ToyVision: A Toolkit for Prototyping Tabletop Tangible Games

Javier Marco
Madeira-ITI
University of Madeira, Portugal
javier.marco@m-iti.org

Eva Cerezo, Sandra Baldassarri
Advanced Computer Graphics Group (GIGA)
Computer Science Department,
Engineering Research Institute of Aragon (I3A)
University of Zaragoza, Spain
{ecerezo, sandra}@unizar.es

ABSTRACT
This paper presents "ToyVision", a software toolkit aimed to make easy the prototyping of tangible games in visual based tabletop devices. Compared to other software toolkits which offer very limited and tag-centered tangible possibilities, ToyVision provides designers and developers with intuitive tools for modeling innovative tangible controls and with higher level user's manipulations data. ToyVision is based on Reactivision open-source toolkit, which has been extended with new functionalities in its Hardware layer. The main design decision taken has been to split the Widget Layer from the lower abstraction layers. This new abstraction layer (the Widget layer) is the distinguishing feature of ToyVision and provides the developer with access to a set of encapsulated classes that give the status of any playing piece handled in the tabletop while the game is running. The toolkit has been complemented with a Graphic Assistant that gathers from the designer all the information needed by the toolkit to model all the tangible playing pieces. As a practical example, the process of prototyping a tangible game is described.

Author Keywords
Tabletop; toolkit; tangible; games; playing pieces; widget; architecture

ACM Classification Keywords
H5.2. [Input devices and strategies]: User Interfaces

General Terms
Design.

INTRODUCTION
Horizontal computer-augmented surfaces (tabletops) enable simultaneous and co-located access to digital content to multiple users around the table. Until recently, the use of these devices were restricted only to research environments [29] [31] [9], but now private companies are offering tabletop solutions [26], and also there is a growing community of hobbyist designers of tabletops [27] thanks to new affordable hardware techniques [34]. Games and entertainment emerge as very promising applications for these devices, since tables are popular spaces for social games due to their physical affordances that engage face to face interaction between players [32].

Recent expansion of tabletop devices gives rise to a new generation of physical and social videogames which mix traditional board games with the new possibilities of digitally augmenting the area of interaction with computer image and audio. Most of the tabletop games are based on multitouch interaction [6] [2] in which playing pieces are virtually projected on the surface and players manipulate them dragging their fingers on the table.

Several tabletop devices are not only capable of detecting user fingers and hands, but also of supporting the identification and tracking of conventional objects placed on the active surface. Thanks to this functionality, physical tabletop games based on tangible interaction are also feasible [11] [20]. By keeping playing pieces in the player's physical environment, emotional impact of videogame is reinforced [16] [14] and digital technology becomes accessible to other user profiles such as very young children [24], users with disabilities [21] and the seniors [1].

On the other side, the creation of a tabletop game usually implies to "hardcode" complex algorithms to process raw data from tabletop in order to detect and track each playing piece manipulated on the active surface. This situation brings a breach between tangible interaction design and the corresponding implementation tasks, i.e., between designers and developers. To tackle the problem, several software toolkits are emerging with the aim of isolating developers from tabletop hardware, so that they can implement their application in a higher abstraction level. These toolkits offer the developer processed data of users' interactions on the table, both tactile and through objects, but unfortunately, for the moment, in a very basic form: tangible interaction is described though simple events (object placed, moved or removed). This simplistic approach is constraining the designer to use playing pieces

EICS'12, June 25–26, 2012, Copenhagen, Denmark.

that can be just moved on the table, and therefore, this situation limits the exploration of more rich tangible interaction possibilities.

This paper proposes ToyVision, a toolkit for the rapid prototyping of tangible tabletop games. ToyVision can be seen as an expansion to already existing and in development "simplistic" toolkits based on TUIO [16] (such as reacTIVision [30] or CCV [5]), created to facilitate designers and developers the implementation tasks. This is done by supplying them with high processed data centered in the function that every playing piece plays in the game. ToyVision proposal is based on Reactivision open-source toolkit [18], which has been extended with new functionalities which are oriented to the rapid prototyping of tabletop games in Action Script 3 (AS3) development environments (Adobe Flash, Air and Flex). Nevertheless, the approach exposed in this paper can be easily translated to other tabletop toolkits and other development environments different from Reactivision and AS3.

The paper first goes through the current state of tabletop toolkits, and then proposes a classification of the different playing pieces to be used in board games. Next, ToyVision toolkit is presented, followed by a comparison of the design of a tangible game with and without the use of ToyVision. Finally, conclusions and future work are outlined.

RELATED WORK

Due to the recent success of multitouch devices, several toolkits have emerged aimed to implement applications in the most popular development environments with independence of the hardware. While earlier multitouch toolkits merely informed developers about raw-tactile events (finger added, moved, left from the table) [36] [3] [22] [35] [23], recent toolkits isolate finger gesture recognition from developers, by sending high abstraction events (zoom, rotation, delete…) [13] [12] [28].

The addition of tangible interaction functionalities to tabletop surfaces requires identifying and tracking conventional objects placed on the interactive surface area. In visual based multitouch surfaces [34] this can be achieved by attaching a printed visual tag (fiducial) [7]. Fiducial recognition is based on a simple principle: a fiducial is composed of infrared light (IR) reflective and non-reflective areas, and the visual software detects the reflective areas as white blobs. Each fiducial has a unique distribution of blobs, so it is possible to distinguish different fiducials, and also to track their position and orientation on the tabletop surface. Using this technique, several multitouch tabletop toolkits are also supporting interaction with tagged objects [4] [30] [5] [37].

Software architecture of tabletop toolkits has been described by Echtler and Klinker [10] using four layers, from lowest to highest abstraction:

- *Hardware*
- *Calibration*
- *Event Interpretation*
- *Widget*

A toolkit that follows a layered architecture (see fig. 1) offers, at least, the Hardware layer in order to hide the visual hardware and blob recognition algorithms. Optionally, the toolkit can add the Calibration layer to correct the position coordinates of each detected blob due to camera optics aberrations. With the Event Interpretation layer, the toolkit keeps track of blob events (added, moved or removed from the tabletop surface). Finally, by adding the Widget layer, the toolkit may associate sequences of events in tabletop regions with predefined actions in the tabletop application.

As toolkits include more abstraction layers, developers receive higher abstraction processed data from user interactions. By separating the toolkit from the developing environment, tabletop applications can be translated to other devices based on different hardware and even different toolkit. This is provided by the use of standard communication protocols between the toolkit and the development environment. In this context, the TUIO protocol [16] has become very popular and has been adopted by most tabletop toolkits [30] [5] [38] [36] [22]. However, TUIO protocol is designed to transmit processed data from the Event Interpretation Layer (EIL): the toolkit sends, embedded in TUIO packets, multitouch and tangible events to the tabletop application (see fig. 1). TUIO support of tagged objects are limited to three simple events (add, remove and move/rotate tagged objects) (see fig.1 left). Although the soon expected launch of the TUIO 2.0 specification is announcing a better support for tangible tabletop applications, this will actually consist in the support of untagged objects [18], keeping toolkits that will support TUIO 2.0 in the same EIL architecture.

There are also some toolkits specific for Tangible User Interfaces which isolate developers from the intrinsic complexity of managing several kinds of sensors hidden in objects. Most of these toolkits use an EIL approach [19], but also there are some interesting proposals of tangible toolkits with a Widget Layer architecture [8]. The work here presented contributes to the state of the art of tabletop toolkits with the addition of a Widget layer in a toolkit based on an EIL architecture in order to support advanced tangible interaction with playing pieces in tabletop games with a high abstraction development approach.

The challenge of designing a toolkit which includes a Widget abstraction layer relies in the huge collection of different existing objects that can be placed on a table. Nevertheless, this work focuses in board games and, therefore, the range of playing pieces should be approachable, as it will be detailed in next section.

CLASSIFYING PLAYING PIECES IN BOARD GAMES

Holmquist et al. [15], pioneers on modeling the relationship between physical objects and an ubiquitous computer system, proposed the term Tokens to describe any object used to represent some stored digital information. Later, Ullmer and Ishii [39] refined the Token definition for Tangible User Interfaces (TUI), and additionally proposed a new kind of object, a constraint; both were defined and related as following:

- Token: any piece that can be placed, moved and removed to interact with the application.
- Constraint: any physical area in which tokens are restricted in translation or rotation.

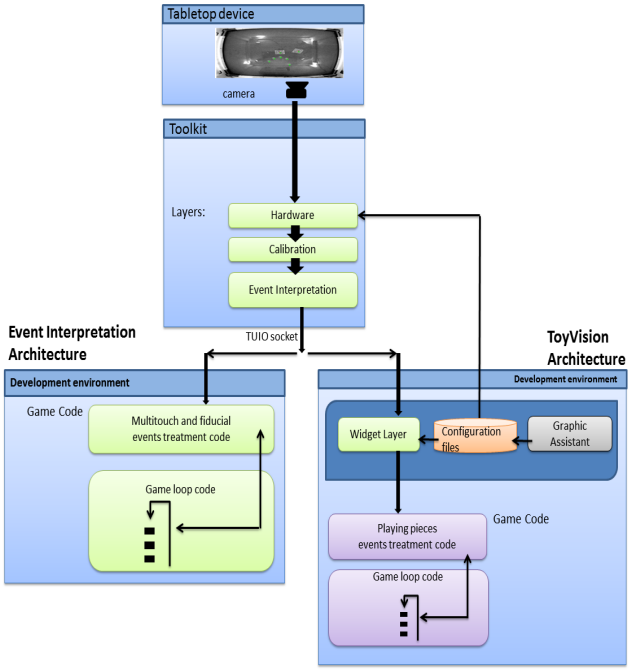

Figure 1. Tabletop toolkits architecture. Left: Current toolkits's architecture characterized by only three layers of abstraction. Right: ToyVision toolkit architecture.

Two kinds of relationships can emerge between Tokens and Constraints to define the kind of interaction that the user is able to carry out with the tokens in the limits of the constraint:

- Associative: The user is limited to place and remove the tokens inside the constraint area.
- Manipulative: the user is able to manipulate (move and rotate) the tokens inside a limited area of the constraint.

Later on, Shaer and Jacob [33] expanded this work, by proposing a Tangible User Interface Description Language (TUIDL) also based on tokens and constraints, so that any kind of object involved in a TUI could be modeled in UML, and translated into a XML specification. Our proposed Widget layer can be seen as a practical implementation of a TUIDL for tangible tabletops in which every playing piece is automatically modeled in an XML specification, as it is explained in next section.

Starting from precedent classifications and trying to adapt them to the tabletop tangible context, we propose four categories of playing pieces: Simple Tokens, Named Tokens, Constraint Tokens and Deformable Tokens. These categories try to cover the wider spectrum of possible playing pieces that may be used in tangible board games (in a very broad sense, considering any ludic activity that could be played on a table such as painting or clay modeling), but should be seen as a starting point to future innovative game designs.

Simple Tokens

Simple Tokens are the most common playing pieces in board games: checkers, marbles, chips… Players arrange a limited amount of playing pieces on the board according to game rules (e.g. Checkers, Ludo, Stairs and Ladders, Roulette…) In general, Simple Tokens can be physically described as small flat cylindrical pieces, all identical with the exception of the colour used to distinguish the piece of each player, or to give different values (money, points…). Usually, most board games need a lot of Simple Tokens to be played, but grouped in few categories (Checkers only uses black and white pieces, Ludo uses four colors…).

Named Tokens

Other kind of playing pieces get from game rules a very specific role and unique name, which are perceived by the player through their physical appearance. A classic example is the Chess game: the Tower piece has a different appearance and rules than the Pawn. It is possible to have more than one instance of a kind of Named Token in a game (Chess has four Towers and sixteen Pawns), or to have unique instances of each playing piece (in card games, each card is unique in the game, and has a unique *name* such as Ace of Diamond, Three of Spade …). A Named Token physically identifies the player it belongs to (usually by colour) and the role it has in the game.

Constraint Tokens

A Constraint Token can be described as a playing piece that acts as a physical constraint of a set of smaller Simple Tokens. The bigger piece can be moved in the game board as any other playing piece, but with the manipulation of the Simple Tokens associated with it, the Constraint Token gets new meanings. For example, the playing piece of the Trivial Pursuit game is composed of six triangular small pieces (Simple Tokens) which can be placed inside a bigger circular piece (Constraint Token) (see fig. 4) which represents the progress of each player during the game. In this particular case, handling of the playing piece is based on *associating* (placing/removing) Simple Tokens inside a Constraint Token. Other playing pieces require more complex manipulations, for example, in the Roulette game, players spin a little marble (Simple Token) inside a circular

plate (Constraint Token); in this case the playing piece is based on *manipulating* (moving/rotating) a Simple Token inside the Constraint Token.

Deformable Tokens

Finally, there are other playing pieces which do not have a constant shape, as they change with the manipulations of the player, as they are made of malleable materials, such us clay, cardboard, cloth… In a table, children use these materials in crafts.

Our proposal related to use Deformable Tokens in a game is that they cannot be identified, but characterized by their size, shape…

Next section details ToyVision's distinguishing features and how the four categories of tokens previously presented are related to the toolkit.

TOYVISION TOOLKIT

ToyVision is a toolkit aimed to prototype tangible games in vision based tabletop devices. The architecture proposed is shown in figure 1 right. The Hardware, Calibration and Event Interpretation layers are based on the open-source Reactivision toolkit. However, new functionalities have been added in the Hardware layer in order to support the identification of the four categories of playing objects presented in the previous section. Besides, a Graphic Assistant tool has been developed in order to allow the designer to easily model each tangible control involved in the tabletop game. This Graphic Assistant outputs the configuration files needed by toolkit to identify and model all the playing pieces. The other distinguishing feature of ToyVision toolkit is the Widget layer, created to support high abstraction coding of games in AS3 development environment.

Following, the new functionalities of the Hardware layer, the Graphic Assistant tool, and the new Widget layer are explained in detail.

Hardware Layer

Original Reactivision's Hardware layer identifies fingers and a collection of fiducials placed on the tabletop surface. These functionalities have been adapted and upgraded in order to accomplish some new requirements raised from the proposed classification of playing pieces (see previous section).

Simple Tokens

Simple Tokens can be visually tracked in the toolkit Hardware layer using the shape and size of the blob generated by its base when placed on the table surface. In order to identify each Simple Token with its player, a fiducial is needed to be attached to their base. Original Reactivision's collection of fiducials can support a large amount of different objects (up to 180), but fiducial designs are too complex to be reliably tracked when printed at sizes smaller than 4 cm diameter when using a normal 640x480 px. resolution camera. For that reason, Reactivision's fiducial tracking algorithms have been expanded to track a new collection of Simple Token's fiducials, with a design adequate to be used in small playing pieces (see fig. 2).

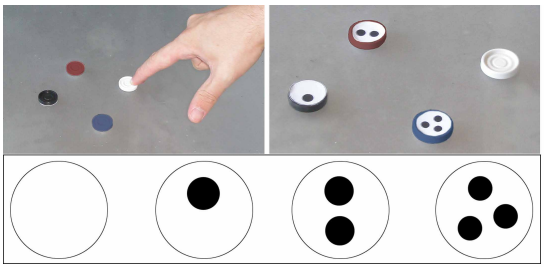

Figure 2. Different Simple Tokens (up-left); with Simple Token's fiducials attached (up-right); detail of the Simple Token's fiducial design (down).

In should be pointed that that fiducials shown in Fig. 2 could be recognized in Reactivision just by extending the mapping tree file but this would led to confusions between two and three topological levels fiducials. The implementation of specific tracking algorithms enables Reactivision to reliably recognize original fiducials together with the extended ones.

Named Tokens

In order to develop tabletop games that use Named Tokens, each playing piece needs a fiducial attached to the base of the object. That way, toolkit's Hardware layer can identify each fiducial by its unique arrange of blobs. ToyVision uses original Reactivision's fiducial collection to identify Named Tokens (see fig. 3).

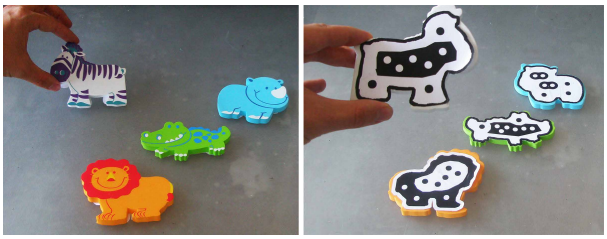

Figure 3. Animal toys used as Named Tokens (left). Reactivision fiducials used to identify them (right).

Constraint Tokens

ToyVision Hardware layer can identify Constraint Tokens by combining the Simple Token and Named Token identification features previously described. In this case, the Constraint Token has a unique Reactivision fiducial attached to its base, and each Simple token is identified with a Simple Token's fiducial (see fig. 4).

Deformable Tokens

As these playing pieces do not have constant shape or size, it is not possible to attach a fiducial to their base. The original Reactivision's Hardware layer tracks these playing pieces as unidentified white blobs. New functionalities have

been implemented in the Hardware layer in order to extract their geometrical attributes: area, perimeter, inertial angles, and contour segmentation (see fig. 5).

Figure 4. Example of Constraint Token: a "Trivial Pursuit" playing piece (left). Piece's base with the fiducial and a Simple Token placed inside the constraint (right).

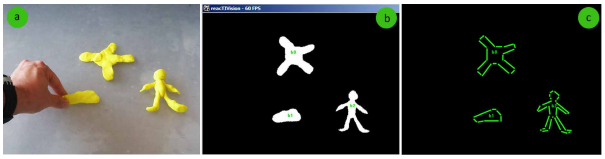

Figure 5. Hardware layer identification of Deformable Tokens. Clay models placed on the tabletop surface (a). White blobs detected by the toolkit Hardware layer (b). Contour segmentation of each blob (c).

Graphic Assistant and XML specification

ToyVision's Graphic Assistant has been designed following a similar approach to that of existing graphic tools included in most popular development environments oriented to code WIMP based applications. These tools enable developers to graphically arrange controls on an application frame and to define attributes for each control. After that process, developers can access to the instantiated classes belonging to each control in the application code. Our Graphic Assistant allows the designer to model, in an easy way, all the data needed by the toolkit (in particular by the Hardware and Widget layers) to detect and track all the different playing pieces involved in the game. The procedure is as follows:

First, the new tangible control must be added to the game by choosing the category it belongs to: Simple Token, Named Token, Constraint Token or Deformable Token (see fig. 6).

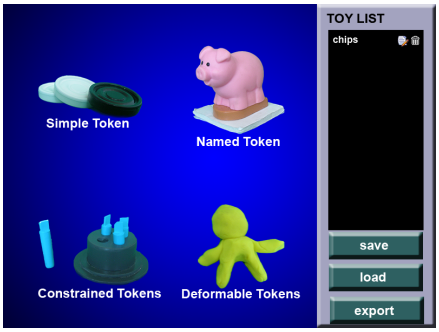

Figure 6. Graphic Assistant: Main menu.

Once the category is chosen, the assistant requires the name of the new tangible control, in order to be identified in the Widget layer.

Next, the assistant enables the designer to graphically model the tangible control. To do that, the designer places the playing piece on the tabletop surface, and an image of the base of the object is captured by the tabletop camera and displayed in the assistant. Then, the assistant asks the designer to graphically introduce all the data needed to model that playing piece. These data are automatically translated into an XML specification. This process varies depending on each token category:

- **Simple Tokens** (see fig. 7): From the image of the base of the pieces, the designer defines the size of the token and a tolerance. This way, any object with a size within the range of tolerance, will be identified as this particular tangible control. Also, the designer sets the number of different kinds of Simple Tokens that will be used in the game and the total number of pieces of each kind.

  These data are automatically translated into XML format to be kept in the configuration files using the following specification:

```
< SimpleTokens name="text" size="number"
tolerance="number" players="number"\>
```

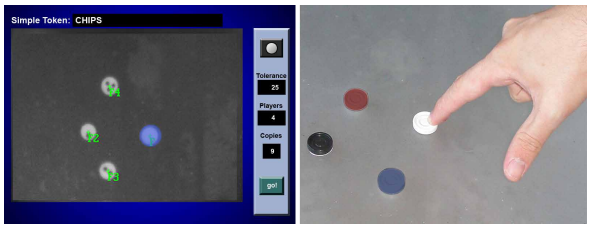

Figure 7. Graphic Assistant: Defining the size (left) of Simple Tokens (right).

- **Named Tokens** (see fig. 8): On the image of the object's base captured by the camera, the designer graphically draws the available area to place a fiducial. The designer also sets the number of copies of that object to be used in the game. These data are translated into XML in the following format:

```
<NamedTokens name="text"
fidID="number_of_fiducial_asigned"\>
```

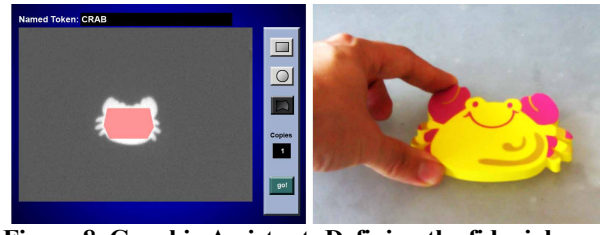

Figure 8. Graphic Assistant: Defining the fiducial area (left) of a Named Token (right).

- **Constraint Tokens**: In this case the first step is similar to the Named Token case; the designer draws the available area for placing a fiducial on the object's base. Then, the designer draws the constrained areas, choosing from two different options: associative or manipulative areas.
    - Associative areas (see fig. 9): The designer draws the areas in which one or more Simple Tokens can be placed or removed.

Figure 9. Graphic Assistant: Defining the Associative Constraint areas (left) of a Constraint Token composed of four Simple Tokens in four Associative areas (right).

    - Manipulative areas (see fig. 10): The designer draws the areas in which one or more Simple Tokens can be moved or rotated.

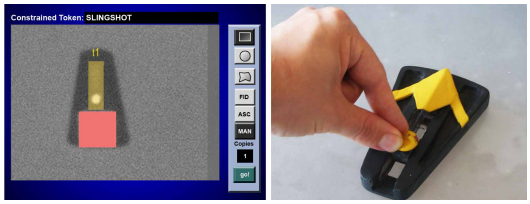

Figure 10. Graphic Assistant: Defining the fiducial area (red square) and the Manipulative Constraint area (yellow rectangle) (left) of a Constraint Token composed of a Simple Token that can be moved along an axis (right).

The position of the constrained areas are converted to polar coordinates relatives to the center and the orientation of the fiducial. Finally, all these data are translated into XML format using the following specification:

```
<ConstraintToken name="text" fidID="num">
    <AssociativeArea name="text" size="num"
    distance="num" angle="num"\>

    <ManipulativeArea name="text"
    areaType="rect" distance="num" angle="num"
    width="num" height="num"\>

    <ManipulativeArea name="text"
    areaType="circ" distance="num" angle="num"
    radio="num"\>
    …
</ContraintToken>
```

- **Deformable Tokens**: In this category, the assistant only asks for an interval of minimum and maximum size of blobs belonging to deformable objects to be tracked, and these data are translated into XML:

```
<DeformableToken minSize="number"
maxSize="number"\>
```

After the designer has created all tangible controls involved in the game, the Graphic Assistant exports the XML specification into configuration files. These files, needed by the Hardware and Widget layer, are loaded by the toolkit at launch. The assistant also creates an Adobe PDF document with all the fiducials required to be attached to each playing piece. This PDF is ready to be printed, and each fiducial can be cut and glued on the base of its respective playing piece (see fig. 11), so that it is ready to be used in the tabletop game.

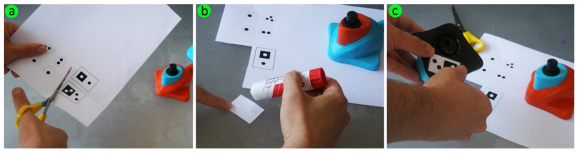

Figure 11. Adapting a playing piece to be used in a tabletop game: Cutting the printed fiducial (a), gluing it (b), attaching the fiducial to the base of the toy (c).

Widget Layer

The new Widget abstraction layer offers all the programing tools needed by the developers to access the status of all the playing pieces placed on the tabletop surface at any moment of the game. ToyVision Widget layer has been integrated in the development environment, Action Script 3 (AS3) in our case. This decision allows the use of the Widget layer with other tabletop toolkits, as far as they use the same communication protocol, TUIO, as the Widget layer receives the raw-events from lower abstraction layers through a TUIO socket.

The developed Widget layer consists of a package of AS3 classes for the Adobe development environments (Flash, Air, Flex). The main class of this package is "ToyList". When this class is instantiated at the beginning of the game code it loads the XML configuration files exported by the graphic assistant (see figure 12). With the data recovered from the XML files, each AS3 class relative to each modeled tangible control is automatically instantiated. While the game is running, the "TabletopEvent" function is automatically triggered each time a tangible control changes its status (placed, moved or removed). In the particular case of Constraint Tokens, an event is also triggered each time any of its associated constraint areas changes. By using the name given in the graphic assistant tool, the developer identifies the tangible control which triggered the event, and writes the code necessary to take appropriated actions in the game.

```
public function Game() {
  //instantiate List of Toys
  gameToys= new ToyList('path to
                    configuration XML file');
  While (true) {//game loop}
}
public function TabletopEvent(toy, eventType)
{
 switch (toy.name) {
    case 'name1':
      if (eventType="add") {//toy placed}
```

```
    if (eventType=”removed”)
                          {//toy removed}
    if (eventType=”updated”)
                        {//toy moved/rotated}
    If (eventType=”constraint”)
        {//toy.updatedConstraint
         // is the area that triggered
         //the toy.constraintEvent
         Switch (toy.updatedConstraint) {
          Case ‘t1’:
           if toy.constraintEvent=”add”
                {//a simple token has been
                 //added into this area
                }
           if toy.constraintEvent=”removed”
                {//a simple token has been
                 //removed from this area
                }
           if toy.constraintEvent=”updated”
                {//a simple token has been
                 //moved in this area
                }
           Break;
          Case ‘t2’;
          …
   break;
 case ‘name2’:...
  ...   } }}
```

Figure 12: Widget layer: Sketched AS3 code for a ToyVision tabletop game.

DEVELOPING A TANGIBLE TABLETOP GAME

In order to show ToyVision usefulness and its advantages in terms of simplicity and versatility, the process of prototyping a tangible tabletop game is outlined in this section. The AS3 code needed when using a conventional Event Interpretation Layer (EIL) toolkit, and when using ToyVision Widget Layer will also be compared.

The game chosen to show this process is a “Pirates” game which has been developed for the NIKVision tabletop [25]. “Pirates” is a cooperative game in which players have to work together to sail a pirate ship and to sink other enemy ships. These actions are carried out by manipulating a set of toys that take advantage of the tangible interaction possibilities of computer augmented tabletops:

- A fan toy is used to control the speed and direction of the ship by spinning the blades of a small fan toy (see fig. 13A), simulating that the toy is virtually blowing the ship’s sails.
- A compass toy is used to point in what direction the player ship should sail to find a ship that can be attacked. The toy is a small cylindrical hash of optic fibers which transmits the image projected from the base of the toy to the top face of the cylinder. A projected needle points to the nearest enemy ship (see fig. 13B).
- A set of black and white chips are used to aim the cannons (see fig. 13C). When a chip is placed on the tabletop surface, ship’s cannons fire a burst of cannonballs to the chip direction. Black chips fire cannonball bursts that spread covering a wide area but travel short distances, and white chips fire precise long distance cannon balls.

These playing pieces belong to different token categories due to their different game functionalities. Next, the way these functionalities are implemented in the game is detailed.

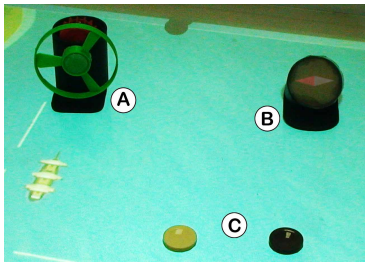

Figure 13. NIKVision Pirates game and playing pieces: A. fan toy, B. compass toy, C. chips

Chips and compass (Simple and Named Tokens)

The first thing to do with these tokens is to attach a printed fiducial on each toy’s base, so that they can be tracked by the toolkit’s Hardware layer.

In an EIL toolkit, the ID associated with the fiducial will be the only way for the developer to handle the corresponding object in the development environment. The TUIO protocol will send an event each time the Hardware layer detects that a fiducial has been placed, moved or removed from the table. The developer has to handle these three types of events and the fiducial IDs to implement the adequate game actions (see fig. 14).

```
Public function AddTuioObject(tuioObject) {
  //triggered when any fiducial is placed
  Switch (tuioObject.ID) {
    Case 0: //fiducial of white chip
      fireLong(tuioObject.x, tuioObject.y);
    case 1: //fiducial of black chip
      fireShort(tuioObject.x, tuioObject.y);
    case 2: //fiducial of compass toy
      drawNeedle(tuioObject.x, tuioObject.x,
        tuioObject.angle, calcNeedleAngle());
  }}
Public function UpdateTuioObject(tuioObject) {
     //Triggered when fiducial is moved/rotated
     If tuioObject.ID==2
      drawNeedle(tuioObject.x, tuioObject.x,
        tuioObject.angle, calcNeedleAngle());
}
Public function RemoveTuioObject(tuioObject){
     //Triggered when fiducial is removed
     If tuioObject.ID==2 eraseNeedle();}
```

Figure 14. Pirates game: schematic AS3 game code for compass toy and chips in EIL toolkits.

In ToyVision, the developer first uses the Graphic Assistant to give a name to each toy and to model it depending on its category. With these data, the Assistant automatically generates a PDF with the fiducials ready to print, cut and glued on the playing pieces’ base. In the ToyVision Widget

layer, a single tabletop event is sent with all the information needed by the developer (name of the toy involved and kind of event) to implement the adequate game actions (see fig. 15).

```
public function TabletopEvent(toy, event) {
   //toy has changed its status
   switch (toy.name) {
    case 'COMPASS':
       if (event=="add" or event=="update")
         drawNeedle(toy.x, toy.y, toy.angle,
            calcNeedleAngle());
       }
       if (event="remove") eraseNeedle();
    case 'CANNON':
       if (event=="add") {
         if (toy.fiducial==0) //white chip
        fireLong(toy.x, toy.y);
      if (toy.fiducial==1) //black chip
           fireShort(toy.x, toy.y);
      }}
```

Figure 15. Pirates game: schematic AS3 game code for compass toy and chips in ToyVision.

Fan toy (Constraint Token)

The action of spinning the blades is detected by the Hardware layer thanks to a half black-white dented wheel in the base of the toy that spins, which is tracked as a white blob appearing and disappearing at the speed the blades are spinning (see fig. 16). In consequence, toolkit's Hardware layer detects two different blobs related to the fan toy: one, the attached fiducial, and the other, the white blob that appears and disappears when the blades spin.

Figure 16. Pirates' Fan toy modeled as Constraint Token (left) with a fiducial and an associative area in its base (right).

In an EIL toolkit, the TUIO protocol sends toy's movement and blades' spin events independently: events related to fiducials are sent as tuioObject events while events related to white circular blobs (blades's spin) are sent as tuioCursor events (see fig. 17). In consequence, when coding the fan toy behavior using the data sent by the EIL, the developer has to implement robust code to find if a tuioCursor event is related to user's manipulations of the fan toy to handle it properly.

On the other hand, using ToyVision, the developer first models the fan toy in the Graphic Assistant as a Constraint Token tangible control. In the AS3 environment, a TabletopEvent is triggered each time the status of the toy changes, either because it has been placed, moved, or removed, or because its constraint areas have changed due to Simple Token manipulations. All the information about the event is available to the developer: name of the toy, type of event, ID of the constraint area that has changed, and the Simple Token status change that may have caused the modification of the constraint area (see fig. 18).

```
Public function AddTuioObject(tuioObject) {…}
Public function UpdateTuioObject(tuioObject) {
     //Triggered when fiducial move or rotate
     If tuioObject.ID==fan_fiducial {
        //update position of fan toy
        fan.x=tuiObject.x; fan.y=tuioObject.y;
        fan.angle=tuioObject.angle;
     }
}
Public function RemoveTuioObject(tuioObject){…}
public function addTuioCursor(tuioCursor) {
   //new circular white blob appeared
   //lots of trigonometric calculations to
   //determine if tuioCursor.y and tuioCursor.y
   //is in the right position and orientation
   //in relation with fan.x, fan.y, fan.angle
   //if true then impulse fan
}
public function UpdateTuioCursor(tuioCursor){}
public function removeTuioCursor(tuioCursor) {
   //circular white blob disappeared
   //lots of trigonometric calculations to
   //determine if tuioCursor.y and tuioCursor.y
   //is in the right position and orientation
   //in relation with fan.x, fan.y, fan.angle
   //if true then impulse fan
}
```

Figure 17. Pirates game: schematic AS3 game code for treating raw tangible TUIO events in an Event Interpretation based toolkit.

```
public function TabletopEvent(toy, event) {
   //toy has changed its status
   switch (toy.name) {
    case 'FAN':
      if (event=="constraint")
        and (toy.updatedConstraint=="t1")
          and (toy.constraintEvent=="add")
        {impulseShip(toy.x, toy.y, toy.angle);}
      If (event=="update")
        {fan.x=toy.x; fan.y=toy.y;
         fan.angle=toy.angle;
      break; }
}
```

Figure 18. Pirates game: schematic ToyVision AS3 game code for the fan toy.

In contrast to other toolkits, ToyVision is also capable of handle tabletop events from un-tagged and deformable objects. To develop a game that uses this kind of materials in ToyVision, developer first uses the Graphic Assistant to create a new playing piece of the Deformable Token kind, giving a name to it. In the AS3 environment, the developer can extract the geometrical data (e.g. a list of segments that compound the perimeter) of any untagged objects manipulated on the tabletop surface (see fig 19).

```
public function TabletopEvent(toy, event) {
   //toy has changed its status
   switch (toy.name) {
    case 'deformable':
      if event=="add" or event=="update" {
        for each (segment in toy.perimeter) {
         //create virtual representation of
         // the un-tagged object}
   }}
```

Figure 19. Schematic ToyVision AS3 game code to treat Deformable Tokens.

The consideration of this kind of playing pieces opens innovative opportunities for tabletop games for using materials and toys in which attaching a fiducial is not suitable. In particular, we have used this new kind of tokens in two simple games developed with ToyVision:

- The Paint game uses conventional brushes to paint on the table (see fig. 20 left). The brush is modeled as a Deformable Token: the width of the stroke will depend on the pressure applied with the brush.
- In the Bugaboo game the players use any kind of deformable material (clay, cardboard…) to build a path so that a virtual flea can jump and climb to reach the fruits (see fig. 20 right).

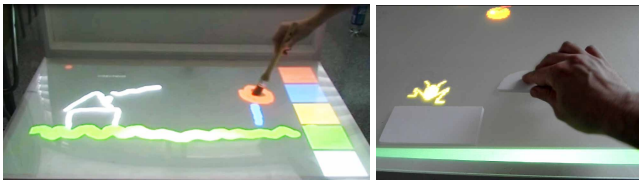

Figure 20. The Paint game (left) and the Bugaboo game (right).

CONCLUSIONS

ToyVision toolkit provides developers and designers of tabletop games with a tool that allows the easy prototyping of games using a great variety of playing pieces, opening new possibilities for tangible interaction in tabletop devices. This has been achieved by adding new functionalities to the Reactivision Toolkit (in the Hardware layer), developing a Graphic Assistant to model each playing piece that automatically generates its XML specification, and adding a new abstraction layer (the Widget layer) that enables designers to face in a high abstraction level the development of a new tangible tabletop game. ToyVision's Widget layer provides access to a set of AS3 classes that give the status of any playing piece handled in the tabletop while the game is running.

Compared to other software toolkits which offer very limited and tag-centered tangible possibilities, ToyVision offers developers with intuitive tools for modeling tangible controls with richer tangible interaction and user manipulations data of higher level.

A beta version of the ToyVision toolkit can be downloaded from http://webdiis.unizar.es/~jmarco/?page_id=297 and can be used and modified under open-source license. The modifications added to the Hardware layer of Reactivision can be easily replicated in other tabletop toolkits based in the TUIO protocol. Likewise, the Widget layer implemented in AS3 may be implemented in other developing environments, and thus, may broaden the number of designers and developers that could take benefit of these new tools. In the near future, ToyVision will be expanded to other environments and to other kind of tangible tabletop games and applications.

ACKNOWLEDGMENTS

This work has been partly financed by the Spanish Government through the DGICYT contract TIN2011-24660.

REFERENCES

1. Al Mahmud, A., Mubin, O., Shahid, S. and Martens, J.B. 2008. Designing and evaluating the tabletop game experience for senior citizens. 5th Nordic conference on Human-computer interaction(NordiCHI '08) pp403-406.
2. Antle, A.N., Bevans, A., Tanenbaum, J., Seaborn, K., and Wang, S. 2010. Futura: design for collaborative learning and game play on a multi-touch digital tabletop. Fifth international conference on Tangible, embedded, and embodied interaction (TEI '11). Pp. 93-100.
3. Bespoke: http://www.bespokesoftware.org/multi-touch
4. Bollhoefer, K. W., Meyer, K., and Witzsche, R.. Microsoft surface und das Natural User Interface (NUI). Technical report, Pixelpark, Feb. 2009.
5. CCV: Community Core Vision Web: http://nuicode.com/
6. Cooper, N., Keatley, A., Dahlquist, M., Mann, S., Slay, H., Zucco, J., Smith, R., and Thomas, B. H. 2004. Augmented Reality Chinese Checkers. Proc. of the 2004 ACM SIGCHI international Conference on Advances in Computer Entertainment Technology (2005). ACE '04, vol. 74. 117-126.
7. Costanza, E., Shelley, S. B., Robinson, J. Introducing audio d-touch: A tangible User Interface for Music Composition and Performance. DAFx '03 Conference.
8. Dey, A.K., Abowd, G.D., Salber, D. 2001. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications, Human-Computer Interaction, v.16 n.2, p.97-166, December 2001
9. Dietz, P. and Leigh, D. DiamondTouch: a multi-user touch technology. In UIST '01: Proc. of the 14th annual ACM symposium on User interface software and technology, pages 219–226. ACM, 2001.
10. Echtler, F., Klinker G. A multitouch software architecture. In Proc of NordiCHI '08. 2008. pp. 463-466.

11. Heijboer M, and van den Hoven, E. 2008. Keeping up appearances: interpretation of tangible artifact design. Proc. of the 5th Nordic conference on Human-computer interaction: building bridges (NordiCHI '08) pp162-171.
12. Hansen, T.E., Hourcade, J.P., Virbel, M., Patali, S. and Serra, T. 2009. PyMT: a post-WIMP multi-touch user interface toolkit. Proc. of the ACM International Conference on Interactive Tabletops and Surfaces (ITS '09). Pp. 17-24.
13. Heng, X., Lao, S., Lee, H., and Smeaton, A. A touch interaction model for tabletops and PDAs. Proc. PPD '08, 2008.
14. Hinske, S. and Langheinrich, M. 2009. W41K: digitally augmenting traditional game environments. Proc. of the 3rd international Conference on Tangible and Embedded interaction (2009). TEI '09, 99- 106.
15. Holmquist L.E., Redström, J., Ljungstrand, P. 1999 Token-Based Access to Digital Information, Proc. of the 1st international symposium on Handheld and Ubiquitous Computing (1999), p.234-245
16. Iwata, T., Yamabe, T., Poloj, M., and Nakajima, T. 2010. Traditional games meet ICT: a case study on go game augmentation. Proc. of the fourth international conference on Tangible, embedded, and embodied interaction (TEI '10). Pp. 237-240.
17. Kaltenbrunner, M., Bovermann, T., Bencina, R., and Costanza, E. TUIO: A protocol for table-top tangible user interfaces. In 6th Int'l Gesture Workshop, 2005.
18. Kaltenbrunner, M. 2009. reacTIVision and TUIO: a tangible tabletop toolkit. Proc. of the ACM International Conference on Interactive Tabletops and Surfaces (ITS '09). Pp. 9-16.
19. Klemmer, S.R., Li, J., Lin, J., Landay, J.A. 2004. Papier-Mache: toolkit support for tangible input. Proc. of the SIGCHI conference on Human factors in computing systems (CHI '04). Pp. 399-406.
20. Leitner, J., Haller, M., Yun, K., Woo, W., Sugimoto, M., Inami, M., Cheok, A. D., and Been-Lirn, H. D. 2010. Physical interfaces for tabletop games. Comput. Entertain. 7, 4, Article 61 (January 2010), 21 pages.
21. Li, Y., Fontijn, W., and Markopoulos, P. 2008. A Tangible Tabletop Game Supporting Therapy of Children with Cerebral Palsy. 2nd International Conference on Fun and Games, Springer-Verlag, pp. 182-193.
22. Libavg web http://www.libavg.de/
23. Lin H.-H., and Chang, T.-W. A camera-based multi-touch interface builder for designers. In Human-Computer Interaction. HCI Applications and Services, 2007.
24. Marco, J, Cerezo, E., Baldassarri, S., Mazzone, E., Read, J. Bringing Tabletop Technologies to Kindergarten Children. 23rd BCS Conference on Human computer Interaction (2009). British Computer Society, Swinton, UK, UK, ISBN:978-1-60558-395-2. pp.103-111.
25. Marco, J., Cerezo, E., Baldassarri, S. Tangible Interaction and Tabletops: New Horizons for Children's Games International Journal of Arts and Technology (IJART). Vol. 5, Nos. 2/3/4. 2012. pp.151-176 ISSN: 1754-8853. Ed. Inderscience.
26. Microsoft surface: http://www.microsoft.com/surface/en/us/default.aspx
27. NUI Group web: http://nuigroup.com
28. Openexhibits web: http://openexhibits.org/
29. Patten, J., Ishii, H., Hines, J.,and Pangaro, G. 2001. Sensetable: a wireless object tracking platform for tangible user interfaces. Proc. of the SIGCHI conference on Human factors in computing systems (CHI '01). Pp. 253-260.
30. Reactivision: http://reactivision.sourceforge.net/
31. Rekimoto, J. and Saito. M. Augmented Surfaces: a spatially continuous work space for hybrid computing environments. Proc. of the ACM Conference on Human Factors in Computing System (CHI'99), pp. 378–385.
32. Rogers, Y. and Rodden, T. 2004. Configuring spaces and surfaces to support collaborative interactions. In O'Hara, K., Perry, M., Churchill, E. and Russell, D. (eds.) Public and Situated Displays. Kluwer Publishers. pp. 45-79.
33. Shaer, O. and Jacob, R.J.K. 2009. A specification paradigm for the design and implementation of tangible user interfaces. ACM Trans. Comput.-Hum. Interact. 16, 4, Article 20 (November 2009), 39 pages.
34. Schöning, J., Hook, J., Motamedi, N., Olivier, P., Echtler, F., Brandl, P., Muller, L., Daiber, F., Hilliges, O., Löchtefeld, M., Roth, T., Schmidt, D. and von Zadow, U. 2009. Building Interactive Multi-touch Surfaces. JGT: Journal of Graphics Tools. Springer.
35. Shen, C., Vernier, F., Forlines, C., and Ringel, M. DiamondSpin: an extensible toolkit for around-the-table interaction. In Proc. CHI '04, pages 167–174, 2004.
36. TouchLib: http://nuigroup.com/touchlib/
37. Trackmate: http://trackmate.sourceforge.net/
38. Touché: http://gkaindl.com/software/touche
39. Ullmer, B., Ishii, H., and Jacob, R. J. 2005. Token+constraint systems for tangible interaction with digital information. ACM Trans. Comput.-Hum. Interact. 12, 1 (Mar. 2005), 81-118.