Learning through Game Making: An HCI Perspective

Jeffrey Earp, Francesca Maria Dagnino, and Michela Ott

Istituto per le Tecnologie Didattiche – Consiglio Nazionale della Ricerche, Italy
{jeffrey.earp,dagnino,ott}@itd.cnr.it

Abstract. One of the areas of Game-Based Learning (GBL) that has been attracting considerable interest in recent years is digital game making, whereby learners play games but also design, construct and share them as active participants in a learning community. Human Computer Interaction (HCI) is a critical aspect of processes and tools within game making, and plays a key role in ensuring that learning experiences are both engaging and educationally fruitful. In this light, this paper examines two different game authoring environments from an HCI perspective, taking account of certain interface characteristics can affect and shape the authoring process and thus have a potential bearing on educational effectiveness. The investigation draws on findings from an EU co-funded project called MAGICAL (MAking Games In CollaborAtion for Learning), which is exploring the potential that game making offers for activating key transversal skills such as problem-solving, creativity and ICT competency, particularly at primary school level.

Keywords: Game Making, Game-Based Learning, Technology Enhanced Learning, Human Computer Interaction, Usability, Accessibility.

1 Introduction

In recent years, interest and enthusiasm for Game Based Learning (GBL) has strengthened considerably within the educational research community and GBL is now gaining wider acceptance among educational policy makers, administrators, practitioners and the public at large [1]. Some see GBL in practical terms as a way to kindle – or rekindle – learner interest, and to get students in formal learning to engage deeply with subject-related contents. From a more theoretical perspective, many GBL advocates see interaction with digital games as a process of active, learner-centered meaning making [2]. In this sense GBL is considered an alternative (or antidote) to instructionist-style "talk & chalk" lecturing, aligning more closely to modern, constructivist visions of education and with the educational principles underpinning the Knowledge Society [3]. It is held to motivate and engage players, immersing them in a learning experience that combines playfulness, challenge and fun [4-5]

While enthusiasm for GBL is spreading, many educational researchers warn that these hallmark characteristics do not *per se* generate effective learning outcomes, and simply having learners play one or other digital game will not necessarily yield the expected educational benefits [6]. Effectiveness depends on a range of factors that includes the nature and suitability of the core digital GBL environment itself. To start

C. Stephanidis and M. Antona (Eds.): UAHCI/HCII 2014, Part II, LNCS 8514, pp. 513–524, 2014.

with, this needs to blend educational and fun dimensions in a manner that is complimentary, suitably balanced, and an integral part of gameplay. This critical factor is summed up thus by a group of eminent UK educational researchers reporting for Nesta [7]: "*games that integrate the knowledge and skills to be learnt directly into the structure of the game activity are both more fun … to play and more effective than those where the game is used as motivation but without connection to the learning content.*" GBL environments proposing unconnected play and learning activities are often dismissed as "chocolate-coated broccoli"; they provide little opportunity for active meaning making and, as Luckin and colleagues [7] point out, are unlikely to foster the sustained engagement needed to enhance learning processes.

Against this background, a growing number of researchers (and practitioners) are taking GBL beyond the confines of game playing and encouraging learners to design and make their own digital games, which they can subsequently share with peers in a learning community. Indeed there is a steadily growing body of academic literature advocating game making as a way to empower learners, letting them take the driving seat in active, hands-on activities [8]. These activities are held to offer opportunities for building a range of knowledge and skills, not just within specific subject areas, such as language and math, but across them too. This covers skills like creativity, problem solving and collaboration, which are commonly characterized as Twenty-First Century Skills (21CS) and, as such, central pillars of modern education [9], [1].

A number of initiatives are currently emerging to investigate the adoption of game making for learning. One of these efforts is an EU co-funded project called MAGICAL (Making Games in Collaboration for Learning)[1], which is conducting school experiments to explore the potential that game making offers for activating key transversal skills such as problem-solving, creativity and ICT competency, particularly at primary school level [10].

Among the various activities undertaken in MAGICAL is investigation of different digital environments that have been (or might be) used fruitfully to support learning processes and educational objectives in game-making contexts. This effort has resulted in the establishment of a community library of game making environments[2], which currently catalogues over fifty game authoring tools. These vary widely and in different ways. For example, the games they can produce range from simple 2D arcade-style platform games to quite elaborate 3D game worlds. The nature of user-system interaction in the editing process also varies quite markedly. Some environments feature a limited set of elementary game elements and properties, making them particularly suitable for younger learners, for game makers with special learning needs (whether specific or non-specific), and also for use in restricted educational time frames. Other authoring tools have an extensive authoring palette that, amongst other things, offers game makers the chance to create fairly sophisticated relations and interactions in games, opening the way to the design of rich and complex gameplay experiences.

When it comes to selecting a game-making environment for educational purposes, HCI aspects are particularly critical, especially where younger students are concerned. Accordingly, adoption of a digital authoring tool calls for careful consideration of usability and ease-of-use factors [11-12]. Furthermore, all-important efforts to

[1] www. magical-project.net
[2] http://amc.pori.tut.fi/game-building-tools/

guarantee Universal Access to education demand that due attention also be devoted to accessibility issues.

In the following, we examine two game authoring environments that have featured in MAGICAL activities and that in certain respects exemplify different approaches to game making. After providing an overview of their main features, we identify some of the differences that these tools present from the viewpoint of adoption within digitally-enhanced learning activities. The ultimate purpose of the comparison is not to "evaluate" the tools or quantify their respective educational potential; doing so is a very complex and contentious matter. Rather, the aim is to provide food for thought on the aspects of HCI that are pivotal for the deployment of digital game making in educational contexts, aspects which may have a decisive bearing on the success of game making for enhancing learning outcomes.

2 Two Game Making Environments Under the Lens

The following section gives a general description of two popular game making environments: Kodu[3], a downloadable application for PC and console, and Sploder[4], an online browser-based tool. For details about these, readers should refer to the respective websites. In this contribution we concentrate on key differences in the authoring interfaces, which exemplify different approaches to game making.

2.1 Kodu: A Code-Based Game Authoring Tool

Kodu presents a visual programming language specially designed to allow non-programmers, and especially young children, to engage in code-based game authoring. As a tool for learning through computer programming, it builds on the constructionist philosophy and legacy of Seymour Papert and others, who developed the Logo visual programming language for children in the 1960s. In the words of Microsoft, "*the Kodu language is designed specifically for game development and provides specialized primitives derived from gaming scenarios*." To render programming concepts more readily comprehensible to young game makers, the language is largely anthropomorphic: behaviors are expressed metaphorically in the graphic interface in terms of real-world physics and human senses like vision, hearing and touch.

Kodu presents a 3D world in which you can build games using its special visual coding language. You can start from scratch with a near-empty 3D space or edit an existing game(world) to make a new, personalized version. Either way, you can opt to set your game in a minimalist-style 3D space with few embellishments or construct a graphically richer fantasy world for the player to explore and interact in. The authoring mode (**Error! Reference source not found.** and Fig. 2 below) offers a palette of easy-to-use graphics tools designed to support rapid generation of stylized 3D environments, which can be enriched with stylized characters and props, as well as sound

[3] http://www.kodugamelab.com/
[4] http://www.sploder.com/

and animation. The range of game making elements to choose from is reasonably varied but, at the same time, is not so extensive as to overwhelm the game author with myriad variations and endless choice.

Fig. 1. Lines of WHEN/DO visual code generated by a Kodu author (with contextual help)

Programming with Kodu is based on constructing WHEN>DO conceptual couplets, each of which generates a condition>response instance in the Kodu world. These building blocks of visual code can be strung together in a sequence to form a complete program (**Error! Reference source not found.** above), which runs as an interactive 3D game in Kodu's play mode. You start programming by selecting an element that you've included in the 3D game-space and associating a WHEN>DO unit to it. To make the unit, you choose two "atoms" (primitives) from the graphic library of ready-to-use objects, behaviors and actions (Fig. 2 below). Coupling these atoms together forms a WHEN condition, e.g. <player-clicks mouse>, <Kodu sees-target>. This WHEN condition then needs to be paired with a corresponding DO response, made by coupling two more atoms, e.g. <missile-fires>, <Kodu moves - to target>. The resulting WHEN-DO molecule forms a logical condition-response instance: e.g. <when player clicks mouse, missile is fired>, <when Kodu sees target, Kodu moves to target>. The programming syntax also includes the possibility to attach a condition to the WHEN-DO molecule, e.g. <when missile hits target, Kodu jumps once>, <when missile hits target, score increases by 50>. These conditions allow you to generate the specific behaviors needed to create a functioning game. Much of the programming logic involved in Kodu revolves around the application of these conditions, which are selected from the same graphical object library containing all the other atomic game elements.

So making a game in Kodu entails stringing together a sequence of these molecules to form the complete code. This can be very basic, e.g., a single programmed mechanic triggered, say, by the player's mouse clicks, or a complex multi-level scenario with gameplay that involves multiform interaction.

Fig. 2. Selecting elements from Kodu's visual programming library to generate lines of WHEN/DO visual code: contextual help displayed

Very experienced users may attempt to string code lines together in a single sweep, making a whole game "sight unseen" as it were; indeed Kodu allows cut-and-paste editing of whole lines of code to facilitate the process. However, most users will find themselves adding a molecule in Editing mode, then switching to Play mode to see how it runs, and then switching back again to tweak code or add a further molecule. Such progressive Edit-Play iterations allow you to check the result of coding on-the-fly and also to monitor how the game you're designing/making is unfolding.

2.2 Sploder: Creating Games through an Online Platform

Sploder in an online platform targeted largely at digital gaming enthusiasts. Along with a set of game making tools, it features social networking functions designed not just to promote game sharing but to support the formation and consolidation of a game-oriented community. Sploder offers five different game making formats: Retro Arcade, Platformer, Physics Puzzle, Classic Shooter and Algorithm Crew. These are largely similar but provide some variation in theme, style of game play, authoring palette and complexity of use. This overview concentrates on the Retro Arcade environment (Fig. 3 below), which is fairly representative of Sploder's game making approach. In Retro Arcade, you can generate tile-based, 2D scrolling platform games of the type that rose to popularity with the advent - and ubiquity - of the first generation game consoles. Like all the other Sploder formats, Retro Arcade allows you to build multi-level game structures, with levels made up of different stages or scenarios. The scenarios are built by selecting from the three ready-to-use game-world templates on offer: Forest, Cave and Tech World. These worlds can be personalized and extended using Retro Arcade's graphic drawing tools, which have been designed expressly to make game-world construction quick and simple. This allows the author's efforts to be channeled into the selection and integration of key game elements like characters, interactions and mechanics. Each game-world template is complimented by more or less the same palette of game-making elements and properties. In keeping with the retro arcade theme, the palette offers a set of player avatars, enemy sprites, hazards,

collectables, rewards, treasures, power-ups and the like. Some of these embed default game behaviors, e.g. the animated "baddie" sprites are pre-programmed to engage the player in battle until they are jumped upon and defeated. The player controls are typical of platformer games, i.e. the keyboard's arrow keys are used for moving horizontally and for executing vertical jumps, whose amplitude is pre-set; gravity is also preset and is fixed.

You can construct and integrate particular gameplay events using a linking tool that establishes relations between elements placed in the game-world. Fig. 3 below shows a simple example under construction in the authoring mode: links have been set between a pair of ground-level on/off switches and a barrier, so that the player's passage will first raise and then lower the barrier, allowing the character to pass (obviously the links are only visible while editing). You can cluster links together to build relational chains, and these can be further refined by applying logical operators like And, Or, Not, which are also overlain graphically in the editing phase. The editing environment features a game preview mode so you can check the game's development on-the-fly before publishing it.

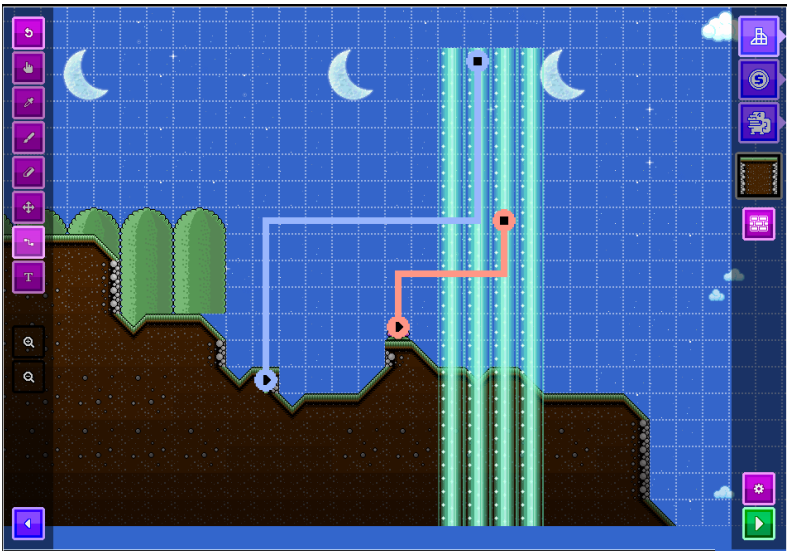

Fig. 3. Sploder Retro Arcade authoring mode: construction of mechanics using links

A distinguishing feature of the Sploder platform is its social networking capabilities. These support a community of game makers and players numbering over 25,000 members, allowing them to share games, graphics, ratings, reviews and comments.

2.3 The Two Game Authoring Environments: Comparison of Main Features

From the examination of the two game making environments presented above, we can draw the following picture (Table 1 below), which provides a synoptic view of how they differ in terms of specific HCI characteristics.

Table 1. Main differences between the two game authoring environments considered

Kodu	Sploder
Downloadable	Online, browser-based
Code-based game authoring	Object-based authoring
Visual programming language	Logical linking in situ for building game mechanics (drag & drop)
All behaviors user programmed	Some object behaviors preprogrammed
3D world	2D platform
User controls camera point-of-view (via mouse only)	Fixed point-of-view
Free movement of game characters in game-world (to be programmed)	Automatic side scrolling
Step-by-step construction of detailed 3D landscape, possibly from scratch	System-facilitated drag & drop construction of highly simplified landscape templates
Point & click navigation of palette menu	Drag-and-drop scrolling of palette menu
Single, open game type	Choice of five preset game types
Possible to structure game by level (<10)	Possible to structure game by level and sub-level
Create game levels from scratch	Create new (sub)levels using preset game-world templates
Switch between authoring and play modes	Switch between authoring and preview modes
Palette of game-world graphics, objects & behaviors	Palette of game-world graphics and objects (some preprogrammed)
Closed set of graphics	Editor for creating personalized graphics and textures
Soft, “toy-like” GUI style: soft tones, contours & shading; fluid/elastic motion; “cute” objects and sounds	Hard, high-contrast gaming-style GUI: pixel-style graphics, sharp motion and sounds, objects inspired by classic console games (nasties etc.)
Game-oriented behaviors: shooting, collecting etc.	Game-oriented behaviors: shooting, collecting, battles etc.
All animation to be programmed	Animated sprites (some preprogrammed)
Games can be saved to 2 external websites for sharing.	Integrated in native social networking platform with high activity levels
No advertising	Advertising present on website

Some of the above are determining factors for these environments' levels of accessibility, usability and ease of use, key areas of HCI. One obvious example that illustrates this point is the employment of drag-and-drop control in both environments, which poses a significant challenge for students with sensory disabilities of various kinds [13]. Without an alternative control method, these environments' accessibility is compromised and, as a result, game-making activities performed with them are less inclusive than they otherwise might be. Factors such as these not only impact on the ultimate effectiveness of game-making as an innovative learning method deployed in educational settings, they can have serious repercussions on practitioners' (and administrators') inclination to approach and adopt game-making in the first place. So, mindful of these considerations and their importance for MAGICAL's mission to support wider uptake of game-making for learning, we have carefully examined accessibility, usability and user experience issues.

2.4 The Two Game Authoring Environments: Accessibility and Usability

To begin with, the two environments were tested for *accessibility* according to the specifications laid out in Italy's law governing software applications destined for or used by public institutions[5] [14]; this law is largely based on Section 508 of the US Rehabilitation Act [15];. Neither application proved to be fully compliant with key accessibility requirements, as illustrated by the above-mentioned absence of any alternative where drag-and-drop control is required. Another obvious accessibility issue regards navigation of palette menus. In Kodu, the menus and items are quite clearly shown on large, graphically bold wheels that pop up automatically and are point-and-click controlled (Fig. 2). The nesting of menu levels is clearly apparent and readily comprehensible through transition from the large main menu wheel to the smaller sub-menu wheel at the point of juncture. While contrast and color differentiation between active/non-active menu items is far from optimal, the items themselves are clearly represented both textually and graphically. In addition, contextual help is presented automatically via high-contrast roll-over text (this function can be disabled). Sploder's palette menu is accessed via the three small icons in the top right-hand corner of the screen (Fig. 3). These open the next level, presented as a vertical toolbar that pops up to the right with much larger labeled icons (not shown in Fig. 3). Although this needs to be scrolled vertically there is no scroll bar, only a text hint displayed in a small help bar at the bottom of the editing window. If a toolbar item contains other sub-items to navigate to, this is indicated by tiny, almost imperceptible dots displayed under the icon. Navigating through these sub-items calls for a mouse-controlled point-click-hold-swipe motion left or right. Graphically, the distinction between active and non-active items is barely discernable. The browser's zoom function and the F11 keyboard shortcut for displaying the browser window full screen are often disabled; the authoring window does not rescale. The native zoom only zooms

[5] Italian Law 4-2004, also called *Legge Stanca*
`http://www.pubbliaccesso.gov.it/biblioteca/quaderni/rif_tecnici/Quaderno_4.pdf`

the game world canvas, not the editing dashboard. By contrast, Kodu can be displayed full screen and at various resolutions. To investigate the applications' *usability*, we refer to the eight Golden Rules proposed by [16]. Compliance with these varies considerably in the two environments, as shown in Table 2 below.

Table 2. Usability Golden Rules and application in two Game Making Environments

Golden Rule	Kodu	Sploder
1. **consistency** Employ uniform actions, terminology, color, layout, and text style. Limit exceptions like confirmation of delete command.	Generally compliant	Some noncompliance: color codes of control panel buttons confusing and sometimes inconsistent. Palette menu organized inconsistently.
2. universal usability Design for plasticity and facilitate content transformation. Cater for differences in expertise, age, (dis)abilities and technology.	Some compliance: display options for GUI buttons; advanced options available for experts; main keyboard/mouse controls displayed by default (text & icon); menu items clearly indicated, contextual help displayed; Windows keyboard controls & shortcuts; guided tutorials embedded in editable games and scaled in complexity	Very little compliance: some limited contextual help for toolset and dashboard controls
3. informative feedback Provide system feedback for all user actions - modest response for frequent / minor actions, more substantial for infrequent / major actions. Visual presentation of objects of interest	See note below	See note below
4. yielding closure group action sequences into beginning, middle, and end. Give accomplishment feedback at completion of a roup	See note below	See note below
5. (user) errors Where possible block or filter inappropriate user actions or input; these should leave system state unaltered (no response). Provide simple, constructive, specific recovery instructions.	Generally compliant: invalid editing selections & actions trigger no system state change. Undo/redo function available for recovery after user errors.	Generally compliant: invalid editing selections & actions trigger no system state change.

Table 2. (*continued*)

Golden Rule	Kodu	Sploder
6. easy action reversal fosters sense of user control and encourages exploration of unfamiliar options	Compliant: Undo/Redo button for back and forward tracking. Access to complete change history. Manual Save/Save- As for versioning. Auto Save prompt when closing editing sessions. Sessions automatically assigned a default version number to foster versioning.	Editor has a toggling eraser to remove objects that are unwanted but no Undo function. Each edit is automatically saved. No versioning function.
7. internal locus of control controllable, responsive interface requiring little menial effort or repetition. No obstacles to desired result.	Compliant: visual programming designed with features to reduce effort and repetition.	Compliant: production of game landscape and textures specifically designed to reduce effort and repetition.
8. short term memory load no memorization required between screens. Allow sufficient training time.	Generally compliant: step-by-step tutorials provided in the form of real, editable games	Non compliant: no tutorials or online guide.

With regard to Shneiderman's Golden Rule 3 on feedback and Rule 4 on closure, authoring environments present some specificities, especially those devoted to the production of more complex, interactive multimedia artefacts such as games. Firstly, game authoring is generally performed iteratively in a sequence of user-driven alternations between design/editing and preview/run phases. In a sense, core "system feedback" and closure can only come when the author triggers a preview/run to check the outcome of executed editing steps. So feedback and closure loops are actually very long and loose, and their amplitude and frequency is user governed (although it could be argued that allowing multiple-level game structuring, as Sploder does, encourages tighter looping to some degree). Furthermore, while the WYSIWYG principle of editing does apply to some (static) graphic elements included in the editing space, it cannot apply to the "programmed" interactions that are the hallmark of digital games.

Following the considerations in [17] related to the fact that "*users approach new software with diverse skills and multiple intelligence*", we acknowledge that the considerations reported above need to be integrated with data from user experience.. Findings on these aspects are expected to emerge from the conclusive stages of the MAGICAL project.

3 Conclusions and Further Work

In this paper we have examined some different aspects of HCI in game authoring environments that emerge from an investigation of two different game authoring environments currently being used with and by young students. The general objective has been to provide useful indications to support the implementation of game making for learning, which is steadily gaining support in formal education, especially for transversal skills development. The examination was carried out in the framework of MAGICAL, an EU project on game making that incorporates teacher training actions as well as school experiments. One of the main results expected of MAGICAL is the production of MAGOS, an authoring environment specifically intended for collaborative game making [18]. The design and development of MAGOS is grounded on a thorough analysis of existing authoring environments, some aspects of which are reported in this paper.

Issues of Human Computer Interaction are of particular significance in this sector, as indeed they are in Games-Based Learning and Technology Enhanced Learning, the wider fields to which digital game making belongs. Here, it is imperative that HCI poses no hindrances to the cognitive processes underpinning learning, but rather supports these in the global effort to achieve efficacy. It is with these concerns in mind that the work reported here was initiated and directed towards fulfilling two immediate aims: informing the game-making experiments performed in schools in MAGICAL's partner countries (Italy, Finland, Belgium and UK); and providing input for the development of the MAGOS environment. In both these areas valuable user-experience data is currently being generating that will serve to validate and enhance the initial findings reported in the paper. The authors intend to integrate that data so as to form a clearer, more detailed picture of HCI within the design and making of digital games for learning. The ultimate aim is to generate HCI-related indications for enhancing interface capabilities and affordances of authoring environments, and thus contribute to the appeal, efficacy and, eventually, wider uptake of game making as an educational practice.

References

1. Lizzy, B., All, A., Ilse, M., Dana, S., Van Looy, J., An, J., Koen, W., et al.: State of Play of Digital Games for Empowerment and Inclusion: a Review of the Literature and Empirical Cases. Publications Office of the European Union, Spain (2012)
2. Frossard, F., Barajas, M., Trifonova, A.: A Learner-Centred Game-Design Approach: Impacts on Teachers' Creativity. Digital Education Review (21), 13–22 (2012)
3. Lytras, M.D., Sicilia, M.A.: The Knowledge Society: a manifesto for knowledge and learning. International Journal of Knowledge and Learning 1(1/2), 1–11 (2005)
4. de Freitas, S., Neumann, T.: The use of 'exploratory learning' for supporting immersive learning in virtual environments. Computers & Education 52(2), 343–352 (2009)
5. de Freitas, S.: *Serious virtual worlds*. A scoping guide. JISC e-Learning Programme, The Joint Information Systems Committee (JISC), UK (2008)

6. Arnab, S., Berta, R., Earp, J., de Freitas, S., Popescu, M., Romero, M., Usart, M.: Framing the Adoption of Serious Games in Formal Education. Electronic Journal of e-Learning 10(2), 159–171 (2012)
7. Luckin, R., Bligh, B., Manches, A., Ainsworth, S., Crook, C., Noss, R.: Decoding Learning: The Proof, Promise & Potential of Digital Education. NESTA, London (2013)
8. de Freitas, S., Ott, M., Popescu, M.M., Stanescu, I.: Game-Enhanced- Learning: Preliminary Thoughts on curriculum integration. In: New Pedagogical Approaches in Game Enhanced Learning: Curriculum Integration. IGI Global (2013)
9. Dagnino, F., Earp, J., Ott, M.: Investigating the "Magical" effects of game building on the development of 21st Century Skills. In: ICERI 2012 Proceedings, pp. 5778–5785 (2012)
10. Bottino, R.M., Earp, J., Ott, M.: MAGICAL: Collaborative Game Building as a Means to Foster Reasoning Abilities and Creativity. In: 2012 IEEE 12th International Conference on Advanced Learning Technologies, pp. 744–745. IEEE (2012)
11. Holzinger, A.: Usability engineering methods for software developers. Communications of the ACM 48(1), 71–74 (2005)
12. Davis, F.D.: Perceived usefulness, perceived ease of use, and user acceptance of information technology. MIS Quarterly, 319–340 (1989)
13. Benigno, V., Bocconi, S., Ott, M.: Inclusive education: helping teachers to choose ICT resources and to use them effectively. eLearning Papers (6), 4 (2007)
14. CNIPA, Italian Law 4/2004 Provisions to support the access to Information Technologies for the disabled (2004), http://www.pubbliaccesso.gov.it/normative/law_20040109_n4.htm (retrieved February 1, 2014)
15. USA Access Board, Section 508 of the US Rehabilitation Act (2011), http://www.section508.gov (retrieved February 1, 2014)
16. Shneiderman, B., Plaisant, C.: Designing the User Interface: Strategies for Effective Human-Computer Interaction, 5th edn., 606 pages. Addison-Wesley Publ. Co., Reading (2010), http://www.pearsonhighered.com/dtui5einfo/
17. Shneiderman, B.: Universal usability. Communications of the ACM 43(5), 84–91 (2000)
18. Earp, J., Ott, M., Romero, M., Usart, M.: Learning through playing for or against each other? Promoting collaborative learning in digital game based learning. In: Proceedings of the ECIS 2012. AIS Electronic Library, Paper 93 (2012), http://aisel.aisnet.org/ecis2012/03