PAC-PAC: End User Development of Immersive Point and Click Games

Filippo Andrea Fanni[1], Martina Senis[1] Alessandro Tola[1], Fabio Murru[2], Marco Romoli[2], Lucio Davide Spano[1], Ivan Blečić[2], and Giuseppe Andrea Trunfio[3]

[1] University of Cagliari, Department of Mathematics and Computer Science
[2] University of Cagliari, Dept. of Civil, Environmental Engineering and Architecture
[3] University of Sassari, Department of Architecture, Design and City Planning

Abstract. We present a tool supporting end-users in the development of point-and-click videogames based on 360^o videos. It aims specifically at people without previous experience in game development and coding. Users can easily create scenes, add simple objects such as transitions or switches, connect scenes to each other and define the game rules. The tool is developed as a part of PAC-PAC, a project for promoting cultural and environmental heritage through videogames.

Keywords: Point-and-Click games, Authoring Environment, End-User-Development, Virtual Reality

1 Introduction

In the PAC-PAC project (an acronym for *"Point-And-Click - Patrimonio Ambientale e Culturale"*, Point-And-Click - Environmental and Cultural Heritage) we aim at fostering the promotion of tourist location through videogames. Its goal is developing an authoring environment for people working in tourism promotion for creating Point-and-Click (PaC) games set in real-world locations, for distributing them on the web as entertaining promotion material. The tool should require the same effort and knowledge needed for creating web or social network contents. The developed games exploit web technologies that work on desktop, mobile and Virtual Reality (VR) devices. We use 360^o videos for supporting immersive virtual visits, which are relatively simple to create with consumer hardware.

2 Creating games in PAC-PAC

The authoring environment represents a PaC game as a graph of 360^o videos nodes (*scenes*), connected by arcs implementing the *transition* from a location to another. The interface for editing the game recalls familiar multimedia content editing applications, such as e.g., MS Power-Point (see Figure 1). All the scenes are available in the left bar (Figure 1-C), which supports ordering and searching in the list. The main part (Figure 1-D) contains the current scene. It displays the

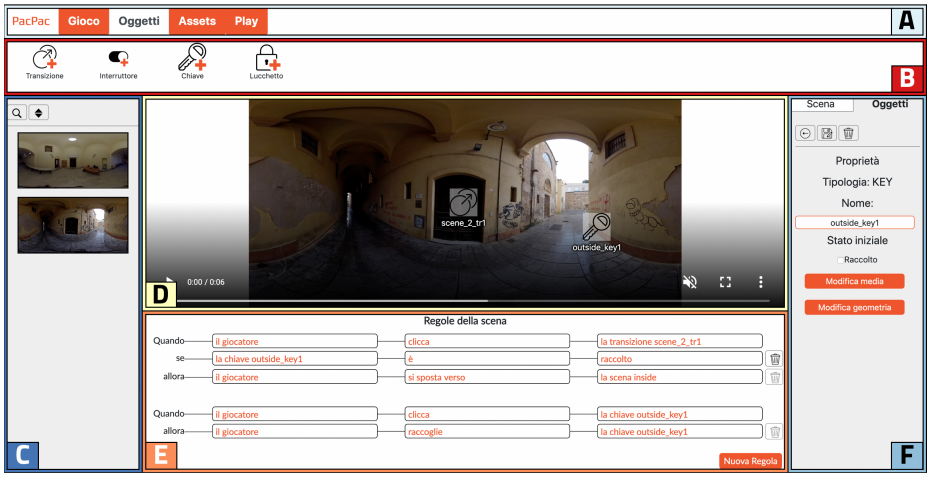

Fig. 1. The PAC-PAC game authoring interface

unwrapped video and the position of the interactive objects included in the scene. The top-level menu (Figure 1-A) has four elements: i) *Game* for adding new scenes and/or modifying game-level properties (e.g., background music, scene groups), ii) *Objects* for adding new interactive objects in the current scene, iii) *Assets* for managing the game assets (videos, overlay images, music, sound files etc.) and iv) *Play* for testing the game by playing it. When creating a new scene, the user selects a regular or 360^o photo/video from the asset collection, or uploads it from her own files. It is possible to group related scenes using semantic tags, which define macro-locations (e.g., all the scenes related to the same dungeon). Each tag is associated with a specific colour.

The authoring environment provides a set of objects helping the user in defining the game behaviour, displayed in the top bar (Figure 1-B). Each object represents a tool that, added to the scene, introduces an interactive element. For inserting an object into the scene, the user presses the corresponding button in the upper bar. Once inserted, the user modifies its configuration filling the properties panel in the right bar (Figure 1-F). Each object has an interaction area, which receives the player's clicks for activating the object. For instance, the interactive area of a transition is the surface the user clicks for changing the scene. The tool supports creating it by clicking the contour points in the 360^o video. In addition, each object has an optional property for animating the interactive area at the object activation through a video (e.g., opening a door when triggering a transition as in Figure 2-A).

The tool includes different types of interactive objects besides the *transition* we already discussed. For instance, the *key* allows unlocking another object when collected, the *switch* allows changing the current scene configuration according to its status (e.g., a light turns on/off), the *counter* counts the clicks on the specified interactive areas, the *timer* triggers an action after the specified time,

Fig. 2. Game interaction and the associated rules in two examples: modelling a door and the associated key (part A) and a switch for turning lights on or off (part B).

the *locker* requires clicking a set of interactive areas in a specified order for triggering an action etc.

Similarly to other EUD tools in different domains [1, 3, 2], our authoring environment uses rules for defining the behaviour of each object in isolation and their interactions for creating the game puzzles. Rules define the dynamic behaviour of a scene, supporting the user in inspecting and controlling the game logic. The rule editor is located in the bottom part of the interface (see Figure 1-E). It contains a list of Event-Condition-Action (ECA) rules expressed in natural language (Italian) according to the following the pattern (elements in angular brackets are required, the ones in square bracket are optional):

```
when   ⟨subject⟩ ⟨action⟩ ⟨object|value⟩
[if    ⟨condition⟩*]
then  (⟨subject⟩ ⟨action⟩ ⟨object|value⟩)*
```

A *subject* is either an interactive object in the game or the player. Each subject has a set of pre-defined *actions* that work on specific *objects* or require specific *values*. Updates and events are (*subject, action, object—value*) triples. They define an event if included in *when* or an update in the *then* part. Therefore, user-defined updates may trigger events having the same triple in the *when* part. The editor supports a guided rule editing, suggesting admissible values while the user types the part name. The editor reports both the type and the name for the selected interactive object (e.g., if we have a *door1* transition object in our game, the object rule part indicates it as *the transition door1*). The *conditions* are simple or composed boolean predicates on the interactive object state. The editor supports their authoring suggesting the proper predicates for the selected object, in order to avoid errors. In addition, the editor adds default rules at the object creation (e.g., an on-click rule when creating a transition), for supporting the behaviour inspection and automating repetitive user's actions.

Figure 2 shows two sample game behaviours and the correspondent definition rules. In the first one, we want to connect the *outside* scene to the *inside* scene through a door. The player has to collect a key for opening it. We need to insert a transition (*door1*) and to create an interactive area around the door in the

video. The editor inserts automatically a rule for managing the transition click and moving the player. The user specifies only the target scene, *inside* in our sample. After that, the user inserts a key object (*outside-key*) specifying the image to overlay on the 360^o video. The editor automatically adds a rule for collecting the key when clicking the associated area. Finally, the user connects the two objects adding a condition in the first rule (if part), for testing whether the player collected the key before triggering the transition. Figure 2-A shows the final interaction result and the associated rules.

In the second example, we want to create a switch that turns on and off a light in the scene *inside*. We add a switch object in the scene and we create its interactive area. The environment adds two rules that define switch behaviour: when the user clicks the *light1* switch, it turns on if it was off and vice versa. The user adds an action to both rules that change the scene background video according to the state: *h7.mp4* for the light on and *h5.mp4* for the light off (Figure 2-B). A video showing the complete procedure for defining the two sample behaviours and the resulting game is available at `http://youtu.be/P13c1-kIt-g`.

Once the user completes the game definition, she can publish it on the web. Both the authoring environment and the game engine exploit web-based technologies, so players can access games using mobile phones, tablets and personal computers. In addition, the engine supports immersive experiences through Virtual Reality Head Mounted Displays, adapting the input modality to the current device.

3 Conclusion and Future Work

In this paper, we introduced an end-user-development authoring environment for creating point-and-click videogames. Our main goal is producing a tool suitable for users without experience in game development and coding. The authoring environment allows users to create scenes and defining the game logic exploiting simple objects and rules. In the future, we want to expand the game logic further by implementing a broader range of objects for supporting more complex puzzles. In addition, we are going to implement a game debugging mode and to evaluate the tool with users.

References

1. Dey, A.K., Sohn, T., Streng, S., Kodama, J.: iCAP: Interactive prototyping of context-aware applications. In: International Conference on Pervasive Computing. pp. 254–271. Springer (2006)
2. Manca, M., Santoro, C., Corcella, L., et al.: Supporting end-user debugging of trigger-action rules for IoT applications. International Journal of Human-Computer Studies **123**, 56–69 (2019)
3. Mi, X., Qian, F., Zhang, Y., Wang, X.: An empirical characterization of IFTTT: ecosystem, usage, and performance. In: Proceedings of the 2017 Internet Measurement Conference. pp. 398–404. ACM (2017)