GAINE - tanGible Augmented INteraction for Edutainment

GAINE - tanGible Augmented INteraction for Edutainment

Andrea Bottino, Andrea Martina, Amirhosein Toosi
Dipartimento di Automatica e Informatica
Politecnico di Torino
Torino, Italy
{andrea.bottino, andrea.martina, amirhosein.toosi}@polito.it

***Abstract*— Interactive tabletops are gaining an increasing interest since they provide a more natural interaction with digital contents and allow the interaction of multiple users at a time promoting face-to-face collaboration, information sharing and the raise of social experiences. Given the potentialities offered by these devices, several entertainment-edutainment applications based on interactive tabletops have been successfully developed in different areas, from medical therapy support to children's collaborative learning, interactive storytelling and cultural heritage. However, the development of such applications often requires complex technical and implementation skills. Taking this into consideration, in this paper we present GAINE (tanGible Augmented INteraction for Edutainment), a software framework aimed at the rapid prototyping and development of interactive tabletop games. GAINE offers developers context specific high-level constructs and a simple scripting language that simplifies the implementation task. The framework is portable on different operating systems and offers independence from the underlying hardware. Two practical case studies are thoroughly discussed to show how GAINE can simplify the development of interactive tabletop applications in the entertainment and edutainment contexts.**

Keywords—Edutainment, entertainment, interactive tabletops, tangible user interfaces, toolkit

I. INTRODUCTION

In the last few years, the multi-touch tables emerged as a powerful tool to integrate interactive surfaces and responsive spaces that embody digital information, providing several advantages over conventional desktop setups. First, interactive tabletops offer users a direct and, thus, more natural interaction with on-screen contents. This naturalness is further enhanced when such devices support the use of tangible objects, recognized and tracked once placed on the table surface. Second, interactive tabletops allow the interaction of more than one user at a time, promoting collaboration, information sharing and the rise of social experiences.

A large variety of applications started deploying the benefits of using this technology, in particular to make computer applications more accessible to different categories of users, like children, users with physical or psychological disabilities and elderlies. Several research in this area, mainly focused on exploiting edutainment and entertainment applications, have been reported in the literature. Interactive tabletops games have been used to teach, improve and exercise social and communicational skills among children with autism spectrum disorders ([5][6][7]). In [8] a tangible tabletop application was developed to support children's decision making and collaboration. The use of psychophysiological measures as input technique for social tabletop gaming application intended for children was investigated in [9]. The potentialities of tabletop interactive games in supporting medical therapies have been also explored. Examples can be found in [10], which describes a game developed for children with cerebral palsy, or in [11], where VR training based on a tabletop game was proved to be a viable adjunct to conventional physical therapy in facilitating motor learning in patients with traumatic brain injuries. Other research showed the capabilities of entertainment and edutainment applications based on tabletop interactive interfaces in various fields, from classroom collaborative learning [12] to interactive storytelling [13][14], pervasive games [15] and museum exhibits [16][17].

Despite the capabilities offered by interactive tables, developing applications that fruitfully exploit them presents some inherent difficulties and requires complex technical and programming skills. In order to mitigate this problem, in this paper we present GAINE (*tanGible Augmented INteraction for Edutainment*), a software framework specifically designed to simplify the prototyping and development of interactive tabletop applications in educational and entertainment contexts.

GAINE provides an abstraction layer that hides the underlying complexity of the hardware and software libraries necessary to manage a tabletop interactive game. Furthermore, the framework offers developers high-level constructs, representing the basic application building blocks, which facilitate the implementation task. In this sense, GAINE is similar to other tabletop toolkits presented in the literature, such as [19][25][27][36]. However, we took a step forward in simplifying the application design by offering developers a reusable, semi complete application that can be completely customized through an easy-to-use, XML-style scripting language.

The rest of the paper is organized as follows. In Section II we review the current state of the art of interactive tabletops toolkits. Then, Section III presents the GAINE framework and Section IV discusses the design of two tangible games using

INTETAIN 2015, June 10-12, Torino, Italy

DOI 10.4108/icst.intetain.2015.259627

GAINE. Finally, conclusions and future works are outlined in Section V.

II. Related Works

Various toolkits have been developed explicitly to facilitate the creation of tabletop applications based on multi-touch and tangible user interfaces. They are all centered on the idea of abstracting (i) the underlying hardware level, and (ii) the complexity of the core tabletop interaction functionality, i.e. obtaining and managing the low level information related to touch and fiducial identification and tracking. These characteristics allow developers to focus on the design of their applications.

The available toolkits can be broadly divided into two categories. The first includes libraries that merely deal with the low level input processing tasks, such as image segmentation, object recognition, noise filtering and calibration. Examples of such toolkits are reacTIVision [30], an Open Source computer vision library initially developed for the Reactable [18], LightTracker [23], Touchlib [26] and Community Core Vision [24]. These toolkits usually run as standalone modules which encode and send data to an external application. The TUIO protocol [30] has become the standard de-facto for transmitting interaction information.

The second category of toolkits includes higher level frameworks. These toolkits often rely on libraries of the previous class to manage low-level task and, on top of them, provide advanced tools that simplify the rapid prototyping of applications. PyMT [25] is a Python library that provides multi-touch widgets and recognition of several multi-touch gestures. However, PyMT does not provide support for tangible interaction. Similar characteristics are offered by uTableSDK [29] and DiamondSpin [28]. More advanced features are provided by TUIO AS3 [27], an Actionscript 3 (AS3) wrapper of the TUIO protocol. Tactive [36] is a software library for the rapid development of multi-touch applications only, since it lacks support for tangible interaction. Tactive is based on a Javascript API that allows developer to use, for the implementation tasks, only web technologies and, consequently, to reuse any web application already available in a multi-touch context. ToyVision [19] is another AS3 wrapper, which is based on an extension of the reacTIVision toolkit that supports a larger set of tangible playing pieces by including in the recognition process shape and color information. An additional contribution is provided by the ROSS library, which aims at exploiting in the interaction a larger portion of Tangible User Interface space. ROSS provides an abstraction of tangible platforms, full-body interaction spaces and responsive objects (e.g. RFID tags, smartphones) that facilitates to developers their integration into any ROSS based application.

According to this taxonomy, the GAINE framework discussed in this paper falls into the higher level toolkits category, further extending the simplifications of the development process offered by other solutions.

III. The GAINE Framework

As we stated in the introduction, GAINE is a software framework aimed at supporting a rapid prototyping and development of tabletop interactive applications. Another characteristic of the framework is that it is portable on different operating systems and hardware setup.

We stress the fact that GAINE is not a general-purpose library; on the contrary, it has been specifically designed to support the development of projects with entertainment and edutainment purposes. Thus, GAINE offers developers specific high-level constructs tailored to these contexts and a simple scripting language that simplifies the implementation task (as we will show in Section IV). Taking this into consideration, in the following we will refer, without loss of generality, to any GAINE-based application as “the game”.

In this section we first briefly introduce the hardware structure used as reference for its development. Then, we provide a detailed description of the GAINE software framework.

A. GAINE Hardware: the interactive table

GAINE was tested and developed on a custom multitouch interactive table that supports user interaction through both multiple touch and tangible objects. The table surface features a rear-projection screen with a 1080p short throw projector, placed beneath the table, which allows the use of a single mirror. The overall height of the wheeled table box is 110 cm. The top surface of size 100x80 cm hosts the screen which is framed into a border that allows players to lean objects on the table.

The table features as well a secondary output, though its use is not mandatory, which can be connected to an external monitor or projector to provide either an information screen (e.g. for displaying game statistics and messages) or a different view on the game environment.

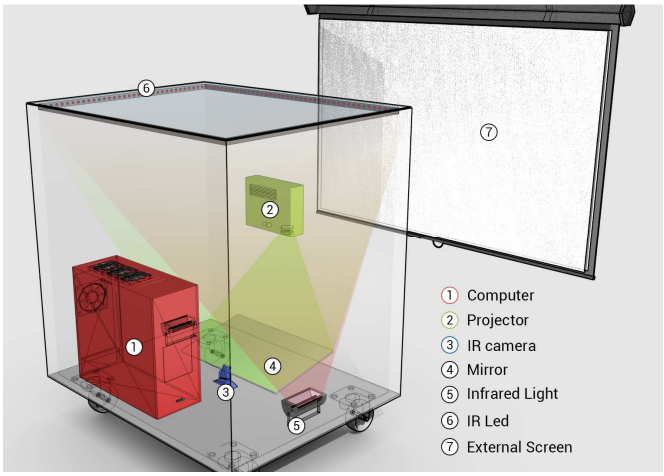

Fig. 1. A schematic of our multitouch interactive table.

The interaction on the table surface integrates both multiple touch and tangibles objects tagged with fiducials. Both touch and fiducial recognition are based on optical tracking techniques. These techniques rely on infrared light (IF) illuminating the table surface and being reflected towards an

IF-camera as soon as fingers or fiducials hits the surface. The images of the IF camera are then processed by a computer vision algorithm to track the objects of interest.

Our interactive table uses a combination of Diffused Illumination (DI) and Frustrated Total Internal Reflection (FTRI). With DI, the table surface is illuminated from IF emitters placed below the surface [21], while with FTIR [22] the lights come from the side and is transmitted into the surface material, a special Enlighten acrylic. Their combination provides a more robust tracking, since DI can reliably track the fiducials, provided that the surface is equally illuminated, and FTIR offers a better solution for tracking fast finger movements. Illuminators for both DI and FTIR have the same wavelength (850 nm).

Fiducial and finger tracking is performed by the reacTIVision toolkit. The library features as well a calibration procedure, aimed at correcting the distortions introduced by the camera optics. ReacTIVision provides a library of 216 distinct fiducials, a number which allows great flexibility in developing tabletop games. If needed by the application, the list of fiducials or their characteristics can be extended in order to deal with specific requirements, as shown in [19].

The reacTIVision trackers runs as a stand-alone module that streams captured data to the application software, according to the TUIO protocol, through an UDP channel. These data include position and ID of detected fingers and position, orientation, size, ID and class type (i.e. the symbol drawn) of detected fiducials. The combination of ID and class type allows tracking and using multiple instances of the same fiducial.

Since reacTIVision abstracts the hardware level, GAINE-based application can be translated to interactive tables based on different hardware devices. Even in case of devices that cannot offer both finger and fiducial tracking or the availability of an external display, GAINE still provides a wide set of functionalities for the development of effective tabletop games and applications.

B. The GAINE Software Framework

The GAINE framework is composed by different intercommunicating functional blocks. The main module is the GameManager, which starts the application, reads the configuration scripts, creates the digital contents, stores and evolves the application state and, finally, generates the outputs. Graphics rendering relies on OpenSceneGraph (OSG), a popular 3D engine framework. OSG is an open source scene graph API that provides high performances, advanced features, a wide range of loaders for different 2D/3D and multimedia contents, support for multi-core and multi-gpu systems and portability on different platforms. Audio rendering is managed through osgAudio, an OSG plugin that wraps OpenAL, a cross-platform audio API.

The EventManager is responsible of handling and communicating the events generated by a GAINE application. The event management system implements the classic delegate event model, characterized by event sources (controls) and event listeners (consumers), which receive the events from the sources.

The main event source is the tabletop surface, which communicates through the TUIO protocol the data related to tracked fingers and fiducials. Multiple fingers of multiple users can be tracked at once, and the objects involved by these interactions are first identified. Then, for each object, interaction data are translated into multitouch gestures and communicated to the object. Currently, only *tap*, *drag*, *pinch* and *rotate* gestures have been considered but, as future work, we are planning to handle other core gestures ([20]). The three TUIO events available for fiducials are *add*, *remove* and *transform* (move or rotate), which are redirected to the ObjectManager module, which handles digital contents related to tangible playing pieces. As for system events, user-defined events and those generated by the GUI elements, the EventManager simply acts as a collector, pairing controls and consumers.

The remaining GAINE modules are the ObjectManager, which handles the creation, destruction and state evolution of the digital playing pieces, and the WidgetManager, which creates and manages the GUI elements.

We recall that the GAINE HW features a dual screen setup: the tabletop surface, where the user interaction takes place, and a secondary screen can be used to provide additional information on the game or a different view on the game board. These two screens can be totally independent, and the decision of which contents to show and how they should be displayed on each of them is let to the developers.

Digital contents (i.e. the playing pieces and the GUI elements) can be either 3D or 2D and they can be categorized into four different classes (*game objects*, *tiles*, *agents* and *widgets*), which are detailed in the following subsections. These elements are defined in two different reference systems: the screen space (for widgets and TUIO data) and the world space (for the 2D/3D models representing game objects, tiles and agents). The transformation that relates the two reference systems is computed during the calibration step, with a procedure that relies on a specific calibration pattern containing a grid of fiducials of known size placed on the table surface.

1) Game objects

Game objects are the basic digital elements displayed by the game. Game objects can be the game board on the tabletop surface or the background 3D scenario shown on the secondary screen. Other examples of game objects are digital playing pieces and accessory elements (e.g. a digital dice). The characteristics of the game objects classes can be completely defined by the developers in a configuration script. Game objects can be made interactive and the developers can define their response to user inputs. As for their representation, one or multiple 2D or 3D models, called *switches* and described in more details in the following, can be associated to an object and updated according to rules, conditions and events. As an example, in a children game, the geometric shape or the color of a game object can be updated when the player taps on it.

2) Tiles

The *tiles* are the digital elements representing the tangible objects used as game pieces. The tangible fiducials are tracked and their position, orientation and size are used to control the tiles representation.

Each tile belongs to a class identified by the class ID of its fiducial. Several parameters characterize a tile class and its instances. One of the instance parameters is the *switch*, i.e. the 2D or 3D texturized model representing the tile in the game views. Different switches can be available for a tile class. The initial switch associated to an instance can be chosen in different ways (e.g. randomly or according to a user-defined condition) and the actual switch can be changed according to events and game rules. The switches can be animated, and their animation can be controlled in different ways (i.e. it can be started automatically, when the object is instantiated or when a specific event occurs, it can be stopped and restarted, it can be lopped or not, and so on). Switches can have any size, both related or not to the physical size of the corresponding tangible playing pieces and their motion in world space is controlled by the transform event of their fiducial.

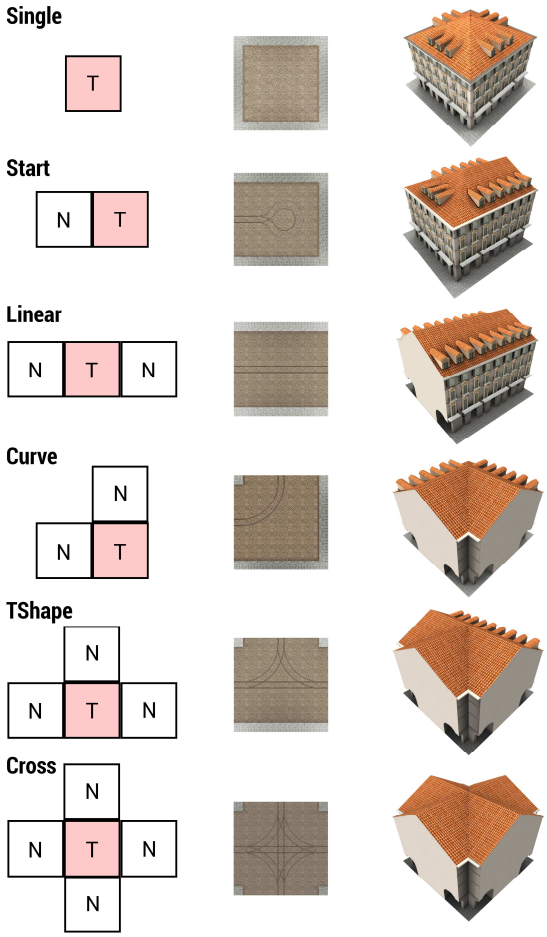

Fig. 2. Labels assigned to a grouped tile according to the number and position of neighboring Tiles (left column) and examples of label switches for two tile classes (Road, central column, and House, right column).

A tile class can also be defined as groupable. In this case, different instances can be chained together to form a compound digital model. Examples of groupable tiles can be road parts or single houses gathered into a block of flats. In order to allow a correct match between the physical layout of the tangible objects and the digital representation of the tile group, the fiducial of a groupable class is constrained to be square and the tangible base and the tile switches must have the same size of the fiducial.

Two groupable tiles are connected when the distance between their centers falls below a threshold related to the tile size. When groupable tiles are linked, a 4 connected-neighbor graph is constructed, and each group element is labeled according to the number and position of its neighbors as follows (see Fig. 2): *Start* (tile with a single neighbor), *Linear* (two neighbors on the same grid direction), *Curve* (two neighbors on different grid directions), *TShape* (3 neighbors) and *Cross* (4 neighbors). A different switch can be defined for each label and the current Tile switch is instantaneously updated at every label change, i.e. when a Tile is removed or added to the group.

Fig. 3. An example of a tile group before (up) and after (down) grid regularization.

According to developer's choices, it is possible to guarantee a seamless junction between grouped tiles, without requiring tangibles to be perfectly aligned and connected, by applying the following grid regularization:

- we create a regular grid, using as grid point distance the tile size; the grid is aligned with the XY axis of the reference system, and tiles are inserted and aligned to the grid according to their relative positions in the tile group;
- we fit the regularized grid with the actual tile group by computing the transformation that minimizes the squared distances between corresponding Tile centers;
- we update the position and orientation of the actual tile centers with those in the regularized grid.

An example of grid regularization is shown in Fig. 3.

3) Agents

The *agents* are autonomous virtual characters that can be generated by the main scene (i.e. the game board or the 3D scenario on the secondary screen) and by single or grouped tiles. For instance, if the game scenario represents a city and

some of the tiles are house blocks, they can generate as agents the city dwellers, which are then free to move around the city according to specific navigation rules. If the tiles represent streets, they can generate cars which are constrained to move on the correct street lane (Fig. 4).

Fig. 4. Examples of autonomous agents generated by the playing pieces.

The agent characteristics can be again defined by the developers. Similar to tiles, each agent is represented by a switch, which can be selected, in various ways, among a set of possible choices. Agents are provided with the capability to navigate the environment in a "life-like and improvisional matter". This capability relies on the so-called "steering behaviours" defined in [4]. Different behaviours can be combined and assigned to an agent and, eventually, the agent behaviour can be modified by events/conditions. Actually, we provide a set of basic steering behaviours that extend those already available in the OpenSteer library [33]. Nevertheless, new type of behaviours can be implemented and offered to the application designer according to his/her specific needs.

The minimal/maximal number of instances per class and the agent generation rules and rates can be defined by developers. Agents can be optionally destroyed according to their life-time, as a consequence of a system event or when the generating tile/tile group has been removed from the table surface. Other parameters associated to agents are their maximal speed, the list of obstacles to avoid during navigation (e.g. a list of game object, tile and agent classes) and a set of values to control their animation. When agents are generated by a groupable tile, the developer can also decide the minimal number of Tiles in the group that allows the generation of agents. This parameter can be used, for instance, to avoid adding cars to a street composed by only two blocks.

4) Widgets

The widgets are the graphical control elements that can be used to create the game GUI. Examples of widgets available are buttons, sliders, labels, image and video widgets, dialog or message windows. Widgets, according to the specific widget type, offer different pre-defined properties and generate events upon user interaction.

The framework includes a peculiar type of widgets, called Sensible Areas, which are (invisible) polygonal regions defined on the tabletop surface that raise an event when a fiducial moves within, enters or exit their area. The event data (i.e., position, instance and class ID of the fiducial that generated the event), can then be used by the application to trigger specific actions, as we will show in the examples in Section IV.

C. Game logic management

The management of the game logic is based on three elements: *parameters*, *events* and *rules*.

Parameters are variables associated to various elements, like the system, game objects, tiles, agents and widgets. System variables include, for instance, the system time and the execution time, and tile parameters can be their position, orientation and current switch. For game objects, tiles and agents, class parameters are available as well, such as the number of created instances. Developers have the possibility to create their own parameters and associate them to any element of the application. It is also possible to define parameters as functions of other parameters. Any element making use of a parameter (i.e. other parameters, GUI labels showing its value or conditions to be checked) is immediately updated when the parameter value is modified.

The rules are actions executed according to an optional conditional statement. This condition, if present, can be expressed as a function of the parameter values. Within a rule, different possible actions can take place: an update of one or more parameters, the raise of an event, the play of a sound or the creation of different elements (e.g. a game object, a message window, a video widget, and so on). Rules are executed when any of their parameters has been modified. Clearly, in the case a rule is controlled by a condition, it is executed only if a parameter involved into the condition has been updated. Finally, developers can assign rules a priority, an integer value defaulted to one, and rules are executed in descending priority order.

Events are actions that can be detected by the program. Events can be predefined by the GAINE framework (e.g. interaction events) or defined by the developers and associated to any GAINE element. For instance, an event can be generated by a tile class when the number of instances exceeds or drops below a predefined threshold.

Any GAINE element, including the system, can register to an event and define an event handler, which consists in a set of rules that are executed only when the event is received.

In Section IV, we will show with a simple example how to use all these elements to define the logic of a tabletop interactive game.

D. Dual screen management

The game view on both available displays can be controlled by the developers. Each display can define its own base scenario (e.g. a 2D image/model for the game board on the tabletop surface and a 3D environment for the secondary screen) and a camera, which can provide a 2D or a 3D view, either parallel or perspective. The camera parameters can be defined in advance and modified during the game. As an alternative, multiple cameras can be defined and switched according to events or conditions. In a similar way, it is possible to define multiple lights, activate/deactivate them and modify their parameters.

As previously stated, game objects, tiles, agents and widgets can be displayed, according to the developers' choices, on one or both the display screens. For instance, in an

augmented chess game, the tabletop surface can display both a chessboard and the GUI elements to control the game, such as a chess game clock and, if the game is subject to time control, buttons to stop one player's clock and start the other. The tangible chess pieces can be associated to fiducials and represented as (possibly animated) 3D models on a 3D virtual chessboard in the secondary display, allowing the game audience a better comprehension of the game evolution (Fig. 5).

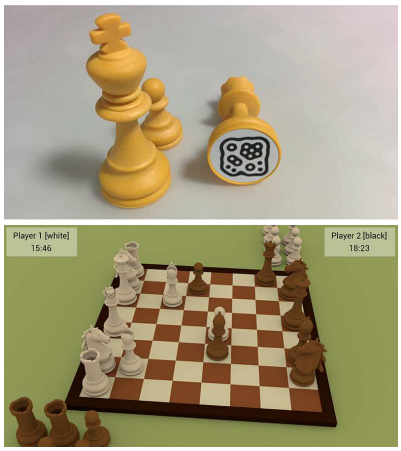

Fig. 5. A chess game developed with GAINE: the playing pieces and their associated fiducials (up) and a snapshot of the game view displayed on the secondary screen (down).

While with our dual screen setup, the user interaction is clearly possible on the tabletop surface only, GUI elements can be exploited on the secondary display as well. For instance, it is possible to display informative labels or message windows that can be hidden when a timer expires.

E. Portability

GAINE has been implemented in C++. As stated in the introduction, one of the goals in the development of GAINE was to provide a framework that is portable on different operating systems and hardware setup.

In our project, the portability issue has been tackled exploiting in the development of the framework only mature, robust and well know portable Open Source software libraries, which are briefly listed in the following:

- reacTIVision [30] for touch recognition and fiducial tracking, which offers as well the abstraction from the underlying hardware;
- OSG [31], the graphics rendering engine;
- osgAudio and OpenAL [32] for audio rendering;
- OpenSteer [33], to manage the navigation of GAINE autonomous agents;
- muParser [34], to parse the mathematical expressions used by parameters, conditions and rules;
- Cmarkup [35], for script parsing.

IV. Case Studies and Discussion

The GAINE framework has been used to develop several entertainment/edutainment applications, ranging from simple games to interactive stories for kids. Some of these applications have been briefly sketched in the previous sections. Here we describe in more details two case studies and we report some data to show how the use of GAINE impacts the development of a multitouch, augmented tabletop application.

The first example discussed is a tabletop augmented version of the classic Tic-tac-toe game. We have chosen this case study since it is simple enough to allow us to briefly discuss its whole implementation and, at the same time, to demonstrate the effectiveness of GAINE in supporting a rapid application development. Tic-tac-toe is a pencil-and-paper game for two players which take turns marking a 3x3 grid with their own symbol (either a "X", cross, or a "O", naught). The player who first succeeds to place three symbols on a row, column or diagonal wins the match. Otherwise, if all marks have been placed without a winner, the game ends with a draw.

Developing the tabletop version of Tic-tac-toe requires to create the digital contents to be displayed (e.g., the game board and the application GUI) and the scripts managing the game flow. In the following, we will show some excerpt from these scripts to provide insights on the development process.

The game board is a 2D textured plane showing the 3x3 playing grid (Fig. 6(a)). In this example, the secondary screen simply shows a 3D perspective view of the game board. The tangible playing pieces are the noughts and crosses, which are marked with two different fiducials (Fig. 6(b)). Their corresponding 3D tiles, shown on the secondary screen (Fig. 6(c)), are configured as follows:

```
<tile type="Nought" fiducial="105" />
<switch display="2" model="models/nought.dae" animated="true" loop="true"/>
</tile>

<tile type="Cross" fiducial="205" />
<switch display="2" model="models/cross.dae" animated="true" loop="true"/>
</tile>
```

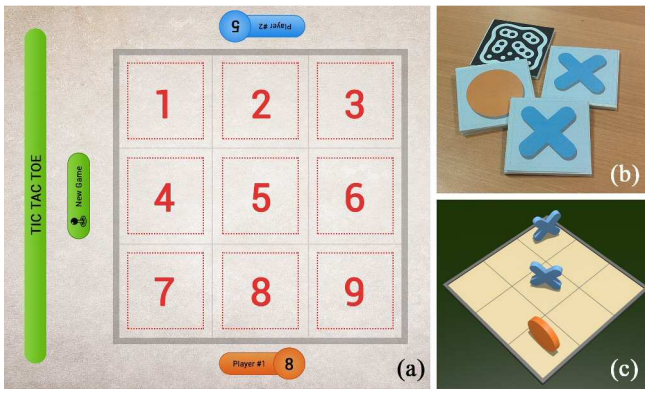

Fig. 6. (a) The Tic-tac-toe tabletop interface, showing the playing grid (where the dotted numbered squares highlight the sensible areas, which are not actually visible), the GUI labels and the start button; (b) the tangible playing pieces; (c) the 3D game view on the secondary screen

In order to manage the game, the following system parameters and events were defined:

```
<parameters>
  <parameter  name="winPlayer1" type="int" value="0"/>
```

```
  <!-- player1 = cross -->
  <parameter  name="winPlayer2" type="int" value="0"/>
  <!-- player2 = nought -->
  <parameter  name="gameActive" type="bool" value="true"/>
  <parameter  name="gameResult" type="int" value="-1"/>
  <!--  0 draw, cross: wins player1, nought: wins player2 -->
  <parameter  name="grid[9]" type="int" value="{0,0,0, 0,0,0, 0,0,0}"/>
  <parameter  name="playedPieces" type="int" value="0"/>
</parameters>

<eventList> <event name="OnGameFinished"/> </eventList>
```

The `grid` parameter is a vector of nine elements that records the playing pieces placed on the grid cells, while `playedPieces` counts the overall number of symbols played. `GameResult` stores the result of the current game, `winPlayer` parameters record the players' scores and `gameActive` indicates if a game is in progress.

Then, we defined a set of rules that check: (i) if all the cells in a row, column or diagonal contain the same symbol, and (ii) if the maximal number of symbols has been played, possibly resulting in a draw. If one of these conditions is verified, the game result is recorded and the `OnGameFinished` event is raised:

```
<ruleList>
  <!-- check possible win in the first grid row -->
  <rule condition="gameActive and grid[1] == grid[2] and grid[2] == grid[3]
and grid[3] != 0">
    <onTrue action="gameActive = false, gameResult = grid[3]"
          emitEvent="OnGameFinished" />
  </rule>
<!– we do not report, for the sake of brevity, the similar rules that check
different combinations-->

  <!-- check draw, N.B.: this rule has lower priority than win checks -->
  <rule condition="playedPieces == 9" priority="0">
    <onTrue  action = "gameActive = false, gameResult = 0"
          emitEvent ="OnGameFinished" />
  </rule>
</ruleList>
```

Each cell of the playing grid is associated with a sensible area that raises an event when a playing piece is placed on (`OnSensibleAreaEntering`) or removed from (`OnSensibleAreaLeaving`) the grid cell. Each sensible area defines an extra parameter `cell`, containing the index of the grid cell it is associated with, which will be used by the event handlers. The sensible areas are defined as follows:

```
<sensibleAreaList>
  <sensibleArea name="c1" vertices(386,184),(626,184),
          (626,424),(386,424)">
    <parameter name="cell" type="int" value="1">
  </sensibleArea>
<!– other cells from c2 to c8... -->
  <sensibleArea name="c9" vertices="(978,776),(1218,776),
          (1218,1016),(978,1016)">
    <parameter name="cell" type="int" value="9">
  </sensibleArea>
</sensibleAreaList>
```

The system registers itself as listener of the sensible areas events. The event handler updates the symbol of the played piece to the proper entry of the `grid` vector and the number of pieces played. The update of these parameters causes the rules to be evaluated. If one of the player wins, or there is a draw, the game is stopped, the current score is updated and a message window is displayed. A "New game" button on the tabletop surface allows, when pressed, to reset the values of all grid cells and start a new game.

The code of the system event handlers is the following:

```
<eventListeners>
  <event name="OnSensibleAreaEntering">
    <rule action="grid[sender.cell]=sender.tileClass,
          playedPieces = playedPieces + 1" />
  </event>
  <event name="OnSensibleAreaLeaving">
    <rule action="grid[sender.cell]=0, playedPieces=playedPieces-1" />
  </event>
  <event name="OnGameFinished">
    <rule action="gameActive = false, newGameButton.enable = true"/>
    <rule condition="gameResult == 0">
      <onTrue playSound="draw.wav"/>
    </rule>
    <rule condition="gameResult == Cross.fiducial">
      <onTrue action="winPlayer1 = winPlayer1 + 1" playSound="win1.wav"/>
    </rule>
    <rule condition="gameResult == Nought.fiducial">
      <onTrue action="winPlayer2 = winPlayer2 + 1"  playSound="win2.wav"/>
    </rule>
  </event>
  <event name="OnButtonClicked">
    <rule condition="sender == newGameButton">
      <onTrue action="grid[] = {0,0,0, 0,0,0, 0,0,0}, gameResult = -1,
gameActive = true, newGameButton.enable = false, playedPieces = 0" />
    </rule>
  </event>
</eventListeners>
```

The widget configuration file[1] creates the labels to display the players' scores and the "New Game" button, which is disabled at game start and raises, if active and clicked, a default event `OnButtonClicked`.

```
<labelList>
  <label name="PlayerOneLabel" icon="img/labpone.jpg" position ="670,1170"
     size="270,100" orientation ="south" displayedParameter="winPlayer1">
    </label>
  <label name="PlayerTwoLabel" icon="img/labptwo.jpg" position="940,18"
     size="270,100" orientation="north" displayedParameter="winPlayer2">
  </label>
</labelList>


<buttonList>
    <button name="newGameButton" position="268,732" size="270,70"
          orientation="west" label="New Game" icon="img/newgame.jpg"
          onClick="img/newgameClick.jpg" sound="startgame.wav"
          enable="false" />
</buttonList>
```

Summarizing, the game execution flow is the following:

- when a player places one of his/her symbols on a grid cell, an event is raised causing: (i) an update of the cell value, (ii) an increment of the number of pieces played, and (iii) an evaluation of the the rules including one of these parameters in their conditions;
- eventually, the `OnGameFinished` event is raised, halting the game, updating the scores and the labels showing them, playing a proper sound and activating the "New Game" button;
- when the "New Game" button is pressed, game variables are cleared, the button is deactivated and a new match can start.

The whole development process of the game (which includes digital content and tangible pieces creation, script writing, testing and debugging) required a total of five man-hours. We think this number highlights the capabilities of our framework to support a rapid application prototyping.

The second example discussed is "Torino 150", an edutainment application designed for children aged between 6 and 13 (i.e. students of first and middle schools). Torino 150 has a gameplay similar to that of the well-known SimCity

[1] For the sake of clarity, we report a simplified version of the script, where we removed most of the parameters related to the definition of the widget appearance

game ([38]). It is a cooperative game in which players have the task of founding and developing their own version of the city of Torino in 1861, the year when the city became the capital of the newly proclaimed united Kingdom of Italy.

The game is aimed at communicating to children a piece of the history of their own city, along with information related to buildings and monuments that have changed their appearance through time or that do not exist anymore. The digital contents and the information displayed are the legacy of ToViA (Torino Virtual Adventure), a Virtual Heritage project aimed at constructing a realistic and historically accurate 3D model of Torino in 1861, whose demonstrative 2D and 3D interactive visualization were shown during the Italia 150 celebration, held in 2011. The information used for creating these digital models were collected from various sources: photos of still existing buildings and monuments, archival material such as maps, photographs, drawings, paintings and other historical documents (Fig. 7).

Fig. 7. Examples of models used by Torino 150 application and of reference documents used to create them.

The aim of the game is to reach the highest level of "welfare" for citizens, where the welfare represents a balance among positive (e.g. presence of hospitals, markets, public transportation) and negative factors (e.g. criminality, pollution, taxes). The game has three difficulty levels, related to the level of taxes (low, medium or high), which can be selected by the players.

The tangible playing pieces represent the different buildings of the city, such as houses, markets, police and fireman stations, industries and streets. As a reward for reaching certain welfare levels, the theatre, cathedral and royal palace blocks are released. When playing pieces are placed on the table, they contribute to modify some of the parameters involved into the computation of the current welfare. For instance, the population increases with the number of houses, the health with the number of hospitals and the criminality is a function of population, richness and number of police stations.

Different types of agents are considered in the game: citizens, spawned by houses, and horse-drawn trams and carriages created by streets (see Fig. 4). Some system events were also introduced. For instance, each hour of gameplay (virtually corresponding to 12 hours in real life) the cathedral, if placed on the table, rings the bell to announce a religious function causing citizens to approach the church. As another example, an earthquake (raised either randomly by the system or explicitly by players with a GUI button) can cause some houses to collapse in ruins and a population decrease.

Opposite to the previous example, this application fully exploits the availability of a secondary screen, which allows players to be immersed in a 3D reconstruction of the virtual city they are building in the game (Fig. 8). This virtual environment can be viewed under different perspectives, namely through: (i) a set of fixed cameras, providing an overview of the city, (ii) a first-person camera whose movements can be controlled by the user, and (iii) a first-person camera attached to any of the citizens moving in the environment. The current viewing camera can be selected and updated through the game GUI.

As for the development time of the whole application, it is virtually impossible to provide an accurate estimate for two main reasons. First, as previously stated, most of the digital contents used by the application were based on those developed for a previous project. The only modeling effort was the adaptation of these models to the needs of a real-time VR application. Second, Torino 150 was one of the main test cases used as reference for the development of GAINE. Thus, the development of the application proceeded in parallel with the implementation of the features and functionalities required by the framework. Nevertheless, we think that the amount of code required by the application scripts to configure contents and implement the game logic, which sums up to 513 lines only, can provide an indication of the simplicity of developing an interactive tabletop game with GAINE.

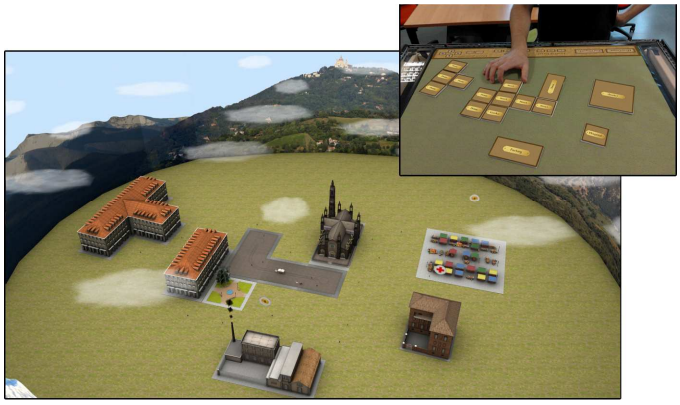

Fig. 8. Torino 150 games: the playing pieces and a screenshot of the view on the secondary screen.

V. Conclusions

In this paper we presented GAINE, a flexible framework for the rapid prototyping and development of tabletop games exploiting tangible interaction on multi-touch interactive tabletops. The main contribution of GAINE is to provide a set of high level structure and a simple scripting language that allows specifying (i) the characteristics of the digital contents managed by the application and (ii) the definition of the game logic. Examples showing the effectiveness of using GAINE in the development of tabletop interactive applications for entertainment and edutainment purposes were provided and discussed.

The framework has been implemented in C++ relying on a set of open source libraries that provide an abstraction from the hardware level and the portability on different platforms.

As for future work, we are planning to expand the functionalities offered by GAINE. As we did in the previous development steps, this process will be carried out by tackling novel projects and novel application scenarios, analyzing the new challenges they require to face and designing solutions for them. Then, following the ideas discussed in [19] and [37], we are also planning to enhance the tangible interaction component, which is currently limited to the use of fiducial-tagged objects, by providing support for a more varied set of playing pieces and responsive objects. Finally, despite the simplicity of the scripting language used by the framework, the design process would sorely benefit from the availability of an IDE (similar to the Graphic Assistant described in [19]), which would provide developers an easier modeling of the game elements and an automation of the script generation.

ACKNOWLEDGMENT

We thank Matteo De Simone for assembling the table hardware during his doctoral thesis and Daniele Argiolas for his contribution to code development.

REFERENCES

[1] M. Kaltenbrunner, S. Jorda, G. Geiger and M. Alonso, "The reactable*: A collaborative musical instrument," In *WETICE 2006: Proceedings of the 15th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pp. 406–411.

[2] E. Tse, S. Greenberg, C. Shen and C. Forlines, "Multimodal Multiplayer Tabletop Gaming," In *PerGames 2006: Proceedings Third International Workshop on Pervasive Gaming Applications*, pp. 139–148.

[3] T. Gross, M. Fetter and S. Liebsch, "The cuetable: cooperative and competitive multi-touch interaction on a tabletop," In *CHI 2008 Extended Abstracts on Human Factors in Computing Systems*, pp. 3465–3470.

[4] C. W. Reynolds, "Steering Behaviors For Autonomous Characters," In *Proceedings of Game Developers Conference*, San Jose, California, 1999, pp. 763-782.

[5] A. Battocchi, A. Ben-Sasson, G. Esposito, E. Gal, F. Pianesi, D. Tomasini, P. Venuti, P. Weiss, and M. Zancanaro, "Collaborative puzzle game: a tabletop interface for fostering collaborative skills in children with autism spectrum disorders," *Journal of Assistive Technologies*, 2010, 4(1), pp. 4-13.

[6] G. F. M. Silva, A. Raposo and M. Suplino, "Par: A collaborative game for multitouch tabletop to support social interaction of users with autism," *Procedia Computer Science*, 27, 2014, pp. 84-93.

[7] R. Zarin and D. Fallman, "Through the troll forest: exploring tabletop interaction design for children with special cognitive needs," In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM, 2011, pp. 3319-3322.

[8] C. McCrindle, E. Hornecker, A. Lingnau and J. Rick, "The design of t-vote: a tangible tabletop application supporting children's decision making," In *Proceedings of the 10th International Conference on Interaction Design and Children*, 2011, pp. 181-184.

[9] A. Al Mahmud, O. Mubin, J. R. Octavia, S. Shahid, L. Yeo, P. Markopoulos and J. B. Martens, "aMAZEd: designing an affective social game for children," In *Proceedings of the 6th international conference on Interaction design and children*, 2007, pp. 53-56.

[10] Y. Li, W. Fontijn and P. Markopoulos, "A tangible tabletop game supporting therapy of children with cerebral palsy," *Proceedings of the 2nd Intl. Conf. on Fun and Games*, 2008 pp. 182-193.

[11] J. Duckworth, P. R. Thomas, D. Shum and P. H. Wilson, "Designing co-located tabletop interaction for rehabilitation of brain injury," *Lecture Notes in Computer Science*, Vol. 8013, 2013, pp 391-400.

[12] S. E. Higgins, E. Mercier, E. Burd and A. Hatch, "Multi-touch tables and the relationship with collaborative classroom pedagogies: A synthetic review," *International Journal of Computer-Supported Collaborative Learning*, 6(4), 2011, pp. 515-538.

[13] X. Cao, S.E. Lindley, J. Helmes, A. Sellen, "Telling the whole story: Anticipation, inspiration and reputation in a field deployment of TellTable," *Proceedings of CSCW 2010, ACM Conference on Computer Supported Cooperative Work*. p. 251-260.

[14] T. Alofs, M. Theune, and I.M.T. Swartjes, "A Tabletop Board Game Interface for Multi-User Interaction with a Storytelling System," *Proceedings of 4th International Conference on Intelligent Technologies for Interactive Entertainment, INTETAIN 2011*, pp. 123-128.

[15] A. Wu, D. Joyner and E.Y.L. Do, "Move, beam, and check! imagineering tangible optical chess on an interactive tabletop display," *Computers in Entertainment (CIE)* 8, no. 3, 2010: 20.

[16] M. Horn, Z. Atrash Leong, F. Block, J. Diamond, E. M. Evans, B. Phillips and C. Shen, "Of BATs and APEs: an interactive tabletop game for natural history museums," In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2012, pp. 2059-2068.

[17] N. Correia, T. Mota, R. Nóbrega, L. Silva, A. Almeida, "A multi-touch tabletop for robust multimedia interaction in museums," In *ACM International Conference on Interactive Tabletops and Surfaces*, 2010 pp. 117-120,

[18] S. Jordà, G. Geiger, M. Alonso and M. Kaltenbrunner, "The reacTable: exploring the synergy between live music performance and tabletop tangible interfaces," In *Proceedings of the 1st international conference on Tangible and embedded interaction*, 2007, pp. 139-146.

[19] J. Marco, E. Cerezo and S. Baldassarri, "ToyVision: a toolkit for prototyping tabletop tangible games," In *Proceedings of the 4th ACM SIGCHI symposium on Engineering interactive computing systems*, 2012 pp. 71-80.

[20] C. Villamor, D. Willis, L.Wroblewski,"Touch Gesture Reference Guide. 2010. Available online at: static.lukew.com/TouchGestureGuide.pdf, last accessed March 2015

[21] N. Matsushita and J. Rekimoto, "HoloWall: designing a finger, hand, body, and object sensitive wall," In *Proceedings of the 10th annual ACM symposium on User interface software and technology (UIST '97)*, 1997 pp. 209-210.

[22] Y.J. Han, "Low-cost multi-touch sensing through frustrated total internal reflection," In *Proceedings of the 18th annual ACM symposium on User interface software and technology (UIST '05)*, 2005 pp. 115-118.

[23] A. Gokcezade, J. Leitner, and M. Haller, "LightTracker: An open-source multitouch toolkit," *ACM Comput. Entertain.* 2010, 8, 3, pp. 16.

[24] Community Core Vision, available online at ccv.nuigroup.com, last accessed March 2015

[25] T. Hansen, C. Denter and M. Virbel, "Using the PyMT toolkit for HCI Research," *Forum on Tactile and Gestural interaction*, 2010.

[26] Touchlib, available online at nuigroup.com/touchlib, last accessed March 2015

[27] J. Luderschmidt, I. Bauer, N. Haubner, S. Lehmann, R. Dörner and U. Schwanecke, "TUIO AS3: A Multi-Touch and Tangible User Interface Rapid Prototype Toolkit for Tabletop Interaction," In *Self Integrating Systems for Better Living Environments: First Workshop, Sensyble*, 2010, pp. 21-28

[28] C. Shen, F. D. Vernier, C. Forlines and M. Ringel, "DiamondSpin: an extensible toolkit for around-the-table interaction," In *Proceedings of the SIGCHI conference on Human factors in computing systems*, 2004, pp. 167-174.

[29] uTableSDK. Available online at http://utablesdk.codeplex.com, last accessed March 2015

[30] ReacTIVision, a toolkit for tangible multi-touch surfaces, available online at http://reactivision.sourceforge.net/, last accessed March 2015

[31] OpenSceneGraph, available online at www.openscenegraph.org, last accessed March 2015

[32] osgAudio, OpenSceneGraph nodekit available at code.google.com/p/osgaudio/, last accessed March 2015

[33] OpenSteer, Steering Behaviors for Autonomous Characters, available online at opensteer.sourceforge.net, last accessed March 2015

[34] muParser, Fast Math Parser Library, available online at muparser.beltoforion.de, last accessed March 2015

[35] CMarkup, available online at www.firstobject.com/dn_markup.htm, last accessed March 2015

[36] O. Gaggi and M. Regazzo, "Tactive, a Framework for Cross Platform Development of Tabletop Applications," In *International Conference on Web Information Systems and Technologies (WEBIST 2014)*, pp. 91-98.

[37] A. Wu, J. Jog, S. Mendenhall, A. Mazalek, "A framework interweaving tangible objects, surfaces and spaces," In *Human-Computer Interaction. Interaction Techniques and Environments*, 2011, pp. 148-157.

[38] SimCity, from Wikipedia http://en.wikipedia.org/wiki/SimCity, last accessed March 2015