact is significant and must be kept in mind.

From our experience, we think that not only has the use of Sifteo cubes, as a technological and didactic resource, a positive influence on the student's learning process, but it also motivates the professor's teaching activity and gives it a greater quality. Certainly, the teaching of the C# language and the explanation of OOP basic concepts are both improved and made easier when they are carried out with the help of this visible and tangible support.

6. Conclusions

In this work, we have applied the use of a specific distributed tangible user interface (TUI) to the teaching of the C# object-oriented programming language, since the operation of Sifteo cubes is controlled by programs written in this language. Such distributed TUI operates as a wireless sensor network connected to a computer through a compact USB radio link.

From the analysis of results presented in the previous section, we can conclude that the technological contribution of Sifteo cubes – used as a distributed TUI by which OOP basic concepts are represented in a tangible and a visible way – to the teaching of the C# language has a positive influence on the learning of this language as well as of such basic concepts.

7. Future work

We have carried out a pilot study to obtain a preliminary measure of the validity for a teaching methodology that incorporates a technological and didactic resource, which we consider to be innovative to a certain extent. As a continuation of this work, we propose to carry out a continued and detailed study about the learning process of a group of students that applies the use of this resource to the study of the C# programming language.

The course *Object-oriented Programming* is imparted in the second year of the Degree in Computer Science and Engineering, also in the School of Engineering (University of Cádiz). So, the pupils of this course make up a suitable population for this new study, for obvious reasons.

In order to achieve this purpose, it will be necessary to develop additional teaching material, as a set of standard guided practical activities for the control group that include usual examples and programming exercises (i.e. which can be written and executed in a computer without needing additional resources), and a set of "innovative" guided practical activities for the experimental group, whose examples and exercises are based on applications for Sifteo cubes.

References

Akyildiz, I. F., Su, W., Sankarasubramaniam, Y., & Cayirci, E. (2002). Wireless sensor networks: a survey. *Computer Networks, 38*(4), 393–422.

Barnett, V., & Lewies, T. (1994). *Outliers in statistical data* (3rd ed.). NY: John Wiley & Sons.

Bruner, J. S. (1979). *Essays for the left hand* (2nd ed.). Cambridge, MA: Harvard University Press.

Détienne, F. (1997). Assessing the cognitive consequences of the object-oriented approach: a survey of empirical research on object-oriented design by individuals and teams. *Interacting with Computers, 9*(1), 47–72.

Dumbill, E., & Bornstein, N. M. (2004). *Mono: A developer's notebook*. USA: O'Reilly Media.

ECMA International. (2006). *C# language specification* (4th ed.) <http://www.ecma-international.org/publications/standards/Ecma-334.htm> Retrieved 07.07.2012.

European Commission. (2013). *The Bologna process – Towards the European higher education area*. <http://ec.europa.eu/education/higher-education/doc1290_en.htm> Accessed 13.02.2013.
Faison, T. (2006). *Event-based programming: Taking events to the limit*. Breinigsville, PA: Apress.
Fink, L. D. (2003). *Creating significant learning experiences: An integrated approach to designing college courses*. San Francisco, CA: John Wiley and Sons, Inc.
Hennesy, J. L., & Patterson, D. A. (2006). *Computer architecture. A quantitative approach* (4th ed.). San Francisco, CA: Morgan-Kaufmann.
Horn, M. S., Solovey, E. T., Crouser, R. J., & Jacob, R. J. (2009). Comparing the use of tangible and graphical programming languages for informal science education. In *Paper presented at the CHI 2009 27th computer–human interaction conference* (pp. 975–984). Boston (MA).
Kernighan, B. W., & Ritchie, D. M. (1988). *The C programming language* (2nd ed.). NJ: Prentice Hall.
Kolb, D. A. (1984). *Experiential learning. Experience as the source of learning and development*. Englewood Cliffs, NJ: Prentice Hall.
Lesgold, A. (2004). Contextual requirements for constructivist learning. *International Journal of Educational Research, 41*(6), 495–502.
Liu, H., Shah, S., & Jiang, W. (2004). On-line outlier detection and data cleaning. *Computers & Chemical Engineering, 28*(9), 1635–1647.
Merrill, D., Kalanithi, J., & Maes, P. (2007). Siftables: towards sensor network user interfaces. In *Paper presented at the TEI'07 1st international conference on tangible and embedded interaction* (pp. 75–78). Louisiana (USA).
Merrill, D., Sun, E., & Kalanithi, J. (2012). Sifteo cubes. In *Paper presented at the CHI 2012 conference on human factors in computing systems* (pp. 1015–1018). Austin (Texas).
Meyer, B. (1997). *Object-oriented software construction* (2nd ed.). Upper Saddle River, NJ: Prentice-Hall PTR.
Michaelis, M. (2010). *Essential C# 4.0* (3rd ed.). Ann Arbor, MI: Pearson Education, Inc.
Microsoft Corporation. (2010). *C# language specification. Version 4.0*. <http://www.microsoft.com/en-us/download/details.aspx?id=7029>. Retrieved 24.01.2013.
Overmars, M. (2004). Learning object-oriented design by creating games. *IEEE Potentials, 23*(5), 11–13.
Platt, D. S. (2003). *Introducing Microsoft.NET* (3rd ed.). USA: Microsoft Press.
Reges, S. (2002). Can C# replace Java in CS1 and CS2?. In *Paper presented at the ITiCSE'02 7th annual conference on innovation and technology in computer science education* (pp. 4–8). Aarhus (Denmark).
Rodríguez Corral, J. M., & Galindo Gómez, J. (2006). *Learning C* (in Spanish) (3rd ed.). In *Third edition revised and extended*. Spain: University of Cádiz Press.
Rodríguez Corral, J. M., García Vargas, I., & Morgado Estévez, A. (2006). Guided practical activities for learning programming languages. In *Paper presented at the m-ICTE2006 IV international conference on multimedia and information and communication technologies in education* (pp. 871–875). Seville (Spain).
Rosson, M. B., & Carroll, J. M. (1990). Climbing the smalltalk mountain. *ACM SIGCHI Bulletin, 21*(3), 76–79.
Rumbaugh, J. (1991). *Object-oriented modeling and design*. Englewood Cliffs, NJ: Prentice Hall.
Schon, D. A. (1991). *The reflective practitioner. How professionals think in action*. Aldershot, London (UK): Ashgate Publishing Ltd.
Sifteo Inc. (2013a). *Sifteo cubes*. <https://www.sifteo.com/product> Accessed 18.03.2013.
Sifteo Inc. (2013b). *Sifteo software development Kit for windows (version 1.1.3)*. <https://s3.amazonaws.com/updates.sifteo.com/Sifteo_SDK_win_1_1_3.exe> Accessed 26.03.2013.
Tanenbaum, A. S., & Woodhull, A. S. (2006). *Operating systems. Design and implementation* (3rd ed.). Upper Saddle River, NJ: Prentice Hall.
Xie, L., Antle, A. N., & Motamedi, N. (2008). Are tangibles more fun? Comparing children's enjoyment and engagement using physical, graphical and tangible user interfaces. In *Paper presented at the TEI'08 2nd international conference on tangible and embedded interaction* (pp. 191–198). Bonn (Germany).
Xmarin. (2012). *Mono. Cross platform, open source.NET development framework*. <http://mono-project.com/Main_Page> Accessed 05.12.2012.
Yan, L. (2009). Teaching object-oriented programming with games. In *Paper presented at the ITNG 2009 6th international conference on information technology: New generations* (pp. 969–974). Las Vegas, Nevada (USA).