



UNIVERSITAT DE VIC
UNIVERSITAT CENTRAL
DE CATALUNYA

Master's Degree in Mobile Applications and Games

Master Thesis

Reactive Games. Tangible game system for geometry learning

by

Josep Hornos Arias

Supervisor: Sergi Grau, Department of Digital and Information Technologies

Department of Digital and Information Technologies

University of Vic – Central University of Catalonia

Vic, September 2016

Acknowledgments

First of all, I would like to thank my thesis supervisor, Sergi Grau of the Department of Digital and Information Technologies at UVIC/UCC. Not only for letting me work in such interesting and useful project, but also for give me another point of view when I was stuck in some part of the project. His ideas and help were simply priceless.

I would also like thank the teachers of the Apps & Games Master's Degree of the UVIC for giving us knowledge about different areas. Although I had previous experience related with different IT fields, I learnt a lot of new concepts and discovered tools that, besides being useful to finish this Master Thesis, will be helpful for my professional career.

Last but not least, I wish to thank my couple, Karen and my parents for their inexhaustible patience and their unconditional support along the time that I was doing the Master. They, 'The Doctor', 'The Captain' and some friends who give me joy in my life worth all the efforts made.

Abstract

From the ancient Egypt to our days games evolved continuously. Several innovations in technology made possible to play with people around the world, or simulate realities to give new experiences to players. But this evolution shows us that could be more than pure entertainment in games. Serious games offer a new vision: a way to teach, to train or even perform advertising campaigns. The use of games for teaching purposes is one of the pillars of this thesis.

Bearing in mind that in our country the learning of mathematics is one of the key problems related with school failure, serious games could help professionals to detect deficiencies on the system in order to find a solution.

The main goal of this master thesis is to focus on one important area of mathematics and implement a tangible game system for teaching children geometry. With the use of a tangible table, kids could interact in a physical way with the designed video games, making the experience more attractive and helpful. Additionally, it is possible to evaluate if the children improved their spatial skills by this method, acquiring useful statistics about them while they are playing.

The games implemented on the prototype (one based on puzzles and the other on geometry questions) are fully functional and the system is prepared to be used. During the implementation process, different adjustments and changes respect the original idea have been made, in order to guarantee that the machine is operative and available to perform a proof of concept: let the children play and verify if the premises about learning by this method were correct.

Table Of Contents

Lists of Figures and Tables	8
Abbreviations	11
1. Introduction	12
1.1. Goals of the project	13
1.2. Report structure	13
2. State of the art	15
2.1. Tangible surface based games	15
2.1.1. ReacTIVision Pilot Mixer	15
2.1.2. Reactable Role Gaming (RRG)	16
2.1.3. KIT Vision	17
2.2. Geometry games for children	20
2.2.1. Kangaroo Hop	20
2.2.2. Four Piece Tangram	21
2.2.3. Fun Shape Game for kids	21
2.2.4. Montessori Geometry	23
2.3. Benchmarking summary	24
2.4. Benchmarking conclusions	25
3. Technology	26
3.1. Hardware	26
3.1.1. Table-based tangible user interface	26
3.1.2. Reactable	27
3.1.3. SONA	27
3.2. Software	28
3.2.1. Game development engine: Unity	28
3.2.2. Simulators	29
3.2.3. Backend persistence	30
3.3. External libraries and services	34
3.3.1. Uniducial. Unity3D TUIO framework application	34

3.3.2.	C# MongoDB Driver	35
3.3.3.	Voice RSS: a free online TTS service	35
4.	Reactive Games. Physical system.	37
4.1.	Description	37
4.2.	Environment	37
4.3.	Interface design	38
4.3.1.	Basic projection screen	38
4.3.2.	SONA basic interface	39
4.3.3.	Base configuration	40
4.4.	Software	40
4.4.1.	Fiducial and finger tracking	40
4.4.2.	Incorporing finger tracking to Fiducial Controller	41
4.4.3.	Cursor controller	42
4.4.4.	Local Storage	43
5.	Information architecture	44
5.1.	Player information	44
5.1.1.	Players collection	44
5.1.2.	Player_Stats collection	45
5.2.	System data	45
5.2.1.	Games collection	45
5.2.2.	Fiducials collection	46
5.3.	Game data	46
5.3.1.	GG_Figures collection	46
5.3.2.	GG_Levels collection	47
5.3.3.	Patterns collection	48
5.3.4.	Connections collection	48
6.	Reactive Games. The Core	50
6.1.	Management module	50
6.1.1.	Description	50
6.1.2.	User Management	50
6.1.3.	Geometric Figures Management	51

6.1.4. Tangram Pattern Management	52
6.2. Game Machine	53
6.2.1. Requirements	53
6.2.2. User and game selection	53
6.2.3. Language Selection	54
6.2.4. Reactive Geometry Game	54
6.2.5. Reactive Tangram Game	59
7. Reactive Games. Implementation	65
7.1. The prototype. Final screen design	65
7.2. Limitations and problems detected	68
7.2.1. Calibrating the system	68
7.2.2. Figures and fiducials physical size	68
7.2.3. MongoDB C# driver incompatibility	69
8. Cost of the project	70
9. Conclusions	71
9.1. What has been learned?	71
9.2. Future of Reactive Games	71
9.3. Final reflection	72
A1. Source code	73
A1.1. CursorInputModule	73
A1.2. Fiducial Controller	75
A1.2. Text To Speech & Audio Controller Scripts	82
A2. Figures and Fiducials assignment	85
A2.1. Geometry Game	85
A2.2. Tangram Game	87
References	88

Lists of Figures and Tables

Fig 2.1 - ReacTIVision pilot mixer. Demonstration	15
Fig 2.2 - Gameplay: combining pilot parts. Fiducials and pieces used in the game	16
Fig 2.3 - RRG Editor. Character creation. Game objects configuration	16
Fig 2.4 - RRG Editor, Level & figure selection. RRG Game Player in action	17
Fig 2.5 - Game player. Selection. A kid playing with tangible Pac-Man version	18
Fig 2.6 - Editor: defining game properties. Game Player: testing & configuring a new game	18
Fig 2.7 - Scheme for KIT vision game	19
Fig. 2.8 - Kangaroo Hop gameplay	20
Fig. 2.9 - Kangaroo Hop results	20
Fig. 2.10 - Four Piece Tangram gameplay	21
Fig. 2.11 - Fun shape game. 2D gameplay. 3D gameplay	22
Fig. 2.12 - Fun shape game. 3D gameplay with different quizzes	22
Fig. 2.13 - Montessori Geometry. Puzzle activities, identification gameplay. Rewards & scores	23
Table 2.1 - Benchmarking summary	24
Fig. 3.1 Fiducials sample	26
Fig 3.2. Reactable (Music Technology Group - UPF)	27
Fig 3.3. SONA project (Multimedia Degree UVIC-UCC, 2016)	27
Fig 3.4. Unity Logo	28
Fig 3.5. Unity Editor. Mono Develop Environment	28
Fig 3.6 - TUIO Simulator in action	29
Fig. 3.7 - TuioPad (iPod version)	30
Fig 3.8. Appcelerator & Arrow logo	31
Fig 3.9. Firebase logo	31
Fig 3.10. GameSparks logo	32
Fig 3.11. mLab logo	32

Table 3.1 - Backend comparison	33
Fig 3.12 - Voice RSS interface. Statistics & analytics section	36
Fig 4.1 - Reactive Games scheme	38
Fig 4.2 - Projection Screen. Feedback sample for Tangram game and User selection screen	39
Fig 4.3 - SONA interface samples. Geometry game. Tangram game	39
Fig 4.4 - Fiducial Controller script options on editor. Fiducial and finger assignment	41
Fig. 4.5 - Modified code of Fiducial Controller to get full cursor detection	42
Fig. 4.6 - Schema of the Cursor Input Controller behaviour	43
Table 5.1 - Definition of Players collection	44
Table 5.2 - Definition of Players_Stats collection	45
Table 5.3 - Definition of Games collection	45
Table 5.4 - Definition of Fiducials collection	46
Table 5.5 - Definition of GG_Figures collection	47
Table 5.6 - Definition of GG_Levels collection	47
Table 5.7 - Definition of Patterns collection	48
Table 5.8 - Definition of Connections collection	49
Fig. 6.1 - Accessing to Reactive Games Management menu	50
Fig. 6.2 - New player creation screen	51
Fig. 6.3 - Screen design for new geometric figure insertion	51
Fig 6.4 - Tangram pattern creation & insertion interface	52
Fig. 6.5 - User Selection Screen	53
Fig. 6.6 - Game Selection Screen	54
Fig. 6.7 - Geometry Game flow diagram	56
Fig 6.8 - Geometry Game activity selection screen	57
Fig 6.9 - Screen projection for Geometry Game. Different messages for various situations along a game playing	58
Fig. 6.10 - SONA interface. Screen design showing the results of interaction over the board for some questions	59
Fig. 6.11 - Geometry Game full screen design	59

Fig. 6.12 - Tangram base square divided in the 7 pieces and different composed figures	60
Fig. 6.13 - Tangram pieces with numbered vertexes	61
Table 6.1 - List of connections vertex-vertex and vertex-side for the given figure	62
Fig. 6.14 - Projection screen design samples for Tangram Game	63
Fig. 6.15 - Construction of a tangram figure	63
Fig. 6.16 - Tangram Game full screen design	64
Fig. 7.1 - Different views of SONA interface for Geometry Game	65
Fig. 7.2 - Full system. Playing Geometry Game	65
Fig. 7.3 - Projection screen (left) and SONA interface (right) for Tangram Game	66
Fig. 7.4 - Full system. Playing Tangram Game	66
Fig. 7.5 - User & language selection (left) and Game selection (right)	67
Fig. 7.6 - Management option selection (left). Geometry Game Activity selection (right)	67
Fig. 7.7 - Different pieces used in Reactive Games	67
Fig. 7.8 - Calibrating the system with reacTIVision application	68
Table 8.1 - Cost of the prototype	70
Table A2.1 - Figures and fiducial association for Geometry Game	85
Table A2.2 - Figures and fiducial association for Tangram Game	87

Abbreviations

In order to help the reader to understand the acronyms used throughout this thesis, a table with the abbreviations and their meanings is shown. Additionally, the page of the first appearance is given.

Abbreviation	Meaning	Page
ACL	Acces Control List	31
API	Application Programming Interface	26
BaaS	Backend as a Service	31
CRUD	Create, Retrieve, Update, Delete (document NoSQL operations)	35
DaaS	Database as a Service	32
OSC	Open Sound Controller	26
MBaaS	Mobile Backend as a Service	31
NoSQL	Not Only SQL - Not Only Structured Query Language	30
REST	REpresentational State Transfer	32
RG	Reactive Games	13
RPG	(Role Player Game)	17
RTDB	Real Time DataBase	33
TTS	Text To Speech	34
TUI	Tangible User Interface	26
UI	User Interface	40

1. Introduction

Games are part of our culture since the very origin of the human race. As some form of entertainment and/or competition, solving different challenges that required thinking or/and physical abilities has been a common practice. From the ancient games (some of them date from above 4000 years), played with simple boards or hand made pieces (the Egyptian *Senet* or the Indian version of *The Royal Game of Ur*) to the first video games (such as *Pong* or *Space Invaders*) a lot of innovations has been made. But in the last years, with the quick evolution on technology (better 2D-3D game engines and Virtual/Augmented Reality) and the use of Internet, conception of games has changed forever. Now we have the possibility to play games with people on other parts of the world giving the players new experiences using virtual environments integrated in the real world (good examples could be *Candy Crush Saga* or *Pokemon Go!*).

But games are not necessarily a form of entertainment: Serious Games made use of them in different areas, such as education, health or communication, in order to improve people knowledge or simply to show something. Specially interesting is their use on education, providing new paradigms on teaching that can help professionals to reduce school failure, and making the learning process easier and more attractive.

Considering that great part of the school failure is due to mathematics in our country, trying to use serious games in this area of knowledge could be justified. As long as this study has several different branches, we want to focus on one of them: geometry. Concepts such as shape, position, size, and properties of figures in the space are the base, so the student must work spatial abilities, language and memory to improve their skills in this area.

Newcombe, N. S., & Frick, A. (2010) [1] found that spatial learning could be improved through activities, such as solving puzzles or jigsaws, in the early years (preschool). These tasks helps the kids to develop pattern recognition on the real world, and also improves their psychomotor coordination, having to perform actions like movement and rotation of shapes. Additionally, teaching the names and properties of figures help kids to develop association aptitudes, apart from improving their linguistic resources applied to mathematics.

Use of serious (educational) games could help children develop spatial abilities and learn geometry. There are a lot of video games for this purpose that teach these concepts in a 2D fictitious world. But, if we can give kids the opportunity of play these video games using real shapes, pieces that they can touch and feel, the learning process could be boosted. We can reach that objective using tangible user interfaces, such as *Reactable*. And this is the main goal of this project: develop games in a tangible table environment for teaching children geometry.

1.1. Goals of the project

The main goal of this project is develop games in a tangible table environment for teaching children geometry: Reactive Games (RG). Implementing a functional prototype for this purpose implies to divide the work in different milestones.

Firstly, we need to adapt the existing tangible table created in the UVIC (SONA) to host a system with different games. That implies not only researching about tangible tables and computer vision frameworks (in our case, reacTIVision) to understand how this environment works, but also making all work properly with Unity game engine.

Once the table is adapted, calibrated and working, time to implement games comes. One good solution is to create two representative games. The first one, related with shape recognition, classification and geometry language. The second, a version of an ancient puzzle, Tangram, useful to work on pattern creation and psychomotor abilities.

An important aspect to evaluate is how a system like this can bring benefits on kids learning. This is why collect data about players evolution in each game is mandatory. To solve that, two solutions are implemented: an online backend and a local storage for offline working.

Finally, some management tools are developed to help teachers and professionals to improve the games adding new options, such as creating new tangram patterns or geometric figures.

1.2. Report structure

In order to develop each topic pointed in introduction and goals section, this thesis is structured following a logic order that must facilitate how the work has been done.

In Section 2 two studies are performed: the first one is related with existing games based on tangible tables, and the second tries to give an insight of geometry video games. Benchmarking gives us an idea about the state of the art in both areas, being useful to design our games.

Technology used in this project is described in Section 3. Hardware subsection gives some insights about tangible table technology, showing the solution that will be part of the prototype. In software subsection information related with the game engine and tools for developing used can be found, apart from a brief study of the backend to use with our system. Finally, a third subsection introduces the external libraries and services used in our development.

Section 4 deals with the construction of the physical system. To understand how it is designed and implemented, a brief description about the environment is given. After that, we can find the design

of the basic interface, and finally, how different tools are developed in order to give full functionality to the prototype, such as the figure and finger tracking control or the local storage.

How is defined the information structure is explained in detail in Section 5, going from the most simple information (related with player and statistics) to system and games data and logic.

Section 6 talks about the core of the project: the design of the management module and the game machine. For the first one, a general description is the starting point for a deep explanation about the management options implemented. For the second one, after defining the requirements about the games to be designed, logic of the game machine and the games development process is fully described.

Finally, in Section 7 we can see how the final prototype looks like once is implemented and tested in a real environment, explaining the actions taken to reach that goal, and which problems and limitations are detected during the whole process.

A small calculation about the cost of developing a prototype like the described can be found in section 8, and Section 9 offers the conclusions for closing this report.

2. State of the art

Before start designing and developing, it is important researching about the kind of games that we pretend to implement. In order to perform this benchmarking two different areas were studied:

- Games based on a tangible interactive surface.
- Games for children, specially in the geometry area.

With this study we want to know if there is something similar to what we are trying to do, and learn something about tangible games and common features of games for children.

2.1. Tangible surface based games

Reactable is the most recognised and important development based on tangible interactive tables, but there are only few games developed for that systems, though the potential is great. After performing a research on this area (tangible tables based on reacTIVision), three examples were relevant. Even though two of them are prototypes, and some are also frameworks, it could very interesting to know something about them.

2.1.1. ReacTIVision Pilot Mixer



Fig 2.1 - Demonstration of how to interact with the game

This is a simple prototype designed and implemented by students and teachers of the *Interaction Design Programme* on the Copenhagen Institute of Interaction Design (CIID). This system use the reacTIVision framework as base to construct a multitouch tangible table and is developed using Adobe Flash.

The game is very simple: with a set of physical pieces attached to different fiducials that represent three parts of a human body (head, torso and legs), players must create a human figure. There are different characters based on the students of the pilot year (2008), so the combinations give to the players very different and funny results.

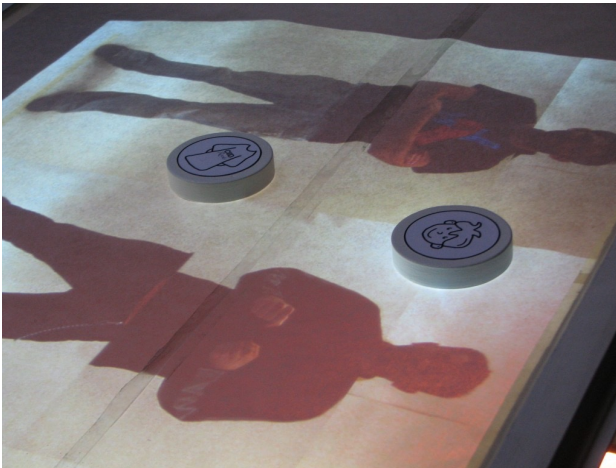


Fig 2.2 - Gameplay: combining pilot parts (left). Fiducials and pieces used in the game (right)

A simple gameplay and interface are remarkable aspects on this prototype. It is a good example of what can be done with this kind of systems. It can be considered a noticeable first approximation to tangible table based games.

2.1.2. Reactable Role Gaming (RRG)

This is a very interesting prototype, consisting on a complete development kit for creating role games for a tangible table based on reacTIVision. This was a Final Degree Project developed by Ramón Viladomat Arrò (Universitat Pompeu Fabra) [2] in 2008. Having the opportunity to work with the creators of the Reactable, this framework is quite interesting and complete.



Fig 2.3 - RRG Editor. Character creation (left). Game objects configuration (right)

The idea behind the project is simple. In an RPG (Role Player Game) exists one main character: the Game Master, who is in charge of create the storytelling, the environment, characters and behaviours (magic, powers, strengths, etc). RRG lets the Master to create this fantasy world in some sort of augmented reality game where players can interact with a tangible surface, improving the experience of playing an RPG.



Fig 2.4 - RRG Editor, Level & figure selection (left). RRG Game Player in action (right)

The prototype is designed using reactTIVision as vision computing engine, with a tangible table programmed properly for that goal (a modified Reactable). Influenced by the popular table game *HeroQuest*, this framework offers all the options needed to create the story for an RRG: character creation, map configuration, creation of objects, items, levels and teams. Also, a module called Player let the people play in order to test the created game.

As a project, the final product is amazing: accurate design, good implementation and a great idea. This was a perfect example of what can be done with a tangible surface.

2.1.3. KIT Vision

This is more than a tangible table based game. It is a framework to create basic tangible games without knowing nothing about programming. Based on reactTIVision, the main idea behind KIT vision is giving a powerful tool to create and use games applied to different areas (teaching, psychology, etc) improving their activity and results in an easy way.

Developed by Clara Bonillo, Olga Blasco, y Javier Marco from the Computing & Systems Engineering Department on the Zaragoza University, KIT vision is available for different platforms, but requires Adobe Air in order to be executed.

KIT vision is composed by two main tools:

- Game editor: lets the user set up all the virtual environment (background, images, sounds), the physical elements (defined by the attached fiducials) and the tasks that compose a game.
- Player: is in charge of execute all the logic of the system, putting all the game information and creating a fully playable version of the game. Also, is the point of entry of all the games that the used has developed for this system.

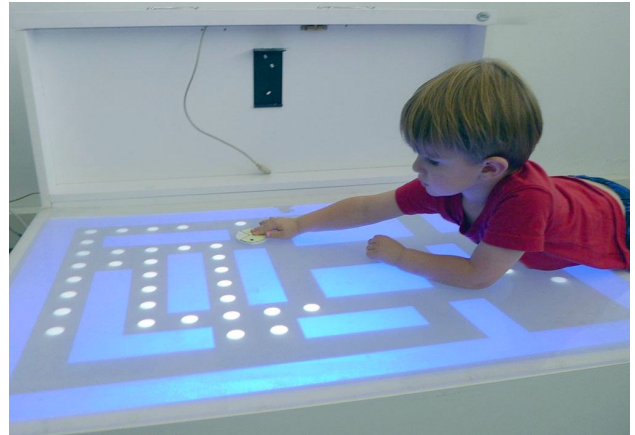


Fig 2.5 - Game player. Selection (left). A kid playing with tangible Pac-Man version (right)

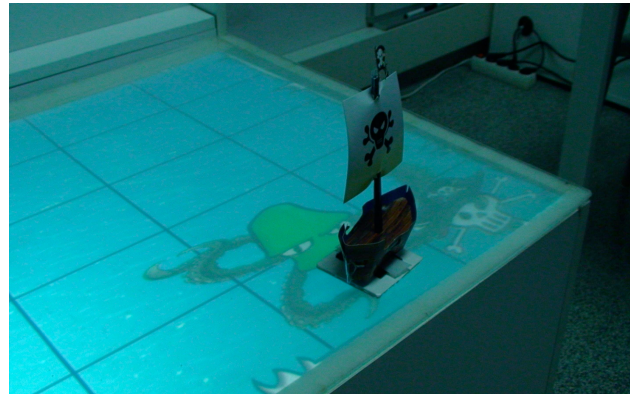
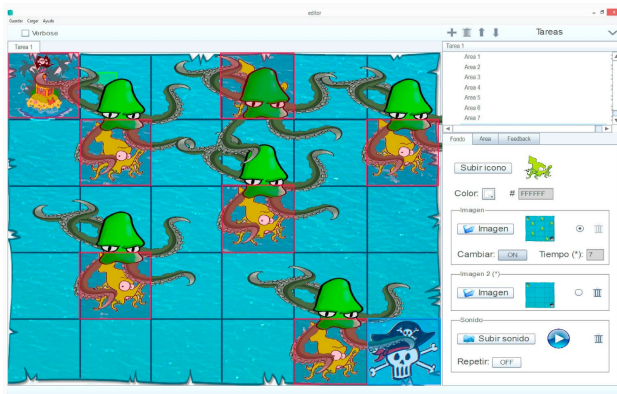


Fig 2.6 - Editor: defining game properties (left). Game Player: testing and configuring a new game (right)

In order to develop this framework, simplicity was the main goal, so all the games must follow a predefined scheme that sets the main parts of a game:

- Every game is composed of n sub-games or tasks
- A task must be finished to reach the next
- Every task is composed by a background image, different areas (where evaluate the interaction), a list of correct objects per area and another one for incorrect ones, and finally, some feedback.

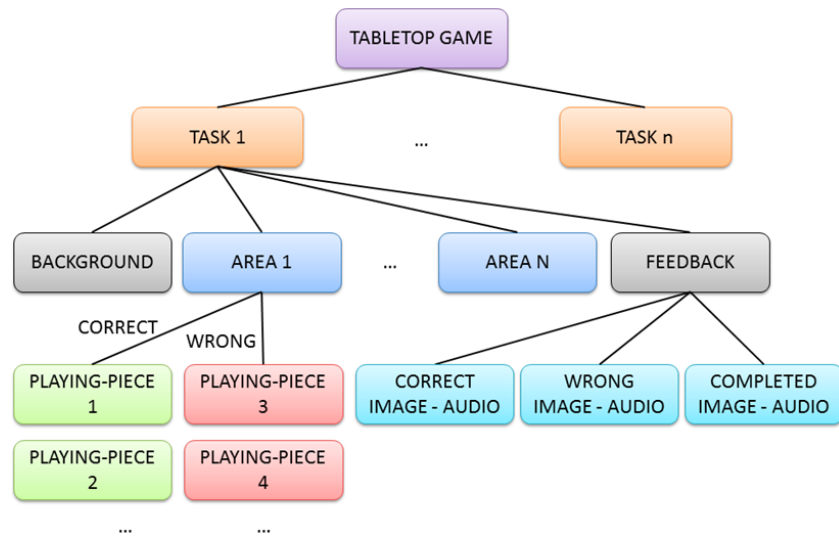


Fig 2.7 - Scheme for a KIT vision game

By default, different games are implemented and downloadable from the project webpage [17] (the most popular is a *PacMan* tangible version).

The idea of offering to a professional with no programming skills the possibility of creating his/her own serious games is simply fantastic. Although the extreme simplicity of the games that can be developed could limit the possibilities, this framework/game machine is a very useful product.

2.2. Geometry games for children

2.2.1. Kangaroo Hop

The main goal in this web game is helping the kangaroo to complete a race, having to hop on the correct geometric figure (avoiding to fall on the water). The figure to reach is indicated on the screen, and the player must choose between 4 options. It is a simple way to test indirectly how many figures the player knows.

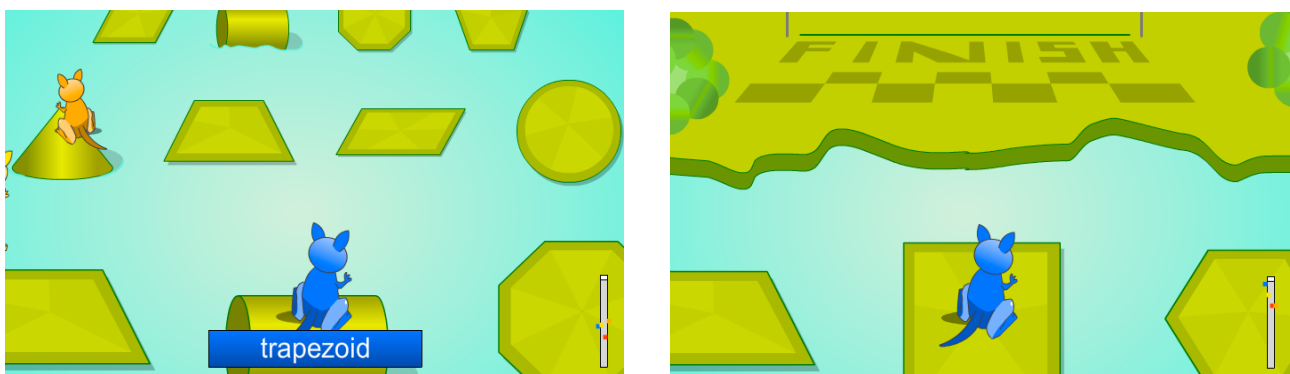


Fig. 2.8 - Kangaroo Hop gameplay

Simplicity is a good point on this game, in addition to an attractive design. The gameplay is quite easy and clear, and the player has the opportunity to test his/her knowledge against other opponents.



Fig. 2.9 - Kangaroo Hop results

One negative aspect is that the game could be a bit repetitive: it is always a race than begins and ends, with the only difference of the order on the requested figures. Another negative point is that although for each race there is a ranking, global results and scores doesn't exist. That gives an impression of discontinuity and no final goal.

2.2.2. Four Piece Tangram

This is a simplification (and variation) of the ancient Chinese game Tangram. The mechanics are quite simple: with the four geometric pieces provided (originally were 7) the player have to complete the figure shown in the pattern. To do it, the pieces can be rotated and flipped, but all of them must be used. This is an ideal game to test and develop not only spatial and geometric thinking, but also to test player's problem solving capacity.

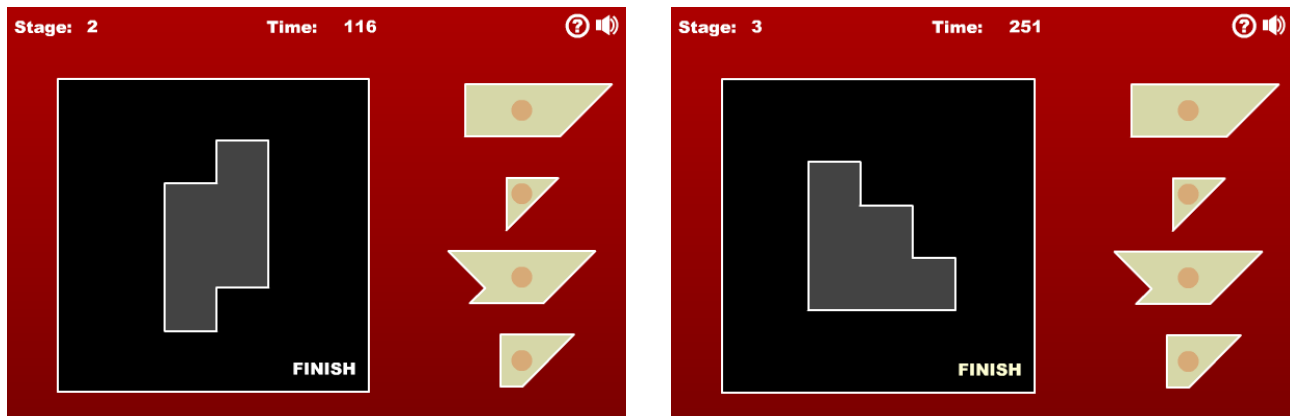


Fig. 2.10 - Four Piece Tangram gameplay

As positive aspect on this web game, we must remark the simplicity of the mechanics, along with the fact that resolving puzzles could be very addictive, although commonly are one player games. A non-attractive interface is the most negative point. Another one is that the game has no continuity: the player passes from one stage to the next, not having the opportunity of save its progress. The last one negative aspect is the playability: the controls not always react in the correct way, giving the sensation of losing control on manipulating the pieces.

2.2.3. Fun Shape Game for kids

This web game consist in classify 2D-3D figures in different blocks, depending on different properties (number of sides, equality of sides, etc). The mechanics are simple: select the figure that appears in the centre of the screen and put it on the correct box. If it is not correctly located, the figure returns to the conveyor belt where it was picked up. Additionally, different questions could appear related with the selected figures.

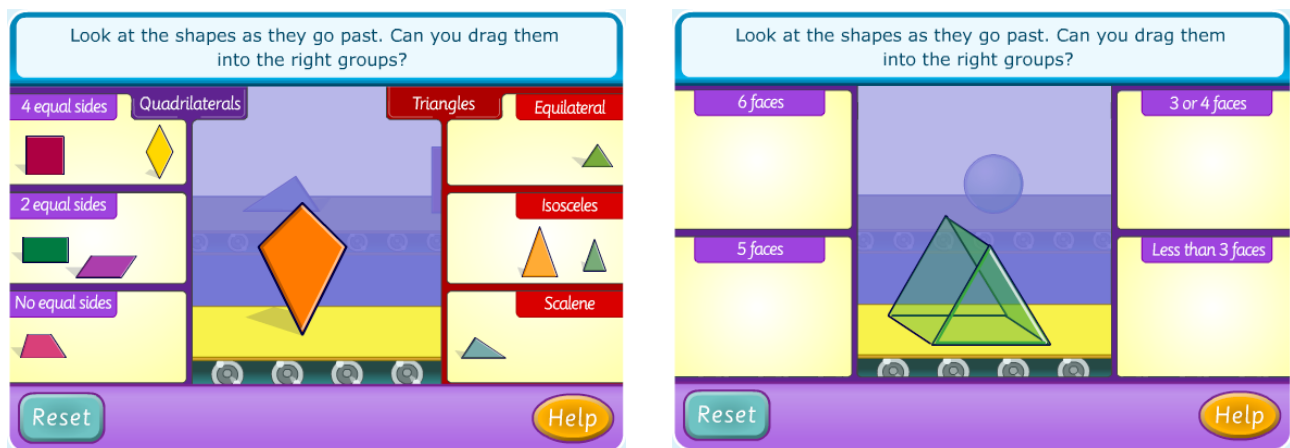


Fig. 2.11 - Fun shape game. 2D gameplay (left), 3D gameplay (right)

That game has very positive points: the interface is clear, attractive and simple. It has good playability, and the goal is quite clear. It has two different parts: first the 2D game and after, the 3D (which has different activities inside). Also, it gives the impression of personal reward on passing to the next level.



Fig. 2.12 - Fun shape game. 3D gameplay with different quizzes

As a negative point, this game has not so many different levels, resulting interesting to play once but no more times. Despite this, it can help to teach, in an easy way, the basics about geometric shapes.

2.2.4. Montessori Geometry

Available for iOS devices, Montessori Geometry is a set of different games and activities related with geometry. Designed from professionals on mathematics teaching, and thought to be played by children from 5 to 10 years old, it is an educational game with the main goal to teach geometry in an easy and fun way.

There are different kind of activities: shape identification (classify figures or detect them into different daily places), puzzles (put figures in the right places), differentiation (set patterns, discard odd figures), categorisation (order figures by properties or by logic), quizzes and a 3D discovering module .

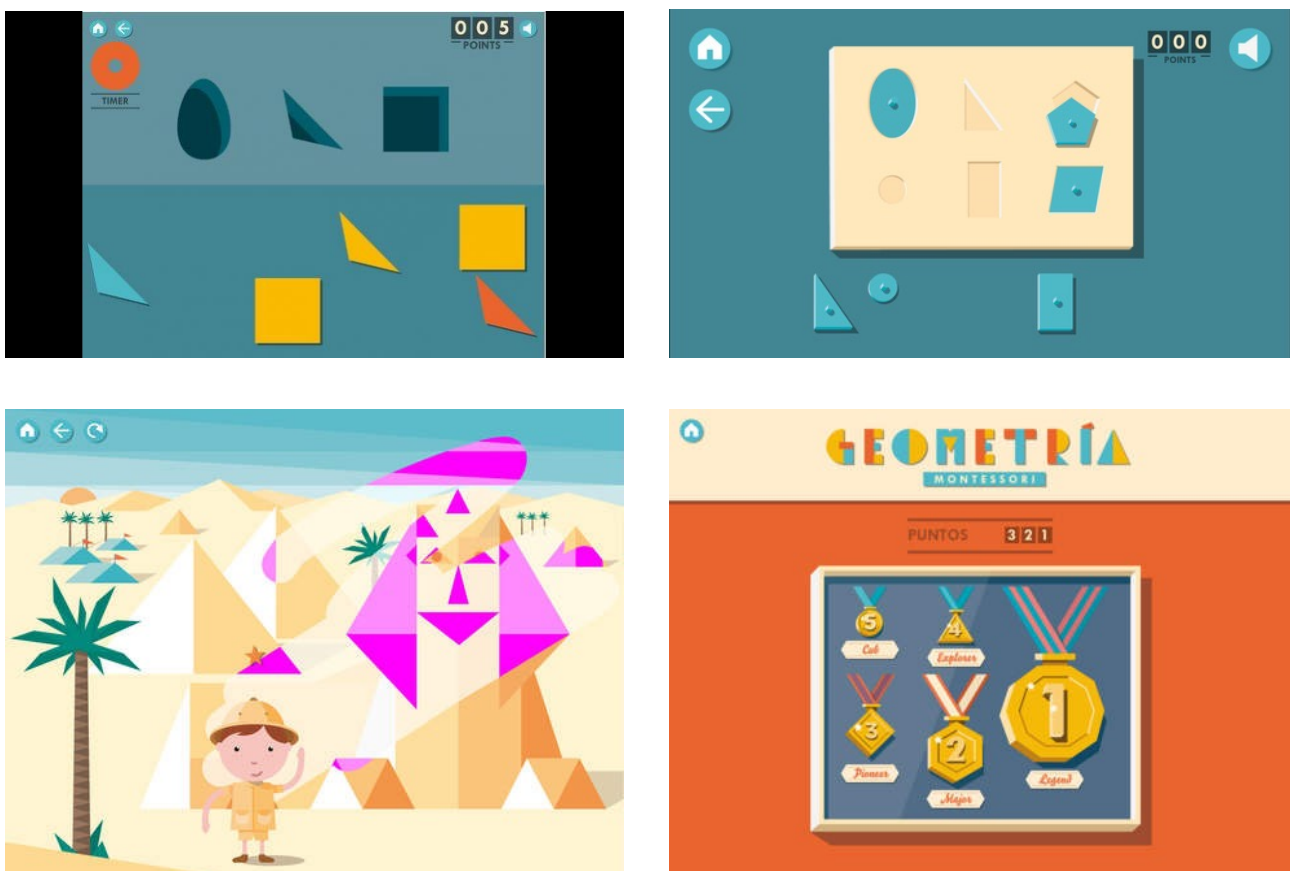


Fig. 2.13 - Montessori Geometry. Puzzle activities (top), identification gameplay (left - bottom).
Rewards and scores (right - bottom)

Montessori Geometry has a well designed interface, mixing simplicity with good taste in each and every detail (colours, shapes, etc). The environment is perfect to attract the attention of a kid. It is highly playable, and the fact that there is a reward system is a very good aspect. In one hand, children have fun, and on the other parents can see how their kids are really learning.

It is difficult to find negative points, but maybe the only one is that it is not a free app. But taking into account how the game is, it worths paying for this high quality product.

2.3. Benchmarking summary

The basic data collected before analysing the games is shown in the Table 2.1.

Table 2.1: Benchmarking summary

	Type	Description	Link
ReacTIVision Pilot Mixer	Tangible table based game	Composition of human figures making combinations of different parts of bodies.	http://ciid.dk/education/portfolio/py/courses/toyview/projects/reactivision-pilot-mixer/
Reactable Role Gaming	Tangible table based game	Framework to create and play Role Games with a tangible and interactive surface.	http://repositori.upf.edu/handle/10230/4739
KIT Vision	Tangible table based game	Framework to create and play simple games for kids, using marked physical figures over a tangible surface.	http://webdiis.unizar.es/~jmarco/?page_id=925
Kangaroo Hop	Web game	Geometry recognition game for children.	http://www.mathplayground.com/ASB_Kangaroo_Hop.html
Four Piece Tangram	Web game	Variant of the original Tangram puzzle, using only 4 pieces.	http://www.mathplayground.com/tangrams.html
Fun Shape Game for kids	Web game	Recognition and classification of 2D-3D geometric figures by their characteristics. Includes quizzes.	http://www.kidsmathgamesonline.com/geometry/shapes.html
Montessori Geometry	Mobile game (iOS)	Complete set of geometry games for kids (recognition, classification, etc). Each game is focused on an area, and includes an interesting reward system.	https://www.edokiacademy.com/en/app-montessori/math/montessori-geometry/

2.4. Benchmarking conclusions

In the performed analysis we saw two different kind of games: tangible table based video games and geometry video games. Unfortunately, no combination of both could be found, so the main goal of Reactive Games could be considered, in a certain way, innovative.

It is quite clear that tangible table based video games is a field with a lot of possibilities, but not so much explored yet. The possibility to create not only a game but a framework to develop them is quite interesting, so having a certain degree of customisation is important for users and players. Simplicity is another thing to take into account, especially when the final product is destined to be for children. TOY vision is a good sample.

Four geometry video games for kids were analysed. There are a great quantity of this kind of games, but the studied fit perfectly with what we wanted to do. Thanks to the benchmarking, useful knowledge about their design and properties has been acquired:

- As we wanted to implement games for teaching, and the objective public will be children, it is important to guarantee that players enjoy them. An excessive difficult gameplay could dismiss people to continue with the game.
- Rewards are important in order to motivate the players to play again.
- Games for kids must be simple and attractive. An impressive interface, with a simple environment and accurate palette of colours can make the game engaging (to the eyes of a child).
- A good story is important: although players are really learning with the game, they must have the impression not to being solving a mathematics problem but, simply, playing. A good sample of that is shown on Montessori Geometry: showing popular locations where the buildings and other objects are geometric figures that must be detected by the child (Fig 2.13), gives some attractive background far from the fact that they are improving shape recognition.
- As we are dealing with players from 4 to 12 years, the interaction must be simple and the objectives completely clear. Giving exact information about what to do we can help the children to achieve the goals and avoid them falling into frustration.

Once the analysis is finished, we have valuable information that can guide us in the effort to design and implement something new in a proper way, mixing the good things of existing geometry games with the possibilities offered by tangible tables.

3. Technology

To implement Reactive Games it is necessary to work with different things. On one side, an interactive tangible table is needed, and on the other, a game engine and software compatible with the system to provide a consistent product. In this section we show the selected options, followed by a brief explanation about why they were chosen.

3.1. Hardware

3.1.1. Table-based tangible user interface

A Tangible User Interface (TUI) is a computer based system that let the user interact whit it using physical objects or multi-touch events. The open framework TUIO defines an API and specifies a protocol for tangible user interfaces. TUIO protocol is based on OSC (Open Sound Controller), a standard for interactive environments.

The base of the system is reactTIVision, an open source computer vision framework built on top of the TUIO protocol (designed to provide a standard communication specification for TUI tables). Thanks to this application, it is possible to perform a robust and accurate fiducial marker and multi-touch finger tracking.



Fig. 3.1 Fiducials sample

In order to process the user interaction, we need two actors: reactTIVision and a TUIO client. The first one detects all the objects placed on the surface, thanks to the markers attached to them (known as fiducials). Additionally, a finger detection and tracking is performed. Using the TUIO protocol, reactTIVision sends in real time the information about the interaction to the client, who is in charge to perform the needed operations. As this operations are related with some interaction with the user, a visual response to its actions must be shown.

In terms of hardware, a basic implementation of the system needs:

- A computer with an instance of reactTIVision running and a TUIO client application.
- A projector, used to show on the surface the visual results of user interaction.
- A camera, to capture the fingertips and the markers placed on the objects.
- A diffuse infrared lamp.
- A top table translucent surface.

The translucent surface let the user see the projection, making fiducials on the top of the table transparent for the camera, so the fingers and fiducials can be tracked correctly. In order to avoid interferences between the projection and the tracking, its necessary to perform this operation in a different light spectra. Using an infrared illumination and a camera with an infrared filter is possible to guarantee an optimal performance.

3.1.2. Reactable



Fig 3.2. Reactable (Music Technology Group - UPF.
<http://mtg.upf.edu/project/reactable>)

Reactable is a tangible modular synthesiser built using the reactTIVision computer vision framework. It is a particular implementation of the tangible table described some lines above, with the particularity of being fully dedicated to create music in an interactive way.

Designed and implemented by the Music Technology Group at the Pompeu Fabra University (Barcelona) since 2003, Reactable has become a standard on tangible table based systems, being adopted by several musicians as a tool to create and innovate in their music styles.

3.1.3. SONA

Based on the initial implementation of a TUI, a group of students and teachers at the UVIC-UCC designed and developed its own tangible table for musical purposes: SONA.



Fig 3.3. SONA project (Multimedia Degree UVIC-UCC, 2016. <http://multimedia.uvic.cat/project/sona-2/>)

This prototype (created for the Multimedia Degree) is built using a Linux operating system (Mint), and is capable not only to play the main role of a Reactable, but also to use an external (secondary) monitor to play videos, show images, etc. Taking advantage of the hardware it is possible to transform SONA in a platform for tangible and interactive games.

3.2. Software

Implementing a game requires the usage of a development environment, a game engine and a database where to store not only the logic of the game, but also player information useful to evaluate the impact of the realized development.

3.2.1. Game development engine: Unity

A good game development engine is one of the most important parts in the process of the creation of a video game. Different options are available, but two that are the most used and reputed: Unreal Engine and Unity. For this project, and due to the knowledge acquired along the Master, the game engine selected is Unity.



Fig 3.4. Unity Logo

Unity is a cross-platform game engine widely used to develop video games for computers, mobile devices, consoles and websites. Developed by Unity Technologies and written in C and C++ offers a full development environment that makes easier the design an implementation of different kind of video games (in both 2D and 3D). As related graphic engines can works with Direct3D (Windows), OpenGL (Linux and MacOS) and OpenGL ES (for mobile operating systems).

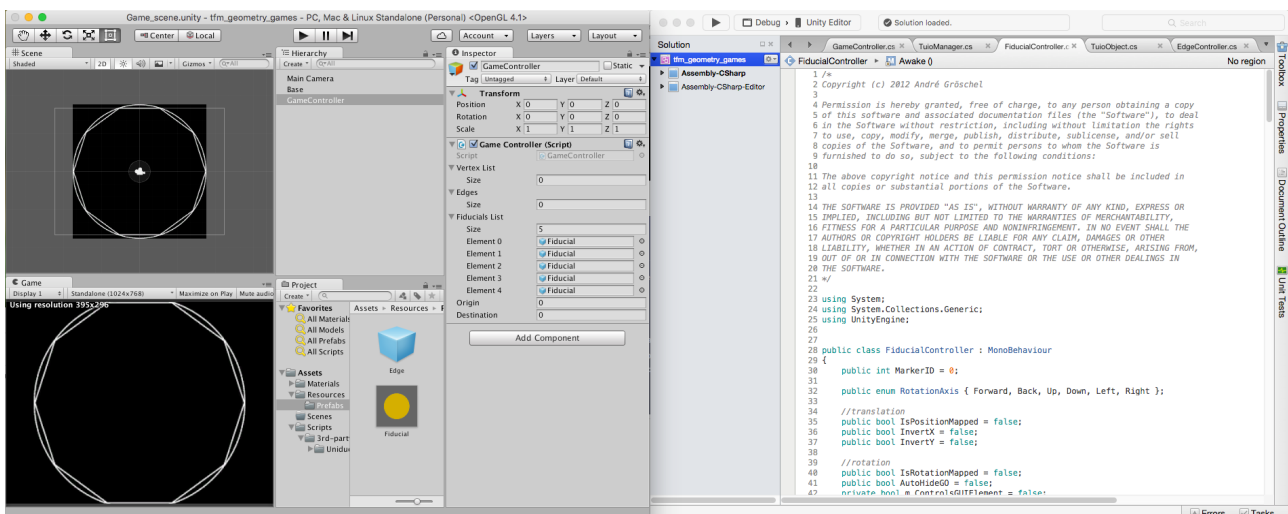


Fig 3.5. Unity Editor (left). Mono Develop Environment (right)

Unity uses an implementation of the standard Mono runtime (derived of .NET) giving different options for programming the scripts, although the most popular are C# (industry-standard language similar to C++) and UnityScript (modelled by JavaScript, specific for Unity). The programming environment is MonoDevelop.

For this project is used C# as scripting language. There are two main reasons to use it. First, the fact that is the most documented and supported option, and second, a personal familiarity with similar languages that make easier the process of learning and thus, programming.

3.2.2. Simulators

Great part of the job must be done without interacting directly with the SONA table. In order to program and test different parts of the game where interaction with fiducials and finger tracking is mandatory, two useful tools were used: *TUIO Simulator* and *TuioPad*.

3.2.2.1. TUIO Simulator

http://prdownloads.sourceforge.net/reactivision/TUIO_Simulator-1.4.zip?download

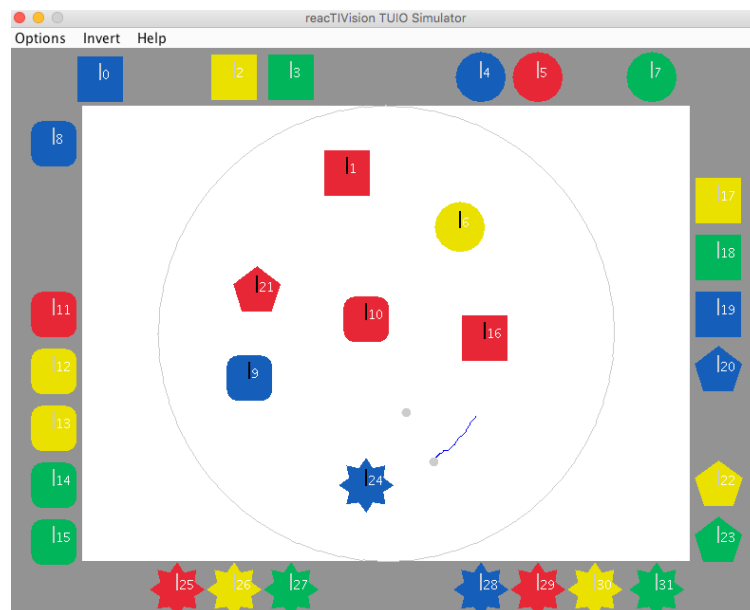


Fig 3.6 - TUIO Simulator in action

TUIO Simulator is a small app developed in Java and provided by tuio.org (more information in *References* section, [11]) that offer to developers a simple way to create tangible surface applications without the need of having a physical table. When a fiducial (one of the 31 figures showed on the borders) is set on the table (white part of the screen) an OSC message is sent to the

corresponding client, where the information is processed and some action is performed. Apart from fiducial tracking, it also offer finger tracking, but it is not useful for our purposes due to some synchronisation problems with our development environment.

3.2.2.2. TuioPad

<https://itunes.apple.com/us/app/tuio-pad/id412446962?mt=8>



Fig. 3.7 - TuioPad (iPod version)

This small iOS app is an open source tracker that can interact with TUIO clients located on the same network. Implementing the 1.1 version of TUIO protocol, this application simulates a tangible surface, capable of detect multi-touch interaction as a real table with reacTIVision could do. That way, it is possible to manage finger tracking in a development environment in an easy way, sending the corresponding information (as OSC messages) to the client (in our case, a C# client) in order to be processed properly.

In spite of TUIO Simulator offers finger tracking, this app has been particularly useful for our project, giving us a real time response that the first one is not able to achieve.

3.2.3. Backend persistence

In a video game certain information must be stored. Not only player data, such times and scores, but also parts of the game as images, sounds, or even great part of the logic that defines it. Taking into account the different nature and formats of the information (unstructured data), a traditional relational database couldn't offer the required flexibility. So the use of a NoSQL database is recommended, due to its scalability, simplicity of design, high response and velocity.

There are many different solutions based on NoSQL on the cloud that offers very useful additional services (such as cloud computing, push notifications, ACL or analytics) for processing the data and dealing with the logic of a game. For this project, the best solution could be Parse, but due to its imminent disappearing, another option must be taken. In the next section a brief analysis of different alternatives is carried out, justifying the final selection.

3.2.3.1. Appcelerator Arrow



Fig 3.8 Appcelerator & Arrow logo

Appcelerator Arrow is a framework to build and deploy applications on the cloud, offering two different services: Builder (design and deployment of custom client APIs) and Cloud (database and push notifications service).

This solution provided by Appcelerator (a private held mobile technology company) is a part of Appcelerator Platform, which is focused in mobile engagement and provides cross-platform native development and a Mobile Backend as a Service (MBaaS) as main products. It is highly opinionated and used in a wide range of industries.

Appcelerator offers a full programming environment available for Windows and Mac OSX. Thanks to its IDE and Titanium SDK it is possible to program in JavaScript and deploy apps natively, having the opportunity to visually design with AppDesigner. Additionally, several APIs are available in order to interact directly with different platforms and external databases.

3.2.3.2. FireBase



Fig 3.9 Firebase logo

Firebase is a mobile platform for designing high quality apps in a quick and easy manner, offering services as a Real-time database, cloud messaging, authentication, analytics or notifications. This solution is provided by Firebase (a cloud computing company) and it is considered one of the most populars Backend as a Service (BaaS).

The real-time database (RTBD) is fully designed in JSON (JavaScript based), being highly effective in real-time updating. Thanks to different SDK's, Firebase can be used to develop native and web-based apps for the top-most mobile operating systems (Android & iOS). A REST API completes the options offered.

3.2.3.3. GameSparks



Fig 3.10 GameSparks logo

GameSparks is, in words of its creators “*The #1 Backend-as-a-Service Platform for Games Developers*”. This platform offers, apart from a Database as a Service, multiple interesting features, such as cloud computing, analytics, notifications, ACL, player management, social and multiplayer options, etc.

Focused mainly as a backend for games, GameSparks offers one integrated environment compatible with different game engines (Unreal and Unity are the most relevant), platforms (Android, iOS, etc), game consoles (PlayStation and Xbox), and even the topmost social networks and some CMS.

Thanks to the integrated API it is possible to work with different languages: C++, C#, Java, Objective-C and JSON, apart from the Cloud Coding. MongoDB is the database used to manage all the data.

3.2.3.4. mLab



Fig 3.11 mLab logo

mLab is a cloud database service built on the top of MongoDB, running on different cloud providers such as Amazon, Google and Microsoft Azure. In simple words, it could be defined as a Database as a Service (DaaS).

The main goal of mLab is to offer its original MongoDB-as-a-Service with automated provisioning and scaling, backup and recovery, monitoring and web-based administration. That way, a developer only has to deal with the data, without worrying about server management tasks.

mLab offers different SDK's in order to be operative with different languages through specific drivers: C, C++, C#, Java, NodeJS, Perl, PHP, Python, Ruby and Scala. It is also possible to connect with the database via MongoShell (command mode) or mLab Data API (REST). All the data is stored in BSON (Binary JSON).

3.2.3.5. Backend study summary

As a quick comparison, the relevant data extracted from the study is shown in the Table 3.1.

Table 3.1: Backend comparison

	Kind	Languages	Client SDK / API	Cost
Appcelerator Arrow	MBaaS	Java Objective-C JavaScript JSON	Titanium, NodeJS, iOS (Objective-C), Android, Mobile Web REST API	from 36\$/month
Firebase	BaaS	Java Objective-C JavaScript JSON	iOS (Objective-C), Android, Web REST API	Spark version: free - Up to 1GB (RTDB) - 5GB for files
GameSparks	GBaaS	C++ C# Java Objective-C AS3 JSON Cloud Code	Action Script, Android, C++, cocos2d-x, iOS, JavaScript SDK, Marmalade, Unity, Unreal Engine REST API	Evaluating & Prototyping: free Indie & Student: free to build & launch game, pay on success (more than 100000 active users/month)
mLab	DaaS	C C++ C# Java NodeJS Perl PHP Python Ruby Scala JSON	C, C++, C#, Java, NodeJS, Perl, PHP, Python, Ruby, Scala MongoShellmLab Data API (REST)	Sandbox (development & prototyping) : free - Up to 500 MB - Single database

3.2.3.6. Backend selected: mLab

Bearing in mind that there are more options than the evaluated, we can assure that from the options analysed GameSparks and mLab are two great solutions for this project. Once is decided that Unity is the game engine, and C# the language, Appcelerator Arrow and Firebase were discarded.

Reactive Games, as the name explicitly say, is a game based system. That way, GameSparks could fit perfectly (having a dedicated Unity SDK), but the designed games are not thought to be commercial fun games, but serious. Reactive Games are conceived to be played on a tangible board, and not to be distributed for mobile platforms. The concept of multiplayer is restricted to a physical environment and the only interaction with the cloud is (almost, in this stage of the project) to store player data in order to evaluate not who is the best, but the progress on learning about the topics touched in each game.

mLab offers all that is required: a cloud database with minimal server configuration, and a driver that let our game engine store data, dealing with great part of the game logic.

3.3. External libraries and services

Different libraries must be used in order to develop this project: one is the TUIO client for C# and Unity; another is the C# driver for MongoDB. Additionally, the interaction with an external service for implementing a simple Text To Speech (TTS) solution is needed.

3.3.1. Uniducial. Unity3D TUIO framework application

Uniducial is a third-party library that implements a complete framework for allowing games developed with Unity3D (under C# programming language) interact with tangible table interfaces based on reacTIVision.

This is more than a TUIO 1.1 standard client: it gives the developers a full fiducial detection and control. Uniducial includes different scripts that can be easily integrated along with Unity:

- *Fiducial Controller*. Basic actions performed by figures attached to fiducials (shifting, rotation, and show/hide object) are transformed into information by TUIO Manager class (and its parent classes). This script is in charge of process the data acquired and act consequently. With a simple editor interface, the game object controlled by this script can be attached to a concrete fiducial, deciding which level of control is necessary.
- *TUIO Manager*. This class (along with other that implements TUIO and OSC protocol and the client) is in charge of detect and process OSC messages about physical objects in real time.

Also, two different groups of scripts are included in Uniducial. The first (TUIO Connector) includes all the logic to implement a client and the protocol, along with OSC. The second is Editor, which let the developer to define the rules for Fiducial Controller script using the variables/information shown on the Unity Editor.

3.3.2. C# MongoDB Driver

mLab is a DaaS which uses as database engine MongoDB. In order to interact with it, Mongo team offers a wide range of official drivers for the top most programming languages. For this development the C# version is used. The driver offers a full API that allows to perform CRUD operations (that are create, retrieve, update and delete documents).

MongoDB Driver has different parts:

- *Driver*: a completely async driver to communicate with MongoDB databases.
- *GridFS*: the distributed filesystem used by MongoDB to store files (images, sounds, etc).
- *Core Driver*: the base library of the driver. Useful to create new custom driver APIs.
- *BSON Library*: a standalone library for serialisation tasks performed over MongoDB.

There are differences between traditional relational databases and NoSQL. Some minor, like the fact that what we know as tables and registers in SQL in NoSQL are called collections and documents. But the most remarkable is that in NoSQL different kind of registers are allowed for the same collection, while in SQL register type is predefined in time of table creation.

In previous versions of the driver (< 2.0) the core driver doesn't exist and GridFS is integrated with the driver.

3.3.3. Voice RSS: a free online TTS service

One important aspect to take care of in a game is the feedback given to the player. But in the case of teaching games for kids, it must be accurate enough to assure that the help is completely understandable and simple. In order to offer the maximum facilities to the children, guiding them to the whole game, two kind of feedback are implemented: visual and audible.

The first one, described on subsection 4.3.1, is based on written messages and images. The second, apart from provide specific sounds (advices), is in charge of transforming this messages in audio, giving a second way to get the feedback. To achieve that we make use of a Text To Speech service: VoiceRSS.

VoiceRSS is a TTS online web service that transforms, in real time, the sent messages in audio files corresponding to the speech of the texts. This service offers a simple API that can work with POST and GET HTTP protocols, supports 26 languages (Catalan, Spanish and English are

included), offers the possibility to get the results in different audio formats (from 8 kHz - 8 Bit - Mono to uLaw - 44 kHz - Stereo) and is compatible with the most common audio codecs (MP3, WAV, AAC, OGG, CAF).

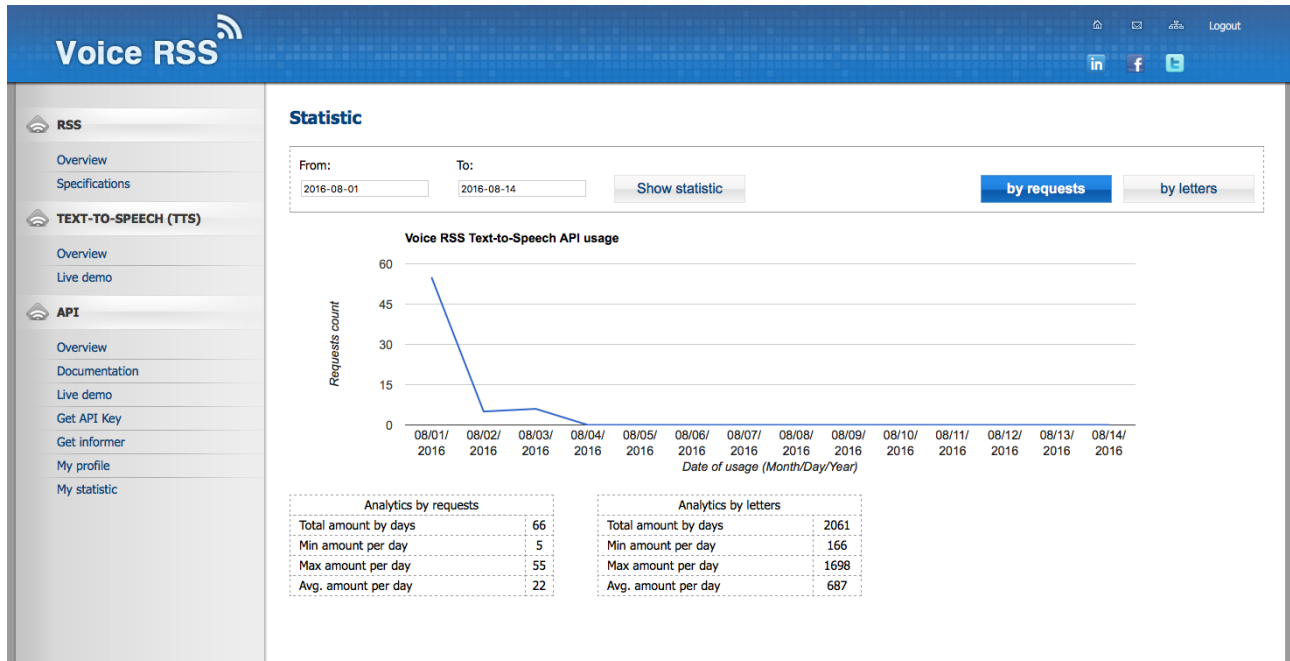


Fig 3.12 - Voice RSS interface. Statistics & analytics section

To use this API we must follow this URL syntax:

`http://api.voicerss.org/?<parameters>`

`https://api.voicerss.org/?<parameters>`

Parameters includes a personal key (a valid user is required to get it), source text, language, speed of speech, audio codec and audio format. Only the key is mandatory (if a parameter is not specified, the default value is used).

This service has a free plan with limitations (350 daily requests in plain text), but is a perfect tool to get a simple but effective TTS in our prototypes.

4. Reactive Games. Physical system.

4.1. Description

Reactive Games is a tangible table based system that offers educational 2D games destined for teaching geometry to children.

The system has to differentiated parts:

- *The Management Module.* This part gives the system administrator a tool for registering new players, insert new geometric figures for the geometry game and finally, create new tangram patterns to solve.
- *The Game Machine.* This part let that a registered kid can play one of the games existing on the platform.

The evolution of every player is registered in a cloud backend (and locally, if going offline is necessary), giving the professional (teacher, psychologist, etc) the opportunity to evaluate the data to determine if the kid is improving his/her skills on geometry. This information could be important in order to detect problems related with geometric thinking on children and find a way to solve it.

To play the games children have to manipulate physical representations of 2D figures solving different activities. Thus, with Reactive Games it is possible to evaluate shape recognition and teach how this figures interact to build complex ones. At the same time, important aspects of geometry such as the vocabulary or the different shape features (angles, sides, etc) are taught.

4.2. Environment

Being a TUI prototype, two elements are mandatory:

- *A tangible table surface:* the SONA table (consult Section 3.1.3 for further details).
- *Objects attached to fiducials.* Predefined figures corresponding, by one side to the 7 basic Tangram shapes and on the other, a set of basic geometric shapes (triangles, quadrilaterals, circles and other polygons) created with a 3D printer.

Two more elements attached to the SONA table are part of the physical environment and to complete the system:

- A *screen projector*. This is used to give visual feedback to players: instructions, help and error/success messages.
- A *sound system*. Provides an audible version of the feedback (speech of the written information) and different sounds as error/success messages.

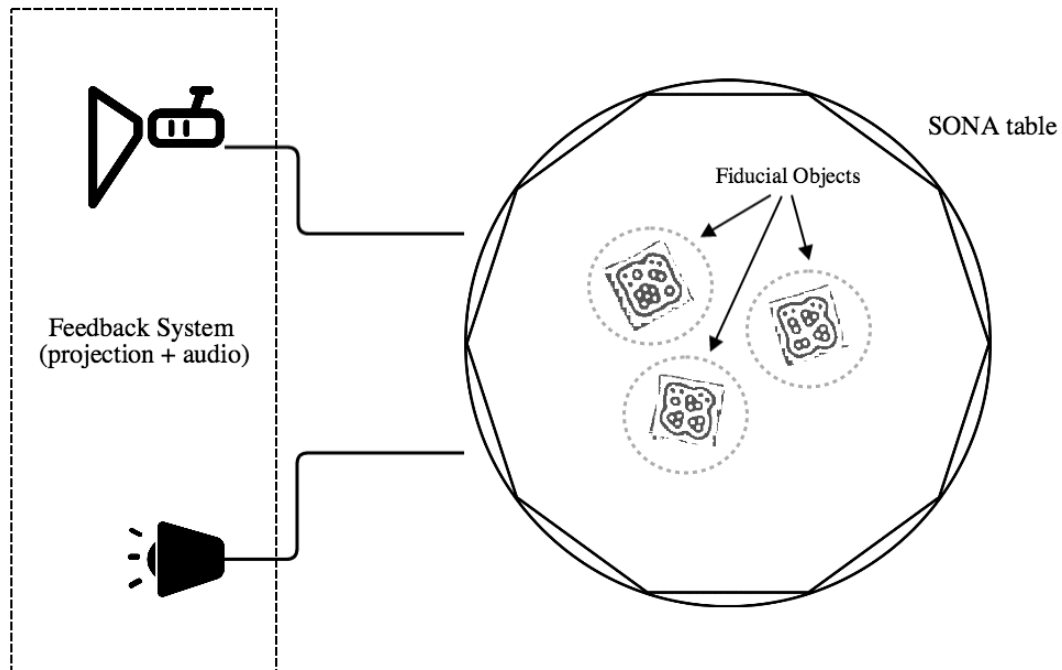


Fig 4.1 - Reactive Games scheme

4.3. Interface design

Once the physical environment is defined and implemented, the next step is to determine the look and feel of both the projection screen on feedback system and SONA interface must be.

4.3.1. Basic projection screen

This part corresponds to the visual feedback system (projection screen), and is usually located on the left side of the environment. Here is where all the messages related with the game, such as instructions, help or game progress must be shown with the use of the external projector. Apart from the feedback, this screen is used for managing purposes (creation of new users, figures and tangram patterns) and to access to users and game selection (as is shown on Fig. 4.2).

In order to make easier these tasks, leaving the tangible table for the interaction with the games, all the control elements such as buttons or drop-downs are controlled using an external keyboard and/or mouse on this screen.

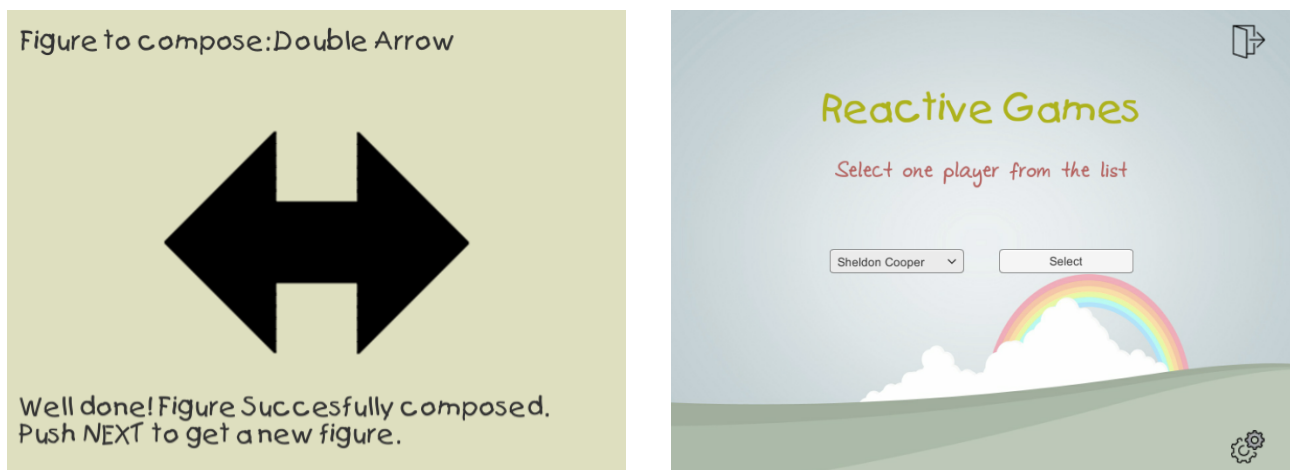


Fig 4.2 - Projection Screen. Feedback sample for Tangram game (left). User selection screen (right)

4.3.2. SONA basic interface

This is the part of the interface shown on the top of the tangible table. Basically, is the place where the games take place, so must handle the user interaction through objects attached to fiducials and fingers. No keyboard or mouse interaction is allowed, in order to preserve the nature of the table (tangible and interactive).

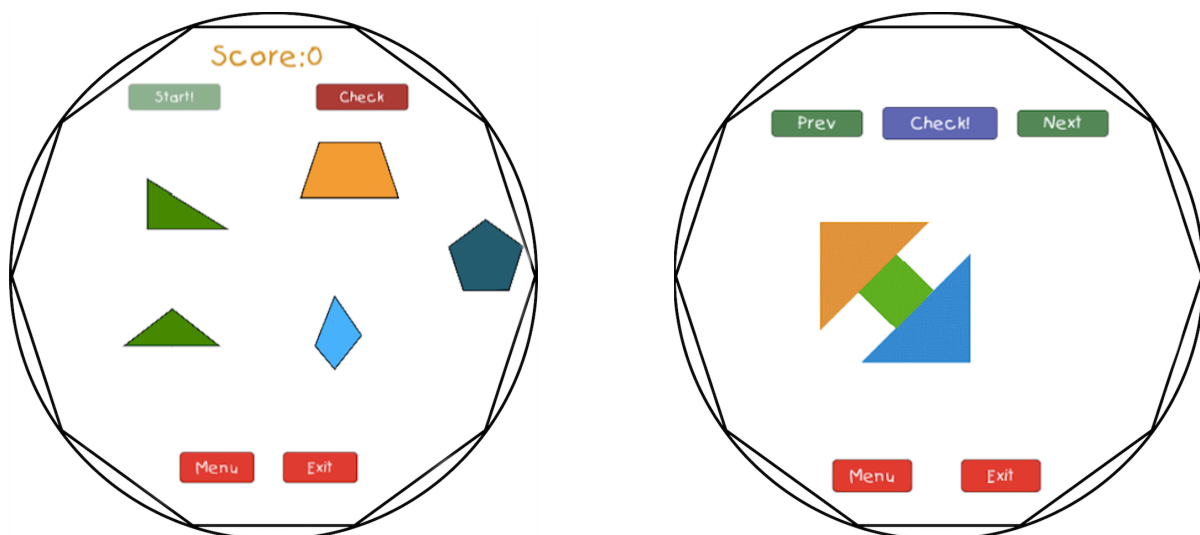


Fig 4.3 - SONA interface samples. Geometry game (left). Tangram game (right)

4.3.3. Base configuration

It is necessary to perform a configuration process in order to get both parts of interface working properly. That is carried out establishing a pattern scene with two cameras: one for the table (Main Camera), one for the projection (Secondary) and one canvas attached to each of them. Every one has its own properties:

- *Main Camera & Canvas.* This camera shows the game interface on the surface of SONA table, and is located as the right half of the screen. Referring to the canvas, it must hold UI as any other object on the scene in order to react to the physical interaction over the table. For that reason, the render mode must be set to *World Space*.
- *Secondary Camera & Canvas Messaging System.* This camera corresponds to projection screen, so it is located on the left half of the scene shows the information of the feedback system and selection/configuration options. In this case, the render mode of the canvas is set to *Screen Space* (the normal working mode, where UI elements are rendered on top of the scene).

Finally, it is important to remark that as we are working with a 2D environment, both cameras must be set to orthographic projection mode, having a viewport rect width of 0.5 (one half for each part of the interface).

4.4. Software

4.4.1. Fiducial and finger tracking

In order to process all the information provided by reacTIVision application Unity uses the TUIO framework Uniducial (see Section 3.3.1 for details). Creating the corresponding prefab as base figure (with Fiducial Controller script attached) is the starting point to get full control on figures tracking (position, rotation, etc), as it is shown on Fig. 4.4. Obviously, every figure has attached different shape representation, so the prefab must be adapted for each one of them.

Although Uniducial is also prepared to process finger tracking, some modifications must be made in *Fiducial Controller* script in order to get it working.

As fiducials have an ID in the range between 0 and 99, numbers out of that range can be used to detect finger tips on the table. In our case, taking into account that an external mouse has no sense on a tangible environment, we need to simulate a cursor on SONA to detect when the player put his finger on the surface and when a click is made. Only one simultaneous finger need to be tracked, so assigning a fiducial ID -1 as value a simple detection is achieved.

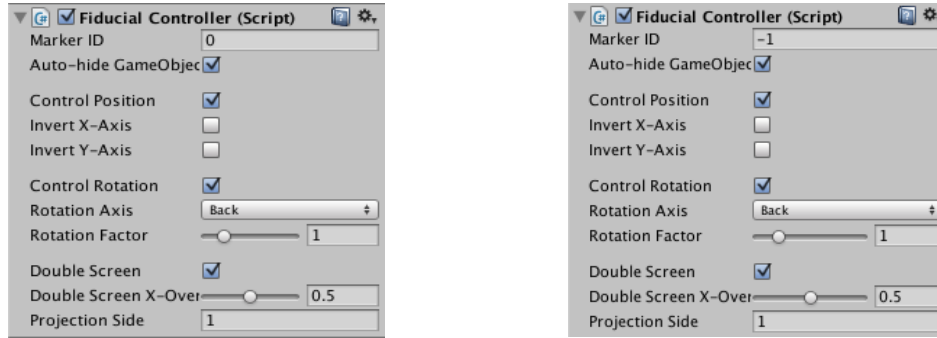


Fig 4.4 - Fiducial Controller script options on editor. Fiducial assignment (left), finger assignment (right)

Getting that behaviour on our table implies, by one side, modify Uniducial scripts in order to properly process the detection of a finger. But on the other, a new input controller must be created: the cursor controller.

One important thing to be noticed is shown on Figure 4.4. In order to indicate what half of the scene must contain the fiducial and finger detection by reacTIVision, and to show the marker properly on the surface we need three parameter:

- *Projection side*: set to 0 to indicate the left half (1 for the right).
- *Double Screen*: indicates that scene is divided in two different parts.
- *Double Screen X*: indicates the factor to be applied to x marker position. By default, reacTIVision set x position between 0 and 1 (a real number). In our case, we are only using the half part of the scene, so in order to show the marker in the correct place a conversion factor is needed.

Source code for the improved version of Fiducial Controller could be found in Appendices A1.2.

4.4.2. Incorporing finger tracking to Fiducial Controller

Fiducial Controller script checks if a marker is detected (a fiducial on the table) on every frame. Then, a new TUIO Marker Object with different parameters (such as position, rotation, movement direction, etc) is created and prepared to be shown. When detection is lost, the object must remain hidden to the interface.

As the detection is only for markers, we must include the cursor detection (for enabling finger tracking). Once is decided that a (unique) cursor corresponds to an ID with value -1, it is possible to determine if the detected is a cursor or a fiducial, enclosing each case in an if-else condition, and creating a new TUIO Cursor Object instead of a TUIO Marker Object in case of finger tracking.

```

// Dealing with a cursor (finger)
if(this.MarkerID == -1) {
    if(this.m_TuioManager.IsConnected && this.m_TuioManager.GetCursorCount(>0)
    {
        TUIO.TuioCursor cursor = this.m_TuioManager.GetCursor();
        this.m_ScreenPosition.x = doubleScreenXOverlay +
            cursor.getX()*(1-doubleScreenXOverlay);
        ...

        //set game object to visible, if it was hidden before
        ShowGameObject();
        //update transform component
        UpdateTransform();
    }
    else {
        //automatically hide game object when marker is not visible
        if (this.AutoHideGO)
            HideGameObject();
        this.m_IsVisible = false;
    }
}
// Dealing with a marker (fiducial)
else {
    if (this.m_TuioManager.IsConnected && this.m_TuioManager.IsMarkerAlive(this.MarkerID))
    {
        TUIO.TuioObject marker = this.m_TuioManager.GetMarker(this.MarkerID);
        ...
    }
}

```

Fig. 4.5 - Modified code of Fiducial Controller to get full cursor detection

With that little modification on the *Update* method, single finger detection and tracking is feasible, obtaining an improved version of the script. For further information, full source code is attached on Appendices A1.1.

4.4.3. Cursor controller

Unity's event system is the way that messages are sent to objects in reaction to some external input (such a keyboard, mouse, touch, etc). Through different input controllers in combination with different Raycasters it is possible to generate an interaction between external devices and the game objects, defining the correct messages.

One of the most common used input module is the *Standalone*, which is in charge to process input mouse events: movement, clicks on different buttons and hold actions such as drag and drop.

In our case, we need to simulate the cursor in order to respond to finger touches on SONA table. Taking into account that the UI is considered as another game object on the scene, creating a custom input module is the best solution. As a good starting point for our purposes it was very interesting and helpful the article of Peter Koch *VR Gaze Input*, about the creation of an input module for controlling gaze tracking for Oculus glasses [26].

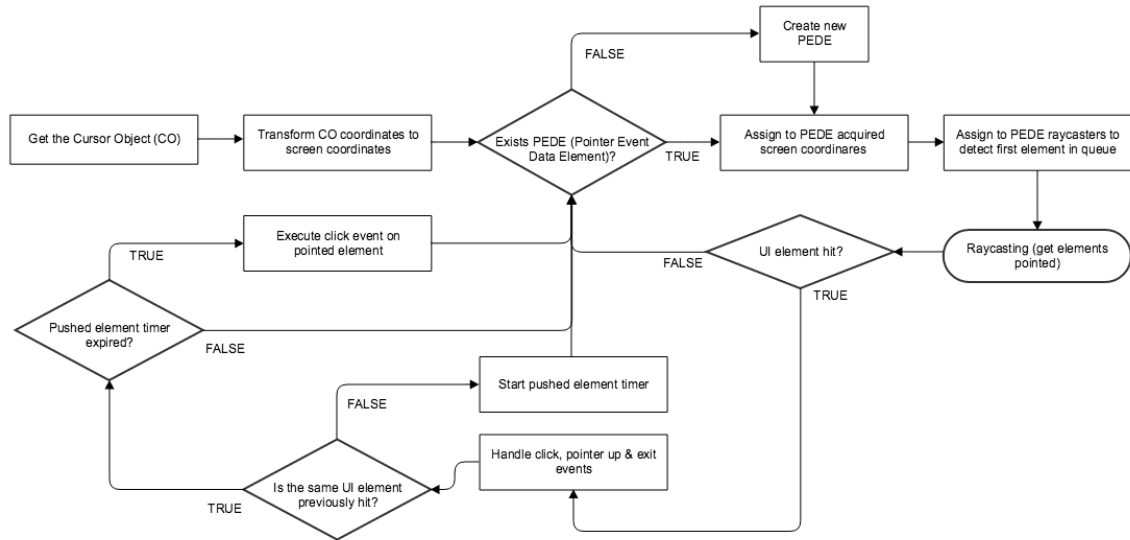


Fig. 4.6 - Schema of the Cursor Input Controller behaviour

The new input module is based on *Pointer Input Module* and is capable to process the movement of the cursor and handle simple and single click events in order to interact with the buttons of the user interface. On Fig. 4.6 a scheme of how the input controller must works is shown. Source code that implements this scheme is available on the Appendices A.1.1.

4.4.4. Local Storage

Although Reactive Games is prepared to work online (using mLab as backend solution), it is interesting that the system will be fully operative in case that have to work offline. A simple Local Storage solution is implemented taking advantage of the file management and serialization methods offered by Unity and C#. To use them, we only need to:

- Make serializable all the classes used to query the backend, in order to store them.
- Select a location to save all the data locally. In our case, we use the persistent application data path, creating a folder called *ReactiveGames* for our purposes.

We are storing two kind of data locally. Firstly, each class that represents a collection on mLab is stored as a single file with *.rg* extension in the root of our folder. Secondly, images needed to represent figures in games are placed in the *rg_images* subfolder. It is important to notice that images are only downloaded if there are no local copy of them.

In order to get both data repositories synchronized, each time the game is started online, all the local files are updated with the backend information. The only exception is done with statistics: they are always stored locally and uploaded to the backend when the system is started in online mode. Once statistics were stored, local file is cleaned. Once all the updates are finished, the system works in local storage mode, in order to avoid delays on the gameplay (i.e., when a query is performed).

5. Information architecture

In order to manage the information of this project, we made use of a backend (mLab) that let us organise and store the all the related data, and great part of the logic. In our case, we need to deal with:

- *Common System data*: fiducials supported, games.
- *Player information*: basic data (such as name or age) and statistics (scores, attempts, time used to solve the problem, etc).
- *Game data*: figures used (properties, assigned fiducial, etc), levels and specific data.

An explanation about how the information is organised could be found in the next sections.

5.1. Player information

Reactive Games is a multi game machine for teaching children geometry. It is important to get data about the evolution of the players in order to know if learning progress is correct and, if it isn't, detect it and act to solve the problem. Due to that, users must be registered in the system, to store some statistics related with their experience while playing the games, in order to carry out further analysis.

There are two collections defined for that purposes: Players and Player Statistics.

5.1.1. Players collection

Contains all the needed data to identify a user.

Table 5.1 - Definition of Players collection

Field name	Description	Type
<code>_id</code>	Unique player identifier. Auto assigned on insert	BSON Object ID
<code>name</code>	Player's name	string
<code>surname</code>	Player's second name	string
<code>age</code>	Player's current age	integer
<code>gender</code>	Player's gender (0 female, 1 male, 2 undetermined)	integer

`_id` is used in the games to identify the player, while parameters such as *gender* or *age* could be useful for establishing patterns along with statistics.

5.1.2. Player_Stats collection

Stores useful data in order to determine player's learning evolution.

Table 5.2 - Definition of Players_Stats collection

Field name	Description	Type
_id	Unique identifier for this register	BSON Object ID
player	Player's unique identifier	string
game	Game's unique identifier	string
level	Level identifier	integer
question	Question's unique identifier	integer
time to finish	Time spent to finish the question / level	float
attempts	Number of attempts used to finish the question / level	integer
points	Score achieved	integer
figures used	Number of figures used to solve a question / level	integer

It is important to notice that some of the fields have different relevance depending on the game. For example, in Tangram Game each level is a problem to solve (but does not have different questions for each level). Another special case is the number of figures used. In Geometry Game can be useful to determine how much better is an answer used to solve a question: suppose the question was 'Put only quadrilaterals on the table', and there are 5 figures of this kind. Putting from 1 to 5 figures on the board could be already correct, but the second answer is more accurate. That is not a problem in Tangram, because using all the available figures (7) is mandatory.

5.2. System data

5.2.1. Games collection

Collection used to get all the games identified by its name and unique id.

Table 5.3 - Definition of Games collection

Field name	Description	Type
_id	Unique game identifier. Auto assigned on insert	BSON Object ID
name	Game's name	string

5.2.2. Fiducials collection

In order to detect a figure on Reactive Games, the use of markers is mandatory. Considering that we want to attach figures to fiducials dynamically and add new ones (if necessary), we created Fiducials collection.

Table 5.4 - Definition of Fiducials collection

Field name	Description	Type
<code>_id</code>	Unique fiducial identifier. Auto assigned on insert	BSON Object ID
image ID	Identifier of the image used for reactTIVision to detect the fiducial	string
fiducial ID	Identifier used for reactTIVision to detect the fiducial (unique)	integer
is assigned	Determines if that fiducial is already used	bool

Each fiducial has its own identifier and image, established by reactTIVision framework, so we need this two parameters to use it. The image can be retrieved from *GridFS* (the MongoDB file system to store media) using its ID. *GridFS* is represented in database as two collections: *fs.files*, where references and metadata of the media is stored, and *fs.chunks*, where the media is stored as BSON binary data.

Also, as we want that only one fiducial can be assigned to a figure, *is_assigned* let us know if we can use that.

5.3. Game data

For each game, we need to store not only specific data, but also part of the logic used to rule them. So, for Geometry Game we use two collections: GG_Figures and GG_Levels, and for Tangram we use another two: Patterns and Connections.

5.3.1. GG_Figures collection

This collection is used to store all the features of each figure on Geometry Game, as long as the reference to the image that represents the shape and the reference to the fiducial that have to be attached to it.

Table 5.5 - Definition of GG_Figures collection

Field name	Description	Type
<code>_id</code>	Unique figure identifier. Auto assigned on insert	BSON Object ID
image ID	Identifier of the image used for representing the figure	string
fiducial ID	Identifier of the attached fiducial	integer
features	Different features related with the figure	JSON Object

Referring to *features*, this JSON Object includes different attributes that defines a figure. These are:

- *GeometricShape*: a boolean value that determines if the shape is a geometric figure.
- *Polygon*: a boolean value that determines if the shape is a polygon.
- *Regular*: a boolean value that determine if the shape is a regular polygon.
- *Sides*: an integer value for the number of sides of a polygon.
- *Vertexes* : an integer value for the number of vertexes of a polygon.
- *KindOfShape* : a string that determines if the figure is a quadrilateral, a triangle, a curved form or a polygon (in the case it has more than 4 sides/vertexes).
- *ByDegrees* : a string used on triangle forms, to determine its kind depending on the angles degrees (right, obtuse, acute).
- *BySides* : a string used on triangle forms, to determine its kind depending on the relation between sides (equilateral, isosceles, scalene).

5.3.2. GG_Levels collection

In this collection is stored the logic of Geometry Game.

Table 5.6 - Definition of GG_Levels collection

Field name	Description	Type
<code>_id</code>	Unique level identifier. Auto assigned on insert	BSON Object ID
activity	Identifier of the activity (level)	integer
questions	Texts in different languages that define the question to solve	JSON Object
attempts	Maximum attempts to solve the question	integer
base points	Minimum points for each question on this activity	integer
areas	Solution to the question (how figures must be placed on board)	JSON Object

This game is divided in different activities (or levels), and each one is composed by different questions. The solution to each question is stored in a JSON object (*areas*). As it will be explained on section subsection 6.2.4.2, the board is divided in different areas (depending on the question), so to solve it, a specific kind and number of figures must be placed inside each area. That way, *areas* is an array that contains from 1 to n spaces that represents the board distribution with the following information:

- *AreaID*: an integer that represents the number ID for the area
- *Operation*: a string value with two possible values (OR, AND), used to specify the relation between area goals.
- *AreaGoals*: array that contains the different goals for an area, identified by two parameters.
 - *Definition*: string that defines the goal, coincident with the possible features of GG_Figures.
 - *Value*: one of the possible values for each feature.

This distribution let us store the logic behind the solutions, having a standard way to get that information for different cases.

In order to guarantee that, almost the instructions given on this games were multilingual, the JSON Object *Questions* contains simple objects that store the problem definition by each given language (for our prototype, in English, Catalan and Spanish).

5.3.3. Patterns collection

This collection is used to store all the basic information about a Tangram pattern: the unique ID, a name that describes the pattern to solve and finally, the reference to the image that represents it.

Table 5.7 - Definition of Patterns collection

Field name	Description	Type
<code>_id</code>	Unique pattern identifier. Auto assigned on insert	BSON Object ID
<code>name</code>	Name of the represented pattern	string
<code>image ID</code>	Reference to the image that represents this pattern	string

5.3.4. Connections collection

Here we can find all the logic of Tangram Game. As will be explained on Section 6.2.5.3, a tangram pattern is composed by 7 figures that interact between them in order to obtain a solution to the puzzle. This interaction takes place when:

- a) one side of a piece touches one side of another piece
- b) one vertex of a piece touches a side of another piece
- c) one vertex of a piece touches a vertex of another piece

Taking into account that a side is the connection of two vertexes, it is possible to determine the composition of a pattern as a group of connections of kind b) and c). With this premises, Connections collection is defined as is shown in Table. 5.8.

Table 5.8 - Definition of Connections collection

Field name	Description	Type
_id	Unique connection identifier. Auto assigned on insert	BSON Object ID
pattern	Reference to the pattern to whom belongs the connection	string
V1 vertex	Vertex ID of the first figure	integer
V1 figure	Figure ID of the first figure	integer
V2 vertex	Vertex ID of the second figure	integer
V2 figure	Figure ID of the second figure	integer
S figure	Figure ID for the figure which side is touched	integer
S vertex1	Vertex ID of the first vertex that is part of the touched side	integer
S vertex2	Vertex ID of the second vertex that is part of the touched side	integer

6. Reactive Games. The Core

Once we have defined the environment and established the information architecture, we have to describe how is designed and implemented the core of Reactive Games. In there, two parts must be differentiated: the management module and the game machine.

6.1. Management module

6.1.1. Description

In the management module the professionals (teachers, psychologists, etc) can perform different tasks related with the main information of the system: user management, creation of new geometric figures for Geometry Game and insertion of new Tangram patterns.

From the main screen of RG, the access to the management menu is done by clicking the wheels located on the bottom right corner. It is important to notice that management options are only available when the system is online (otherwise, this option is invisible). Once the user is on this screen, it is possible to select one of the three available management options.

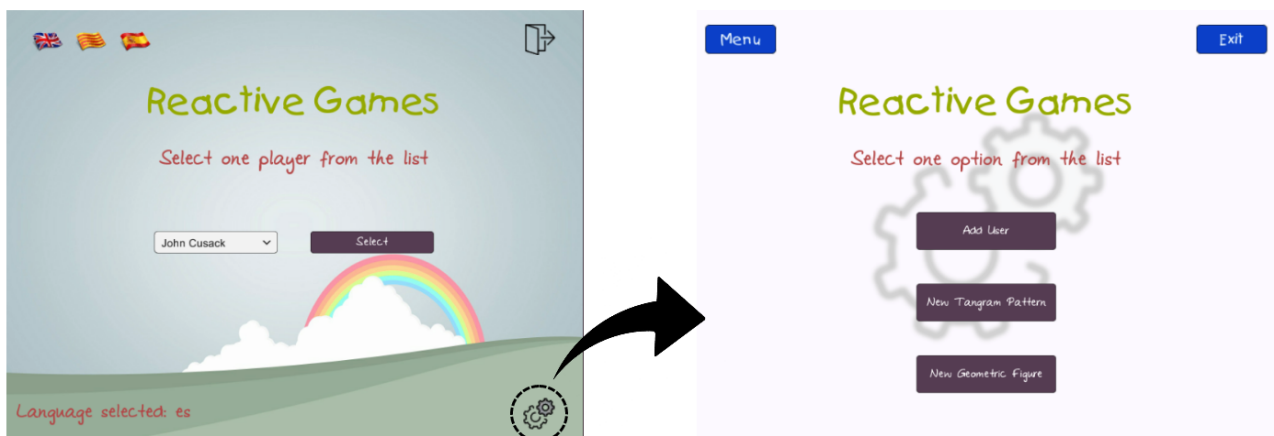
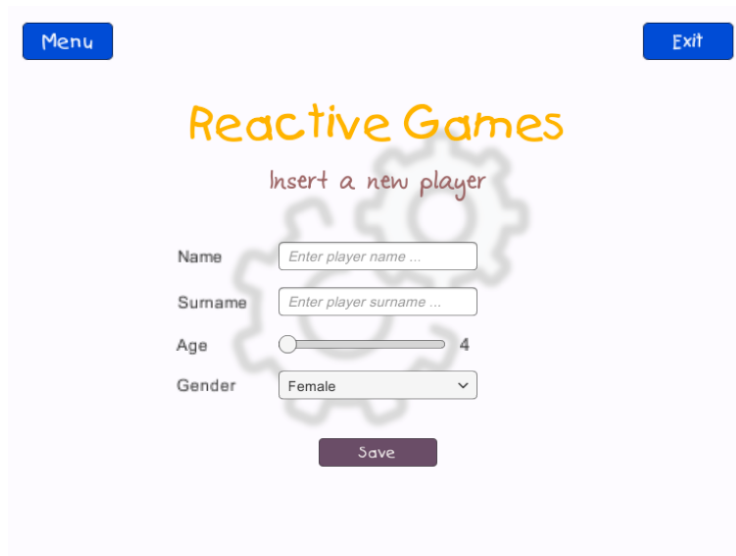


Fig. 6.1 - Accessing to Reactive Games Management menu

6.1.2. User Management

In order to get data of player's learning progress, a register of users is needed. With this module, professionals have an easy way to register new people on the system, being possible to process their evolution on geometric knowledge in the future.



The interface for creating a new player is titled "Reactive Games" in orange. It features a "Menu" button in the top left and an "Exit" button in the top right. Below the title, the instruction "Insert a new player" is displayed in red. The form includes input fields for "Name" and "Surname", both with placeholder text "Enter player name ...". The "Age" field is a slider set to 4, and the "Gender" field is a dropdown menu currently showing "Female". A "Save" button is located at the bottom center of the form.

Fig. 6.2 - New player creation screen

The interface design is shown on Fig 6.2. Located in the projection screen, it is quite simple, and offers the required options to store basic data for a person. It is important to notice that this module only creates the profiles: the statistics will be attached automatically to them as soon as each player participate in one game.

6.1.3. Geometric Figures Management

As we want to offer professionals the opportunity to improve Geometry Game, one of the most interesting options to be given is a form to insert new geometric figures on the system. That way, it is possible to create more questions, while the levels could be increased in difficult, trying to make quizzes more interesting to the players.



The interface for inserting a new geometric figure is titled "Reactive Games" in green. It features a "Menu" button in the top left and an "Exit" button in the top right. Below the title, the instruction "Insert a new geometric figure" is displayed in red. The form is divided into two main sections. The left section contains checkboxes for "Geometric Shape", "Polygon", and "Regular Polygon", all of which are checked. Below these are sliders for "Sides Number" and "Vertices Number", both set to 3. There are also dropdown menus for "Kind of Shape" (set to "Triangle"), "Class. By Side" (set to "Equilateral"), and "Class. By Angle" (set to "Acute"). The right section has an "Image Path" input field containing "print/Geometry/Equilateral_blue.png". Below this, there are two visual representations: a blue triangle labeled "Figure Representation" and a black outline of a triangle with internal points labeled "Fiducial To Assign. ID: 28". A "Save Figure" button is located at the bottom right of the form.

Fig. 6.3 - Screen design for new geometric figure insertion

Figures management interface is located in the screen projection side, and offers the most common features of a geometric shape on the left. On the right panel, when the user puts the full path for the image that represents the new figure to upload, two thumbnails appear below: the image itself (to check that is correct) and the fiducial image and id (last free one on database) to be assigned to the new figure. This information is useful in order to locate, print and attach this fiducial representation to the physical shape that will interact with the system.

6.1.4. Tangram Pattern Management

Reactive Games comes with few playable tangram patterns by default. But, couldn't be interesting insert new ones? That's why this module is designed. Having a simple way to create new tangram patterns, players always have new milestones to overcome and professionals more opportunities to evaluate them.

For this case, we made use of both interface parts. On the projection screen, the user can insert the full path of the image that represents the new pattern and a clear name to identify it. Once the path is filled, the image is shown in order to get the reference.

A professional (manager) must solve the pattern using the tangram figures on the SONA table. That way, it is possible to generate all the connections (between vertexes and sides) that define the new pattern. For further information about how this is done, check Section 6.2.5.3. Once the form is filled and the figure composed, all the information is ready to be sent to the database.



Fig 6.4 - Tangram pattern creation & insertion interface

One important thing to remark is the functionality of *flip* button. Due to the fact that the parallelogram figure (in yellow) can't be mirrored (see Section 6.2.5.1 for details), a solution to use it and compose certain patterns is to flip horizontally the pattern. That way, it is possible to compose (practically) all kind of tangrams.

6.2. Game Machine

6.2.1. Requirements

The Game Machine is where the players can select a game and play to it. Although in this prototype only two games are available, it is possible to integrate new ones following certain guidelines and restrictions:

- The games must be related with mathematics.
- Difficult must be adapted to the objective public: kids between 4 and 12 years old.
- Being games for children, the interface must be clear enough and visually attractive. Also, the mechanics of the game must be as simple as it can be.
- Incorporate visual and auditive helps in order to make easy the gameplay. When indications about what to do should be given, this is especially important and useful.
- It is mandatory that the game incorporate some way to store data about the player progress in each game.
- Indications must be available in three languages: English, Catalan and Spanish.

6.2.2. User and game selection

The screen of user selection is the entry point to Reactive Games. Here is where using a drop-down menu all registered users could be selected.



Fig. 6.5 - User Selection Screen

Once the user is selected from the drop-down and *Select* button pressed, game selection screen appears. Here is where the player can select the desired game in order to start it.



Fig. 6.6 - Game Selection Screen

6.2.3. Language Selection

In the main screen (*User Selection*, shown in Fig. 6.5) the user can choose the language by clicking the flag buttons (top left corner). It is important to notice that, in this version of the prototype, only information messages on Geometry Game are adapted to the selected language.

6.2.4. Reactive Geometry Game

6.2.4.1. Description

This game, composed of different questions organised in levels (activities), tries to put to test different skills related with geometry:

- Connection between names and the shapes.
- Shapes recognition.
- Identification of figure features, in order to classify the shapes.

The prototype has available a set of 28 figures: 6 triangles, 7 quadrilaterals, 5 polygons (more than 4 vertexes/sides), 6 circular shapes and 4 non geometric figures. This basic set could be incremented, if necessary, using the figure manager described in Section 6.1.3.

6.2.4.2. Figure detection algorithm

To complete a question on Geometry Game, pieces must be put on the board grouped in different areas. To be concrete, it is possible to divide the board from 1 to 4 parts, and each one has defined goals: properties that the shapes must have to fit in the area. It is possible, thanks to the logic implemented (shown in Section 5.3.2), to get different goals simultaneously for an area (i.e. *figure must be a triangle and must be scalene*), giving to the system certain degree of freedom to create complex questions.

So, we define the areas and the goals to determine which figures can be placed in there. To give the player freedom to play (and to make game easier), the areas doesn't have a specific position on the board. A question will be correctly answered if all the pieces placed on the board, inside the defined areas, follow the rules of the area goals. Let's think about a simple question such as *put on one side polygons and non polygons on the other*. In that case, two areas are clearly defined, but it really does not matter if polygons are on the right side or on the left, and non polygons on the opposite side.

To achieve that behaviour, we need two group of areas: the first contains all the goals that must be accomplished, and the second the solution for each area given by the player. Having that, we can check if the figures given in each solution follow the rules defined in one of the area goals. Only if all the solutions fit with all the goals, the answer given is correct. The algorithm followed to do that is shown below as pseudo-code:

- *For each area*
 - *Compare each possible solution with the goals for this area*
 - *If any solution satisfies an area goal, left the comparison*
 - *Else, continue comparison for the next area*
- *If all areas have their goals satisfied by the solutions given, the answer is correct*
- *Else, the answer is wrong*

6.2.4.3. Mechanics

Game mechanics are simple: the player must place on the top of the tangible table as much correct figures as he/she can in order to solve each question before reaching a maximum number of attempts. Consuming all the attempts force the player to solve the level/activity from the start. If the question is solved, another one is shown, and this go on until a level is finished. The Fig 6.7 shows a diagram that clarifies the explanation given.

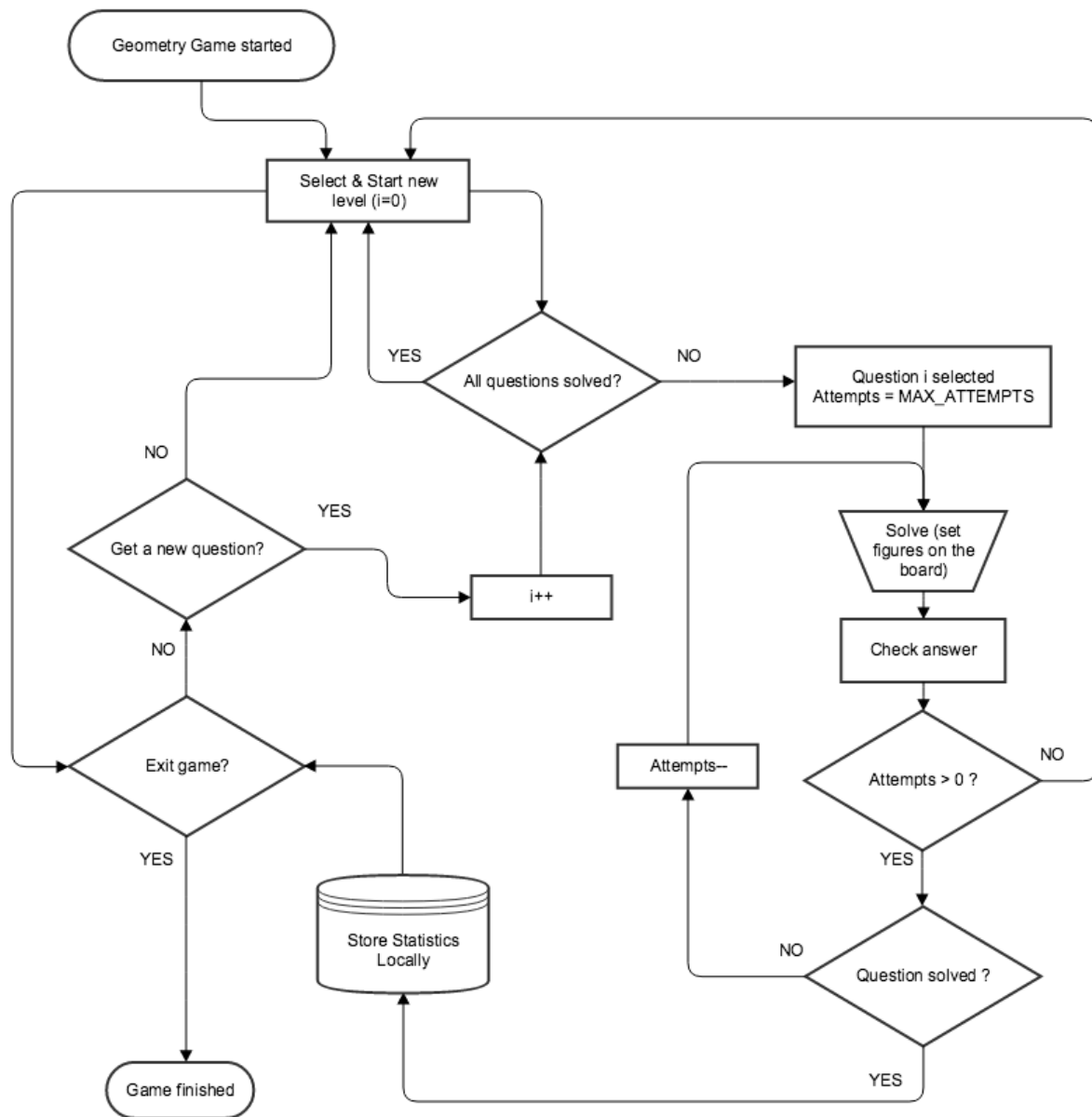


Fig. 6.7 - Geometry Game flow diagram

Every time that a question is solved, information about the progress is stored locally: number of attempts used, time to give the correct answer and the partial score acquired (calculated by multiplying the number of attempts left by the base point for this level). Additionally, this partial score is accumulated to get the game score, shown on the tangible screen.

In order to solve a question, certain actions must be done:

- Before pushing ‘*Start*’, the board must be clean of figures. Otherwise, the system gives and advice to remove the pieces that remain on top of the surface.
- To check a solution, the button named ‘*Check*’ must be pressed. Whether the solution is correct or not, information about the results and what to do is shown in the projection screen.
- Once ‘*Start*’ button is pressed, a timer is started. Although it is stopped every time that player wanted to check the solution, only is reseted when a new question appears.

Finally, its important to remark that instructions shown in the projection screen are also spelled by the game (thanks to TTS system), in order to offer the player another option to get the information. Also, some representative sounds are played when something happened (for example, when an answer is wrong or right).

6.2.4.4. Screen Design

As is described in Section 4.3, games on RG use two interfaces: the projection screen (to show information or options) and the SONA interface (where the interaction and the game take place). In Geometry Game, that scheme is followed at 100%, so the screen design fits perfectly with this pattern.



Fig 6.8 - Geometry Game activity selection screen

After entering in an activity, a previous screen is shown (see Fig. 6.8). In there, the user can select between 6 levels/activities related with different geometry concepts by pressing the buttons labelled

‘*Activity x*’. For each one, a battery of questions is selected and shown in a random way in the game (SONA) screen. Another possibility is select all the questions mixed, pressing the button called ‘*All Activities*’.

On projection screen different messages are shown while playing:

- The question to solve and/or some instructions to figure it out.
- The number of attempts left to answer the question.
- Information about the figures (Are figures placed correctly? Is there any extra figure on the board? Is there almost a figure on the board?).
- Information about the progress (question solved, level finished, wrong answer...).

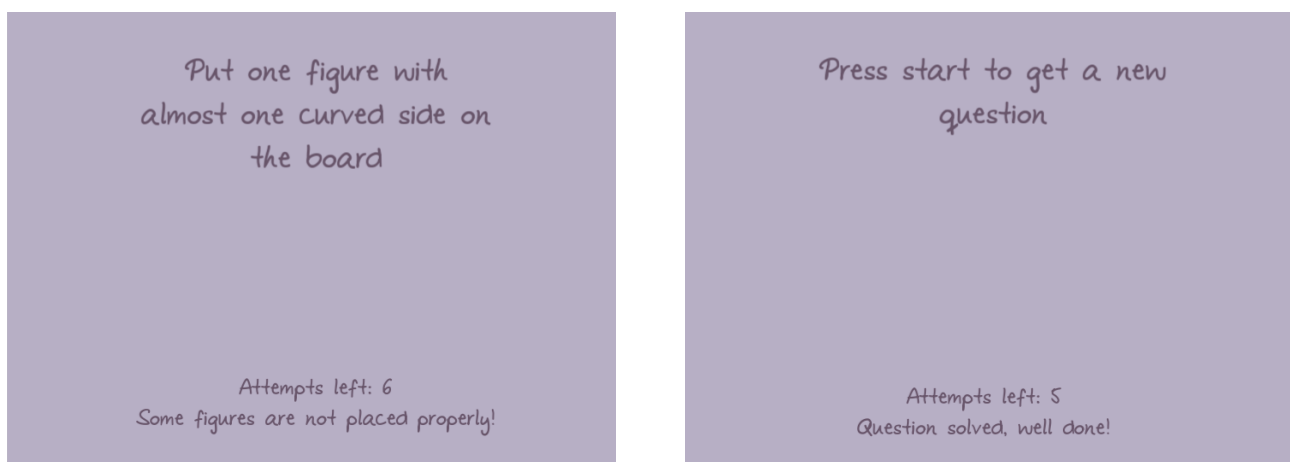


Fig 6.9 - Screen projection for Geometry Game. Different messages for various situations along a game playing

On the other side, the design of what is shown on the SONA surface is quite simple:

- On the centre of the screen, the board can be divided between one and four zones, depending on the nature of the question. This is the place where the physical figures were represented by its own version, surrounding them in order to know that they are really on the board.
- On top of the screen, we can find the global score and two buttons, the first used for start solving the question (*Start*) and another to check if the result is correct (*Check*).
- On the bottom, two buttons are placed. One to return to the activity selection screen (*New Activity*) and another to exit the game and RG system (*Exit*). A third button appears over them, to get the user the option to replay this activity from the scratch (*Reload Activity*) when the attempts are zero.

Some examples of screens design are given in Fig. 6.10 and a complete view of the complete interface (projection + surface) in Fig. 6.11.

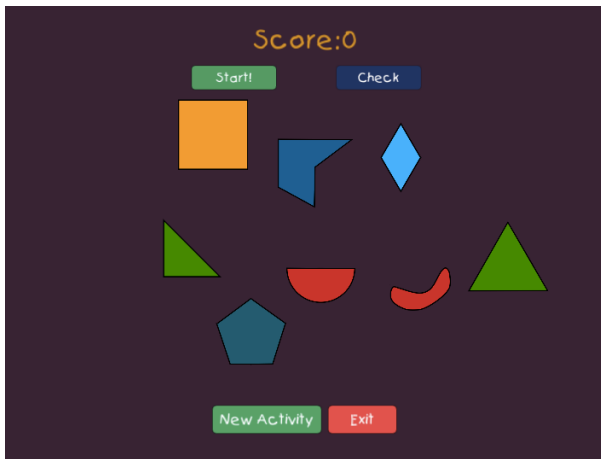


Fig. 6.10 - SONA interface. Screen design showing the results of interaction over the board for some questions

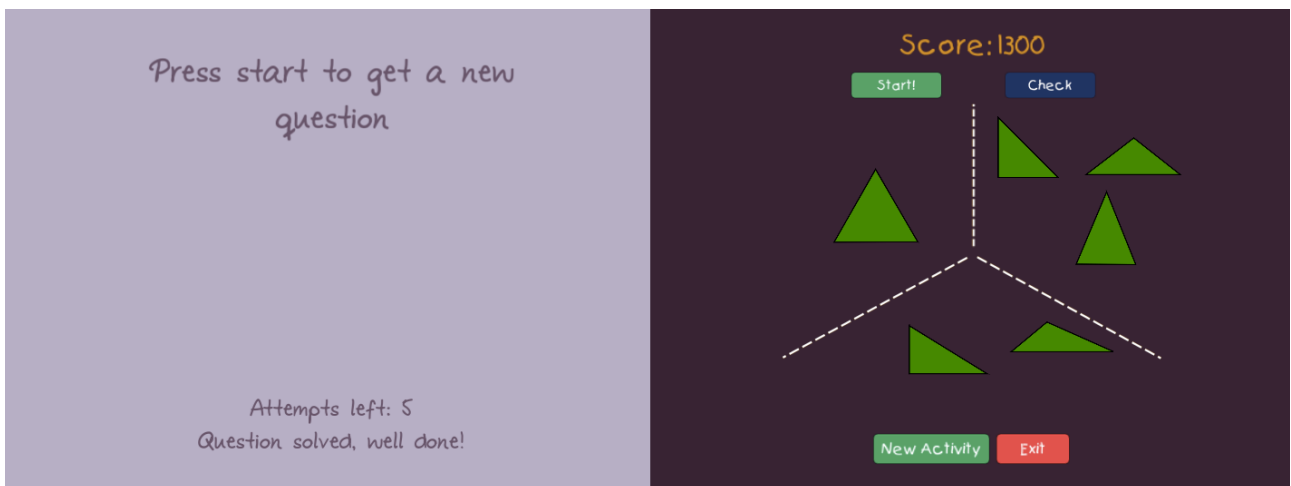


Fig. 6.11 - Geometry Game full screen design

6.2.5. Reactive Tangram Game

6.2.5.1. Description

Tangram is an ancient Chinese game (that can be considered a puzzle) where the main goal is to compose different silhouettes of figures (patterns) with 7 geometric figures given. This pieces are the result of dividing a square of side 1 in a certain manner (see Fig. 6.12). Certain peculiarities must be known:

- All the triangles are isosceles, and there are 5:
 - Two big (a, b), with sides of $1/\sqrt{2}$ and the hypotenuse of 1
 - One medium (c), with sides of $1/2$ and the hypotenuse of $1/\sqrt{2}$
 - Two small (d, e), with sides of $1/2\sqrt{2}$ and the hypotenuse of $1/2$

- 1 square (f) , with side of $1/2\sqrt{2}$
- 1 parallelogram (g) with sides of $1/2$ and $1/2\sqrt{2}$.
- All pieces can be mirrored only by rotation, except the parallelogram.

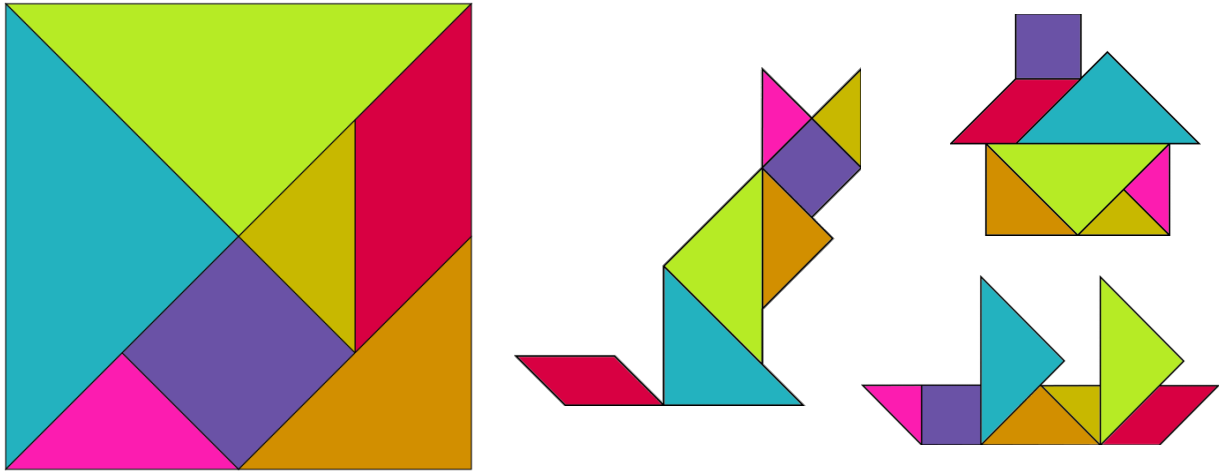


Fig. 6.12 - Tangram base square divided in the 7 pieces and different composed figures

There are hundred of figures that can be composed: animals, buildings, numbers... In the figure 6.12 we can see three representative samples.

6.2.5.2. Mechanics

The main goal of this game consists on solving each proposed puzzle. In order to do that, the player must put all the pieces on top of the tangible table and combine them in a proper way (moving or/and rotating them). There are some considerations about how to construct a figure:

- Pieces that are equal are interchangeable, and the result can be already correct. As an example, if we take a look at the *cat* pattern shown on Fig 6.12, it is possible another solution by putting big triangle (a) in the place of big triangle (b) and vice-versa.
- It does not matter if the solution is a figure rotated in comparison of the original pattern. The result is checked by comparing coincidences between pieces on their sides and vertexes.

In that game, it is not mandatory to solve every tangram in order, so the player can go from one to another and solve the one that he/she wants in every moment. But when a figure is composed, it must be checked in order to determine if the solution is correct. That way it is possible to get some little statistics, such as time to finish or the attempts used in doing it. All that data is stored locally every time that the player change the figure to solve, but only if he/she tried to give a solution.

Finally, it is interesting to notice concrete behaviours behind some actions while playing:

- Once a figure is selected (pushing ‘*Next*’ or ‘*Previous*’ buttons), a timer is (re)started. Although it is stopped every time that player wanted to check the solution, only is reseted when a new figure is selected.
- To check if a solution is correct, the button ‘*Check*’ must be pressed. Whether the solution is correct or not, information about the results and what to do is shown in the projection screen.

6.2.5.3. Pattern creation/checking algorithm

Mechanics for tangram are quite simple, but we need a way to computationally construct and check different figures in a standard way.

As a game based in mathematics (to be concrete, geometry), different complex algorithms were created in order to know if a figure is well constructed in comparison with a pattern. An interesting approach is proposed by *Oflazer, K.* [23]. For this prototype, a simplified algorithm is created.

Some premises must be considered:

- We have different pieces that are equal, so its possible to interchange them and the result must be the same.
- Pieces can be rotated. That is important, especially in two cases: the square and the parallelogram. If we rotated the first, the result must be the same, as it has 4 equal sides. A rotation on the second (of 180°) gives us the same shape, so the result cannot vary.

A simple but effective way to compute a pattern is to check how the vertexes and sides of each piece interact. To concrete, we look at the connections between vertexes of different figures (vertex connections) and between vertexes and sides (side connections), storing a figure composed as a list of connections. To do that, it is necessary to number each vertex of every figure, as it is shown on the figure 6.13. Doing that, we already have numbered each side, as they are made by the union of vertexes (side 12 is formed by the vertexes 1 and 2).

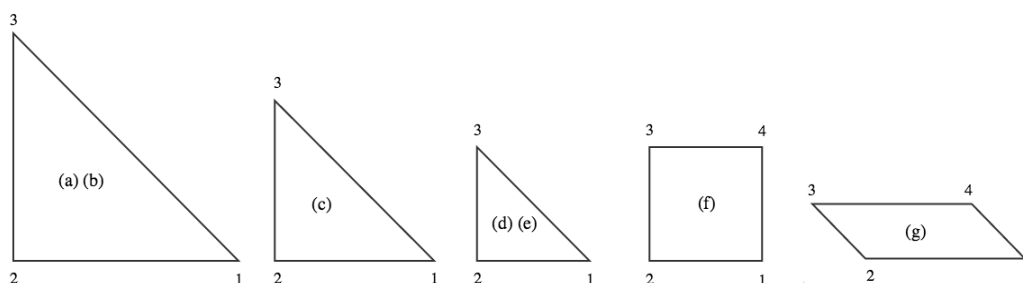


Fig. 6.13 - Tangram pieces with numbered vertexes

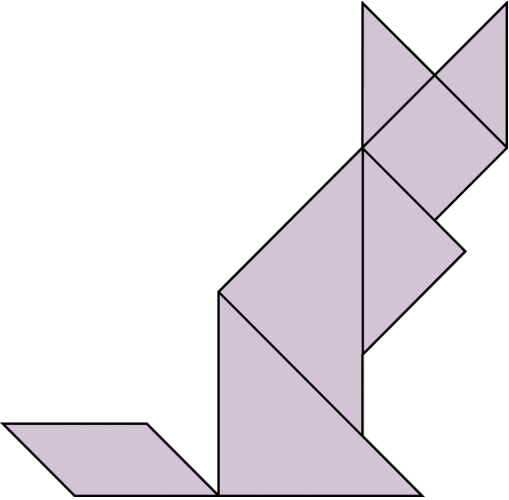
Following this idea, creating a pattern is easy. Only one thing must be done, in order to reduce the volume of information: save only one coincidence of vertexes or/and side. The same way that we construct the pattern, the solution is processed, so the same number of connections must result in both cases.

In the case of checking if a figure composed by a user is the same than a pattern, comparison can be done in a proper way following this algorithm (pseudo-code):

- *If the solution given does not have vertex connections nor side connections, solution is wrong*
- *Else, we must compare if solution vertex connections are equal to pattern vertex connections*
- *If they are not equal, solution is wrong*
- *Else, we must compare if solution side connections are equal to pattern side connections*
- *If they are not equal, solution is wrong*
- *Else, the tangram is successfully solved*

When we say ‘equal’, we must take into account the premises given few lines before. As an example, in Table 6.1 we can find the list of connections after processing the figure shown.

Table 6.1 - List of connections vertex-vertex and vertex-side for the given figure

(a) v1	(b) s13	
(a) v2	(b) v3	
(a) v3	(c) v1	
(a) v3	(d) v3	
(a) v3	(f) v2	
(b) v2	(g) v1	
(c) s12	(f) v1	
(d) v2	(e) v2	
(d) v2	(f) v3	
(e) v2	(f) v4	

6.2.5.4. Screen design

As is described in section 4.3, and detailed when described the interface for Geometry Game (section 6.2.4.4), Tangram Game follow the same design based on two screens.

In the case of the projection screen different messages are shown:

- Information about the name of the figure to compose.
- A pattern image (in black) representing the figure.
- Information about the progress (figure composed, wrong solution, etc).

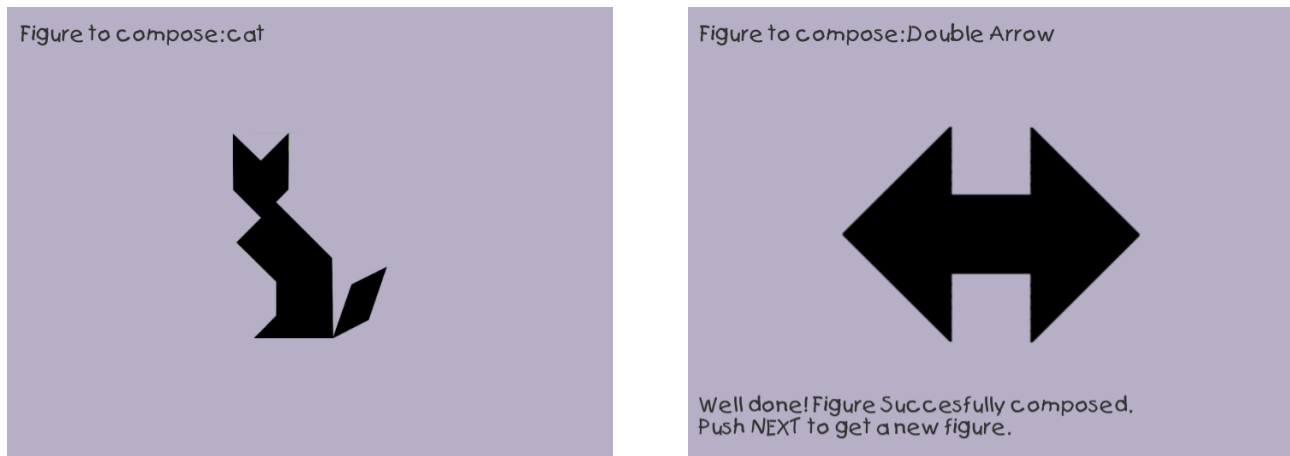


Fig. 6.14 - Projection screen design samples for Tangram Game

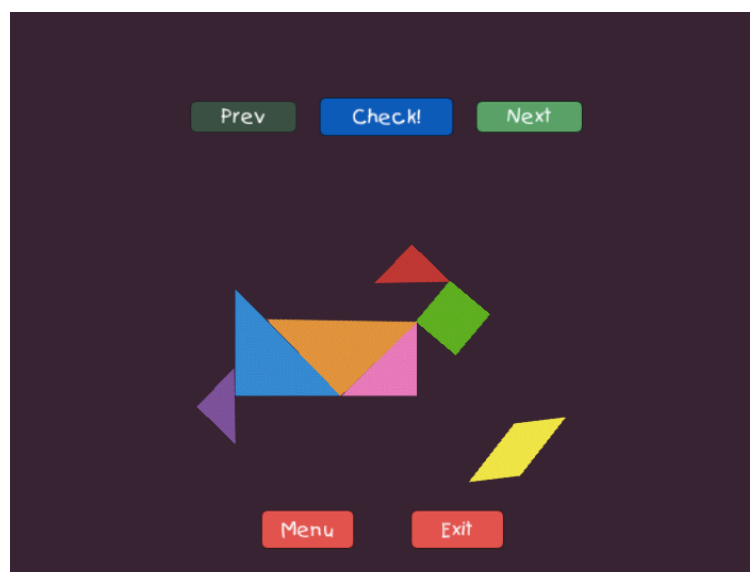


Fig. 6.15 - Construction of a tangram figure

Referring to the design of what is shown on the SONA surface, we can see:

- On the centre of the screen, the composed figure is available. To be more precise, the silhouette of each physical piece is surrounded by its image, in order to know that is placed on the board.

- On top of the screen, we can find three buttons: two to go to the next figure (*Next*) or to the previous one (*Prev*), and a third button (placed between the others) to check if the composed figure is correct (*Check*).
- On the bottom, two buttons are placed. One to return to game selection menu (*Menu*) and another one to exit the game and RG system (*Exit*)

An example of a figure composition is shown in Fig. 6.15 and a view of the complete interface (projection + surface) can be found on Fig. 6.16

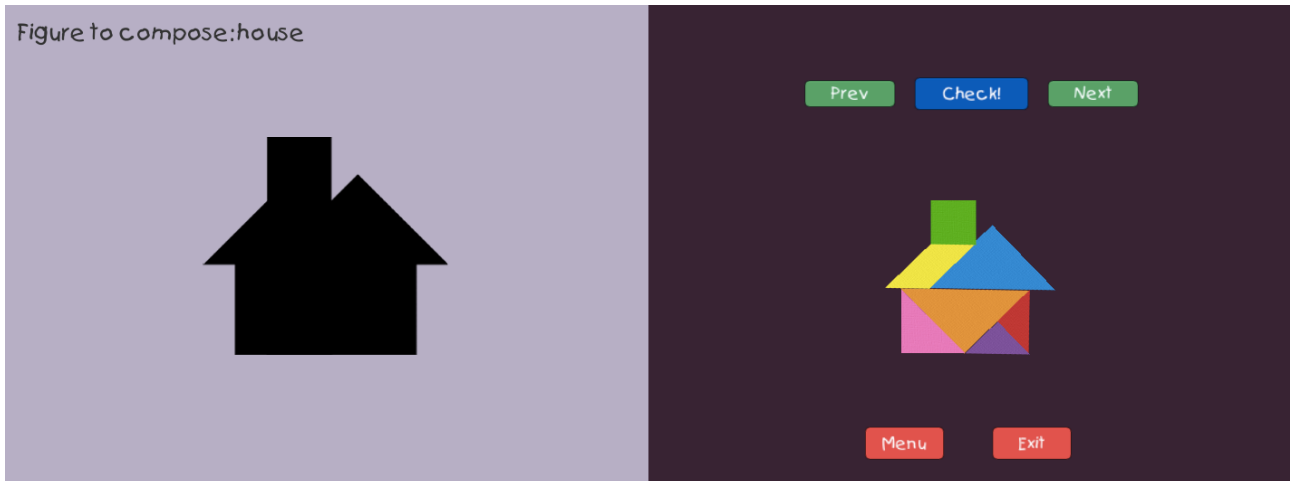


Fig. 6.16 - Tangram Game full screen design

7. Reactive Games. Implementation

After researching and designing the prototype, the time of implement it and make the tests has come. Right here, it is possible to prove if all the development is correct, see if there are some limitations and try to improve the results in order to get the prototype working.

7.1. The prototype. Final screen design

Once we have fully functional implemented system, we can verify if the prototype looks like the initial design. Following this lines, different figures show the final result in the real world.

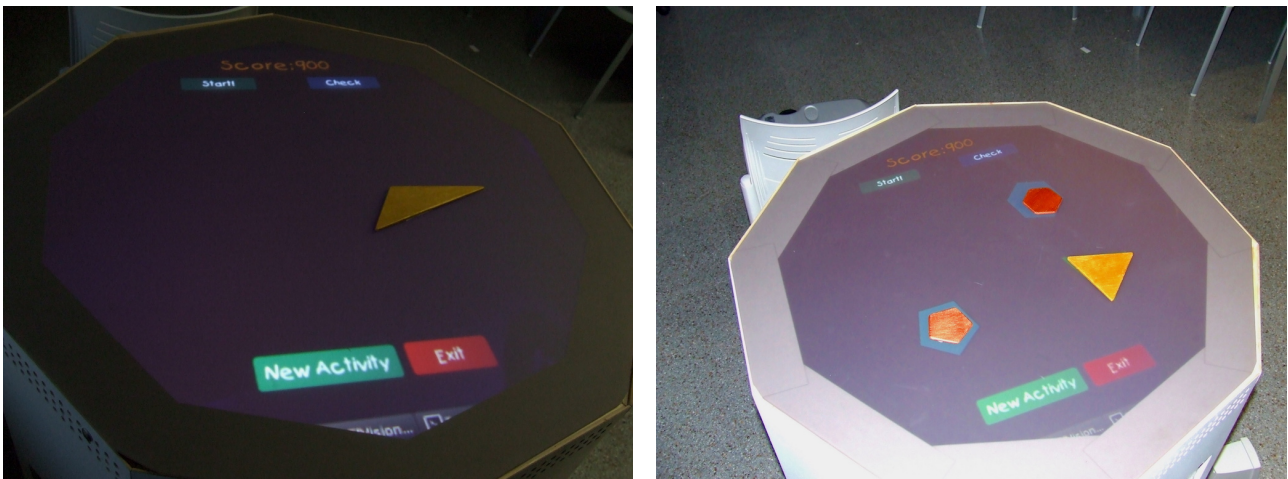


Fig. 7.1 - Different views of SONA interface for Geometry Game



Fig. 7.2 - Full system. Playing Geometry Game

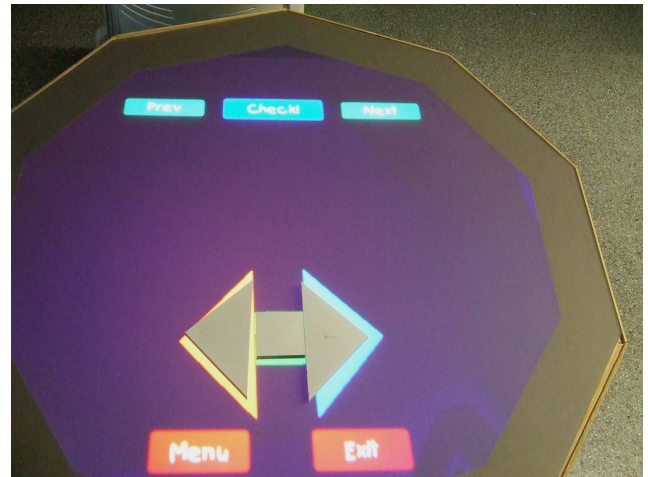
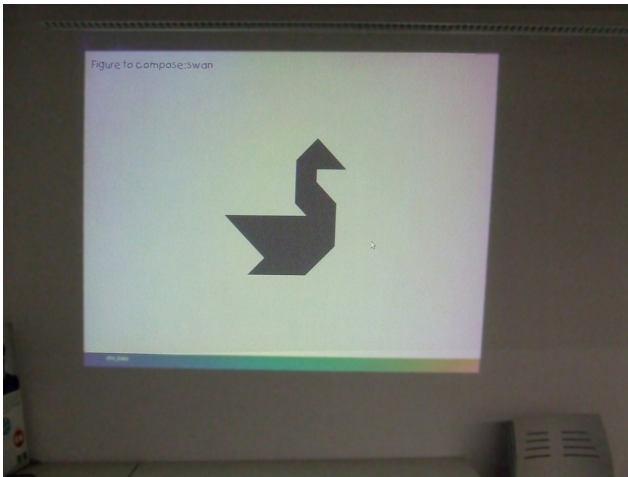


Fig. 7.3 - Projection screen (left) and SONA interface (right) for Tangram Game

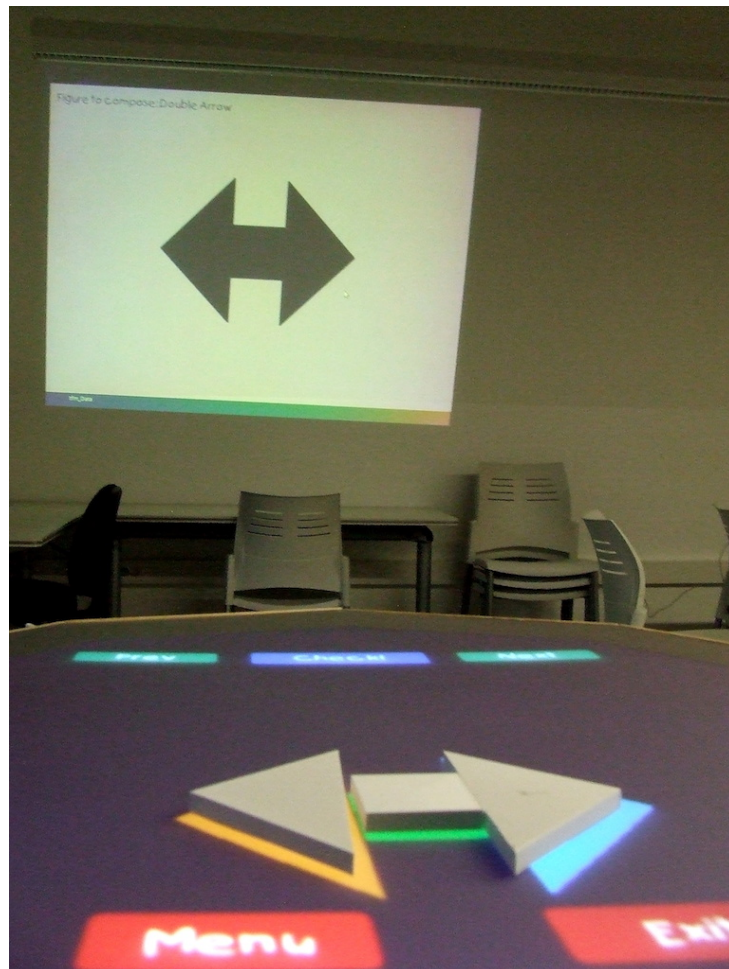


Fig. 7.4 - Full system. Playing Tangram Game



Fig. 7.5 - User & language selection (left) and Game selection (right)

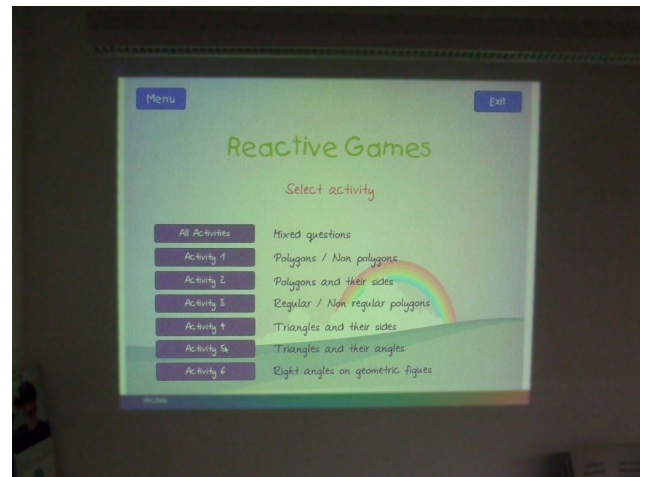
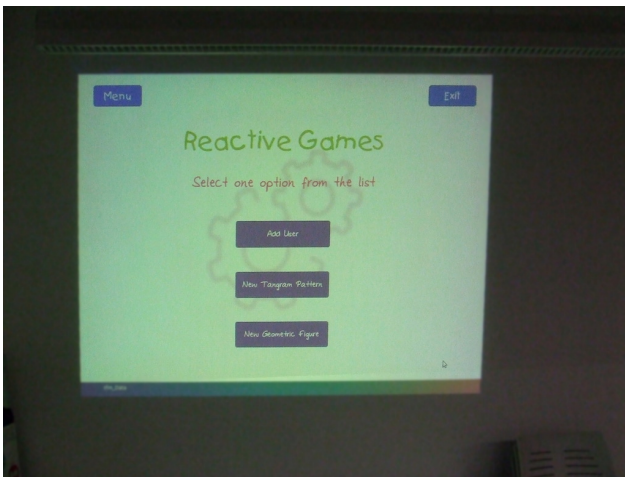


Fig. 7.6 - Management option selection (left) and Geometry Game Activity selection (right)



Fig. 7.7 - Different pieces used in Reactive Games

7.2. Limitations and problems detected

7.2.1. Calibrating the system

One of the first actions to be taken was calibrating the table in order to detect the fiducials properly. Although it was originally configured and prepared to work as a Reactable, some physical improvements were made in order not to make possible Reactive Games runs perfect, but also the whole system.

To achieve it, camera, mirrors and projector were re-adjusted to get a better fiducial detection and clearer projection on top of the table. That implies to adjust reactTIVision application, in order to correct different distortions, as is shown on Fig. 7.8.

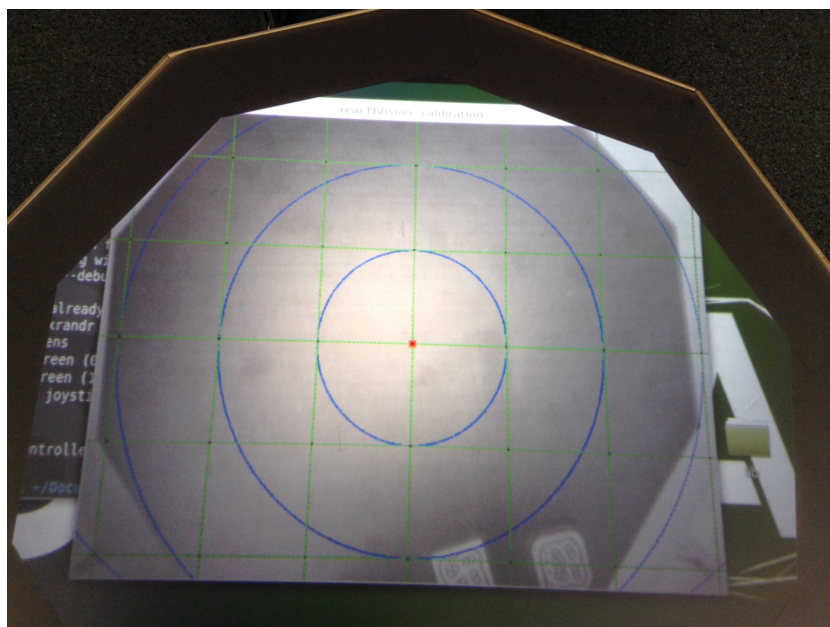


Fig. 7.8 - Calibrating the system with reactTIVision application

Finished that process, the system is ready to work. Although a complementary calibration by software could improve the results performing little adjusts in an easy way, or correcting distortions produced by the mirrors (that can deforms the surface projection), if the components of the table (mirrors, camera, projection or infrared lamps) are not set correctly this method could be useless.

7.2.2. Figures and fiducials physical size

Testing RG is very simple. We need to create a physical representation of each figure involved on both Geometry and Tangram games, attach to them the corresponding fiducials, and play.

We need that figures cover the fiducial in order to hide it, so their size is conditioned. Taking into account the dimension of the fiducial image, and the shapes that we have to represent, some figures must have a big size. That is a problem, because it limits the amount of pieces that can be placed on top of the board simultaneously.

One simple solution is try to reduce the fiducial size, decreasing that way the size of the pieces. Unfortunately, as smaller is the size, worse is the detection and tracking on the table due to its dimensions and the hardware (more specifically, the camera used in SONA). Errors are especially noticeable in Tangram Game, as rotation is dramatically affected. Therefore, it was necessary to reduce the number of pieces used in both games:

- For Tangram, we decided to use only 3 pieces: the big triangles and the square. Although that limits notably the amount of figures that could be formed, the game is playable, thanks to the detection algorithm implemented (by construction, it's not mandatory to the 7 figures).
- For Geometry, only a set of basic shapes were used: 1 circular figure, 3 parallelogram figures, 3 triangles, 2 polygons and a non geometric form.

After different experiments, we can conclude that with the current SONA hardware the minimum size of fiducials must be about 90% of the original size in order to get the system working properly. That way, although the prototype is limited, we can perform a good proof of concept and see in action Reactive Games.

7.2.3. MongoDB C# driver incompatibility

Whilst the official MongoDB C# driver used to connect Reactive Games with the database it was supposed to work, some problems appeared (related with some functions that deal with recursion). Due to the fact that Unity uses a set/subset of the .NET language based on versions 2.0.5 and the driver is compiled with newer versions (concretely, 4.6), some incompatibilities were detected, making impossible to use it properly.

The use of the mLab Data API (REST) was discarded, taking into account that mLab recommends the of use a driver to achieve *“better performance, better security, and more functionality”*. Fortunately, this is a reported problem, so after consulting different sources the most effective solution was to adapt the driver to work with the Unity version of .NET (that was, recompile it).

With the help of some recommendations, especially after some debugging and following the advices given on [20] (see References section for further information), having an adapted version of the driver must be easy. Unfortunately, several problems appeared when compiling due to the development environment, so in the current prototype we make use of a fully functional version of the driver offered by one of the developers (*Roman Bogdziievych* [20]).

8. Cost of the project

In a development, a group of different professionals spent hours working in several areas: design, implementation, tests, linguistics... In the real world, that supposes an investment of time and money. In this section, and taking into account that all the tasks are covered by one person, we can find the cost of bringing the initial idea to a prototype. Obviously, this is not a final product, so the real cost could be incremented (creation of an adapted table for kids, purchase an advanced plan for storing more data on the backend...).

Although all the numbers presented are an estimation, it is important to separate times and monetary costs by concepts. For calculations, we have supposed an average salary by hour of 30€, taking into account that we are not (yet) professionals on video games.

Table 8.1 - Cost of the prototype

Concept	Hours	Partial Cost	Final Cost
Research	40	30 € by hour	1200 €
Design	80	30 € by hour	2400 €
Programming	250	30 € by hour	7500 €
Testing	60	30 € by hour	1800 €
Other (data insertion, translation, etc)	20	30 € by hour	600 €
Equipment used (Computer + electricity)	-	10€/month * 6 months	60 €
		Total	13560 €

9. Conclusions

9.1. What has been learned?

The master thesis, apart from a mandatory final stage, is a great opportunity for researching about interesting and innovative topics, where knowledge acquired during the master must be applied in order to develop something. During this process, additional concepts and methodologies could be learned improving our skills.

All the work done in this master thesis let me familiarise with the tangible user interface technology, something that I didn't know before. Also gave me the chance to work in games applied to a concrete area, so I had to put into practice all the learning acquired during the master, gaining more expertise in this area.

From the initial idea behind Reactive Games to the final prototype many hours researching and developing were spent. Although a lot of work have been done, time was limited and some things left undone. In spite of that, Reactive Games is implemented and prepared to be tested in a real environment.

One important lesson learned is that game design is not as easy as someone could think. There are a lot of keys to touch in order to arrive to an optimal product. A good idea it is not enough, but significant specific work must be done. Make a good design, creating an attractive and coherent story, or analysing deeply the target in order to adapt the game to it are a small part of a complex process. And to achieve optimal results team working is mandatory, so different professionals have to develop tasks in a concrete field.

But above all, I want to remark the importance of researching. Not only is a good way to enlighten oneself, but also gives the opportunity to develop new things that maybe no one could imagine ever, but could help someone in different manners.

9.2. Future of Reactive Games

Although objectives were accomplished, it is always possible to get better results. Some possible improvements for the prototype are:

- As we are designing a tangible table based system for children, this must be adapted to them. SONA table is perfect to implement a functional prototype that let us perform our proof of concept, but if this must be a real project, the dimensions of the table must be reduced in order to be fully accessible to kids.
- To make fully playable the games designed we need to use all the figures (or more, if necessary). Unfortunately, we can't reduce fiducials size, and the shapes are so big. Changing the camera

used to process the images on top of the table could help to reduce both fiducials and figures dimensions.

- The designed prototype is prepared to process single finger tracking, limiting its use to control buttons on the table as we could do with a mouse. Applying little modifications it is possible to allow multi-finger tracking, and with this, design games where multi touch could be used, in example, to paint or construct figures.
- Gamification is important in games, but could be decisive on educational ones. Customization of some parts of the games, giving rewards on completing activities, or even create some kind of story behind each game could make them more attractive to the kids.
- Including more games could be interesting in order to test different geometry aspects that maybe are not completely covered with the two included in the prototype.
- Including more configuration options, such the introduction of new tasks on Geometry Game, extend the translation into three languages for all the interface or improve helping system are tasks that can make from the prototype something close to a final product.
- In order to correct different distortions caused by little misconfigurations in the optics (mirror, camera, etc), it could be a good idea to implement a simple calibration system.
- Finally, as a further step, it could be interesting to grant access to selections or configurations (now showed on projection screen) in an external device (computer, tablet or smartphone). That way, professionals can have more control on the system hiding some actions to the children, and at the same time use projection screen only for the feedback.

9.3. Final reflection

This master thesis could be a good (humble) starting point to open new horizons on teaching, using innovative technologies that can help both professionals and kids to improve knowledge about different learning areas. This work proves that is possible to find creative and modern ways for teaching geometry, but Reactive Games could be expanded to be used in other areas. Only future can tell if all that is presented here was only a dissertation or a powerful and useful tool.

A1. Source code

A1.1. CursorInputModule

```
public class CursorInputModule : PointerInputModule {
    public GameObject cursorObject = null;
    public GameObject currentPointedObject = null;

    private Vector2 auxVec2;
    private PointerEventData pointer;
    public float holdTimeInSeconds = 0.2f;
    private float enterClickTime = 0f;
    private float exitClickTime = 0f;
    public bool clicked = false;
    private bool overButton = false;

    void Start () {
        base.Start ();
        if( cursorObject == null)
            GameObject.Destroy( gameObject );
        else
            cursorObject.transform.position = Vector3.zero;
    }

    public override void Process ()
    {
        if (clicked)
            clicked = false;

        // 3D-coordinates to Screen-coordinates
        Vector3 screenPos = Camera.main.WorldToScreenPoint( cursorObject.transform.position );

        auxVec2.x = screenPos.x;
        auxVec2.y = screenPos.y;

        // Raycasting
        if (pointer == null)
            pointer = new PointerEventData (eventSystem);

        pointer.position = auxVec2;
        pointer.delta = Vector2.zero;

        eventSystem.RaycastAll( pointer, this.m_RaycastResultCache );
        RaycastResult raycastResult = FindFirstRaycast ( this.m_RaycastResultCache );
        pointer.pointerCurrentRaycast = raycastResult;

        ProcessMove (pointer);

        if (pointer.pointerEnter != null) {
```

```

        GameObject handler = ExecuteEvents.GetEventHandler<IPointerUpHandler>
(pointer.pointerEnter);

        if (currentPointedObject != handler && !overButton) {
            currentPointedObject = handler;
            enterClickTime = Time.realtimeSinceStartup + holdTimeInSeconds;
        }

        if (currentPointedObject != null && (Time.realtimeSinceStartup >
enterClickTime)) {
            ExecuteEvents.ExecuteHierarchy (currentPointedObject, pointer,
ExecuteEvents.pointerClickHandler);
            enterClickTime = float.MaxValue;
            clicked = true;

            ExecuteEvents.ExecuteHierarchy (currentPointedObject, pointer,
ExecuteEvents.pointerUpHandler);
            ExecuteEvents.ExecuteHierarchy (currentPointedObject, pointer,
ExecuteEvents.pointerExitHandler);
            overButton = true;
            exitClickTime = Time.realtimeSinceStartup + 1.0f;
        } else if (Time.realtimeSinceStartup > exitClickTime) {
            cursorObject.transform.position = Vector3.zero;
            overButton = false;
        }

    } else {
        currentPointedObject = null;
    }
}
}

```

A1.2. Fiducial Controller

Here we can find the modified version of this script in order to allow cursor detection (for single-finger tracking). Changes are shown in red.

```
using System;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class FiducialController : MonoBehaviour
{
    public int MarkerID = 0;

    public enum RotationAxis { Forward, Back, Up, Down, Left, Right } ;

    //translation
    public bool IsPositionMapped = false;
    public bool InvertX = false;
    public bool InvertY = false;

    //double screen enabled?
    //then only half of the screen have to show projection
    public bool doubleScreen = true;
    public float doubleScreenXOverlay = 0.5f;
    //side used on projection: 0 = left, 1 = right
    public int projectionSide = 1;

    //rotation
    public bool IsRotationMapped = false;
    public bool AutoHideGO = false;
    private bool m_ControlsGUIElement = false;

    public float CameraOffset = 10;
    public RotationAxis RotateAround = RotationAxis.Back;

    public bool IsCursor = false;

    private UniducialLibrary.TuioManager m_TuioManager;
    private Camera m_MainCamera;

    //members
    private Vector2 m_ScreenPosition;
    private Vector3 m_WorldPosition;
    private Vector2 m_Direction;
    private float m_Angle;
    private float m_AngleDegrees;
    private float m_Speed;
    private float m_Acceleration;
    private float m_RotationSpeed;
    private float m_RotationAcceleration;
    private bool m_IsVisible;
```

```

public float RotationMultiplier = 1;

void Awake()
{
    this.m_TuioManager = UniducialLibrary.TuioManager.Instance;

    //uncomment next line to set port explicitly (default is 3333)
    //tuioManager.TuioPort = 7777;

    this.m_TuioManager.Connect();

    //check if the game object needs to be transformed in normalized 2d space
    if (isAttachedToGUIComponent())
    {
        Debug.LogWarning("Rotation of GUIText or GUITexture is not supported. Use a plane with a
texture instead.");
        this.m_ControlsGUIElement = true;
    }

    this.m_ScreenPosition = Vector2.zero;
    this.m_WorldPosition = Vector3.zero;
    this.m_Direction = Vector2.zero;
    this.m_Angle = 0f;
    this.m_AngleDegrees = 0;
    this.m_Speed = 0f;
    this.m_Acceleration = 0f;
    this.m_RotationSpeed = 0f;
    this.m_RotationAcceleration = 0f;
    this.m_IsVisible = true;
}

void Start()
{
    //get reference to main camera
    this.m_MainCamera = GameObject.FindGameObjectWithTag("MainCamera").GetComponent<Camera>();

    //check if the main camera exists
    if (this.m_MainCamera == null)
    {
        Debug.LogError("There is no main camera defined in your scene.");
    }

    if (doubleScreen == false)
        doubleScreenXOverlay = 0.0f;

    //debug = GameObject.Find ("debug").GetComponent<Text> ();
}

void Update()
{
    if(this.MarkerID == -1) {
        if(this.m_TuioManager.IsConnected
        && this.m_TuioManager.GetCursorCount(>0)

```



```

    {
        TUIO.TuioCursor cursor = this.m_TuioManager.GetCursor();
        this.m_ScreenPosition.x = doubleScreenXOverlay + cursor.getX()*(1-
doubleScreenXOverlay);

        this.m_ScreenPosition.y = cursor.getY();
        this.m_Acceleration = cursor.getMotionAccel();
        this.m_Direction.x = cursor.getXSpeed();
        this.m_Direction.y = cursor.getYSpeed();
        this.m_IsVisible = true;

        //set game object to visible, if it was hidden before
        ShowGameObject();

        //update transform component
        UpdateTransform();
    }
    else
    {
        //automatically hide game object when marker is not visible
        if (this.AutoHideGO)
        {
            HideGameObject();
        }

        this.m_IsVisible = false;
    }
}
else {
    if (this.m_TuioManager.IsConnected
        && this.m_TuioManager.IsMarkerAlive(this.MarkerID))
    {
        TUIO.TuioObject marker = this.m_TuioManager.GetMarker(this.MarkerID);

        //update parameters
        this.m_ScreenPosition.x = doubleScreenXOverlay + marker.getX()*(1-
doubleScreenXOverlay);
        this.m_ScreenPosition.y = marker.getY();
        this.m_Angle = marker.getAngle() * RotationMultiplier;
        this.m_AngleDegrees = marker.getAngleDegrees() * RotationMultiplier;
        this.m_Speed = marker.getMotionSpeed();
        this.m_Acceleration = marker.getMotionAccel();
        this.m_RotationSpeed = marker.getRotationSpeed() * RotationMultiplier;
        this.m_RotationAcceleration = marker.getRotationAccel();
        this.m_Direction.x = marker.getXSpeed();
        this.m_Direction.y = marker.getYSpeed();
        this.m_IsVisible = true;

        //set game object to visible, if it was hidden before
        ShowGameObject();

        //update transform component
        UpdateTransform();
    }
    else

```

```

        {
            //automatically hide game object when marker is not visible
            if (this.AutoHideGO)
            {
                HideGameObject();
            }

            this.m_IsVisible = false;
        }
    }

}

void OnApplicationQuit()
{
    if (this.m_TuioManager.IsConnected)
    {
        this.m_TuioManager.Disconnect();
    }
}

private void UpdateTransform()
{
    //position mapping
    if (this.IsPositionMapped)
    {
        //calculate world position with respect to camera view direction
        float xPos = this.m_ScreenPosition.x;
        float yPos = this.m_ScreenPosition.y;
        if (this.InvertX) xPos = 1 - xPos;
        if (this.InvertY) yPos = 1 - yPos;

        if (this.m_ControlsGUIElement)
        {
            transform.position = new Vector3(xPos, 1 - yPos, 0);
        }
        else
        {
            Vector3 position = new Vector3(xPos * Screen.width,
                (1 - yPos) * Screen.height, this.CameraOffset);
            this.m_WorldPosition = this.m_MainCamera.ScreenToWorldPoint(position);
            //worldPosition += cameraOffset * mainCamera.transform.forward;
            transform.position = this.m_WorldPosition;
        }
    }

    //rotation mapping
    if (this.IsRotationMapped)
    {
        Quaternion rotation = Quaternion.identity;

        switch (this.RotateAround)
        {
            case RotationAxis.Forward:

```

```

        rotation = Quaternion.AngleAxis(this.m_AngleDegrees, Vector3.forward);
        break;
    case RotationAxis.Back:
        rotation = Quaternion.AngleAxis(this.m_AngleDegrees, Vector3.back);
        break;
    case RotationAxis.Up:
        rotation = Quaternion.AngleAxis(this.m_AngleDegrees, Vector3.up);
        break;
    case RotationAxis.Down:
        rotation = Quaternion.AngleAxis(this.m_AngleDegrees, Vector3.down);
        break;
    case RotationAxis.Left:
        rotation = Quaternion.AngleAxis(this.m_AngleDegrees, Vector3.left);
        break;
    case RotationAxis.Right:
        rotation = Quaternion.AngleAxis(this.m_AngleDegrees, Vector3.right);
        break;
    }
    transform.localRotation = rotation;
}

private void ShowGameObject()
{
    if (this.m_ControlsGUIElement)
    {
        //show GUI components
        if (gameObject.GetComponent<GUIText>() != null && !
gameObject.GetComponent<GUIText>().enabled)
        {
            gameObject.GetComponent<GUIText>().enabled = true;
        }
        if (gameObject.GetComponent<GUITexture>() != null && !
gameObject.GetComponent<GUITexture>().enabled)
        {
            gameObject.GetComponent<GUITexture>().enabled = true;
        }
    }
    else
    {
        if (gameObject.GetComponent<Renderer>() != null && !
gameObject.GetComponent<Renderer>().enabled)
        {
            gameObject.GetComponent<Renderer>().enabled = true;
        }
    }
}

private void HideGameObject()
{
    if (this.m_ControlsGUIElement)
    {
        //hide GUI components
        if (gameObject.GetComponent<GUIText>() != null &&

```

```

gameObject.GetComponent<GUIText>().enabled)
    {
        gameObject.GetComponent<GUIText>().enabled = false;
    }
        if (gameObject.GetComponent<GUITexture>() != null &&
gameObject.GetComponent<GUITexture>().enabled)
    {
        gameObject.GetComponent<GUITexture>().enabled = false;
    }
}
else
{
    //set 3d game object to visible, if it was hidden before
        if (gameObject.GetComponent<Renderer>() != null &&
gameObject.GetComponent<Renderer>().enabled)
    {
        gameObject.GetComponent<Renderer>().enabled = false;
    }
}
}

#region Getter

public bool isAttachedToGUIComponent()
{
    return (gameObject.GetComponent<GUIText>() != null || gameObject.GetComponent<GUITexture>() !=
= null);
}
public Vector2 ScreenPosition
{
    get { return this.m_ScreenPosition; }
}
public Vector3 WorldPosition
{
    get { return this.m_WorldPosition; }
}
public Vector2 MovementDirection
{
    get { return this.m_Direction; }
}
public float Angle
{
    get { return this.m_Angle; }
}
public float AngleDegrees
{
    get { return this.m_AngleDegrees; }
}
public float Speed
{
    get { return this.m_Speed; }
}
public float Acceleration
{

```

```

        get { return this.m_Acceleration; }
    }
    public float RotationSpeed
    {
        get { return this.m_RotationSpeed; }
    }
    public float RotationAcceleration
    {
        get { return this.m_RotationAcceleration; }
    }
    public bool IsVisible
    {
        get { return this.m_IsVisible; }
    }
    #endregion
}

```

A1.2. Text To Speech & Audio Controller Scripts

Here we can find the scripts that gives the TTS functionality. Firstly, the class that interacts with the on-line service VOICE-RSS to transform text into sound. The second script (*AudioSystemController*) is used to communicate TTS with the rest of the environment, apart from grant access to sound used in games development.

```
using System;
using UnityEngine;
using System.Collections;
using System.Collections.Generic;

// Class to interact with the TTS on-the-cloud system VOICE-RSS

public class TTS {
    public static string text;
    // Speech list
    private static List<AudioClip> arrayTextsToSpeech = new List<AudioClip> ();

    // The audio clip to manage
    private static AudioClip clip;
    // Language for the speech (by default, Great Britain English)
    public static string language = "en-gb";

    public static void AddTextToSpeech (string newText) {
        text = newText;
        arrayTextsToSpeech.Add (getSpeech());
        Debug.Log (arrayTextsToSpeech.Count.ToString());
    }

    public static void AddTextArrayToSpeech(List<string> textArray) {
        foreach(string newText in textArray) {
            text = newText;
            arrayTextsToSpeech.Add (getSpeech());
        }
        Debug.Log ("Processed texts: " + arrayTextsToSpeech.Count.ToString());
    }

    public static AudioClip selectedAudioClip(int index) {
        return arrayTextsToSpeech [index];
    }

    public static AudioClip getSpeech() {
        string url = "http://api.voicerss.org/?key=xxx&src=" + WWW.EscapeURL(text) + "&hl=" +
            language + "&f=8khz_16bit_stereo&c=OGG";
        Debug.Log (url);
        WWW www = new WWW (url);
        Debug.Log (www.bytesDownloaded.ToString());
        return www.GetAudioClip(false,false,AudioType.OGGVORBIS);
    }
}
```

```

        public static AudioClip getSpeech(string message) {
            string url = "http://api.voicerss.org/?key=xxx&src=" + WWW.EscapeURL(message) + "&hl="
+
                language + "&f=8khz_16bit_stereo&c=OGG";
            Debug.Log (url);
            WWW www = new WWW (url);
            Debug.Log (www.error);
            Debug.Log (www.bytesDownloaded.ToString());
            return www.GetAudioClip(false,false,AudioType.OGGVORBIS);
        }
    }
}

```

```

using UnityEngine;
using System.Collections;
using System.Collections.Generic;

// Class to manage the audio of the whole game
public class AudioSystemController : MonoBehaviour {

    private string url;
    public AudioSource source;
    WWW www;

    public List<AudioClip> sounds;

    void Awake() {
    }

    // Use this for initialization
    void Start () {
    }

    // Update is called once per frame
    void Update () {
    }

    public void SetLanguage(string lang) {
        TTS.language = lang;
    }

    public void InitializeTextsList (List<string> texts) {
        TTS.AddTextArrayToSpeech (texts);
    }

    public void playSelectedAudioClip(int index) {
        source.clip = TTS.selectedAudioClip (index);
        playSound ();
    }

    public void playAudioClip(string message) {
        source.clip = TTS.getSpeech(message);
    }
}

```

```

        playSound ();
    }

    public void playSound() {
        if (!source.isPlaying && source.clip.loadState == AudioDataLoadState.Loaded)
            source.Play ();
    }

    public void PlayGameOverSound() {
        if (sounds.Count > 0 && sounds [3] != null) {
            source.clip = sounds [3];
            playSound ();
        }
    }

    public void PlayErrorSound() {
        if (sounds.Count > 0 && sounds [2] != null) {
            source.clip = sounds [2];
            playSound ();
        }
    }

    public void PlayCleanSound() {
        if (sounds.Count > 0 && sounds [1] != null) {
            source.clip = sounds [1];
            playSound ();
        }
    }

    public void PlayCorrectSound() {
        if (sounds.Count > 0 && sounds [0] != null) {
            source.clip = sounds [0];
            playSound ();
        }
    }
}





















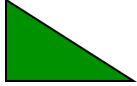



```

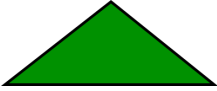







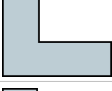







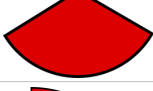
















A2. Figures and Fiducials assignment

In this Annex the association between shapes and fiducials used is shown. Additionally, the field *Basic* indicates if the figure is mandatory to perform the proof of concept.

A2.1. Geometry Game















Table A2.1 - Figures and fiducial association for Geometry Game

Name	Figure	Fiducial ID	Fiducial	Basic
Square		0		yes
Rectangle		1		yes
Parallelogram		2		yes
Trapezium		3		no
Rhombus		4		no
Quadrilateral 1		5		no
Quadrilateral 2		6		no
Equilateral Triangle		7		yes
Isosceles Triangle 1		8		no
Right Triangle 1		9		yes
Right Triangle 2		10		no
Scalene Triangle		11		yes

Name	Figure	Fiducial ID	Fiducial	Basic
Isosceles Triangle 2		12		no
Polygon 1		13		no
Pentagon		14		yes
Hexagon		15		yes
Polygon 2		16		no
Polygon 3		17		no
Circle		18		yes
Semicircle		19		no
Circular Sector 1		20		no
Circular Sector 2		21		yes
Curved Figure 1		22		no
Curved Figure 1		23		no
Non Geometric Figure 1		24		yes
Non Geometric Figure 2		25		no
Non Geometric Figure 3		26		no
Non Geometric Figure 3		27		no

A2.2. Tangram Game

Table A2.2 - Figures and fiducial association for Tangram Game

Name	Figure	Fiducial ID	Fiducial	Basic
Big Triangle 1		0		yes
Big Triangle 2		1		yes
Medium Triangle		2		no
Small Triangle 1		3		no
Small Triangle 2		4		no
Square		5		yes
Parallelogram		6		no

References

Spatial intelligence

- [1] Newcombe, N. S., & Frick, A. (2010). Early education for spatial intelligence: Why, what, and how. *Mind, Brain, and Education*, 4(3), 102-111.
- [2] Improving spatial skills in children and teens: Evidence-based activities and tips <http://www.parentingscience.com/spatial-skills.html>. Last visited: 2016-05-10

Benchmarking. Games

- [3] Kangaroo Hop. http://www.mathplayground.com/ASB_Kangaroo_Hop.html. Last visited: 2016-05-12
- [4] Four Piece Tangrams. <http://www.mathplayground.com/tangrams.html>. Last visited: 2016-05-16
- [5] Fun Shape Game for Kids. <http://www.kidsmathgamesonline.com/geometry/shapes.html>. Last visited: 2016-05-12
- [6] Montessori Geometry. <https://www.edokiacademy.com/en/app-montessori/math/montessori-geometry/>. Last visited: 2016-05-13

Benchmarking. Backend

- [7] Appcelerator Arrow. <http://www.appcelerator.com/mobile-app-development-products/mbaas-arrow/>. Last visited: 2016-05-02
- [8] Firebase. <https://firebase.google.com>. Last visited: 2016-05-02
- [9] GameSparks. <http://www.gamesparks.com>. Last visited: 2016-05-08
- [10] mLab. <https://mlab.com>. Last visited: 2016-05-08

Tangible user surface

- [11] TUIO.org. <http://www.tuio.org>. Last visited: 2016-04-19
- [12] reactTIVision 1.5.1. A toolkit for tangible multi-touch surfaces. <http://reactivision.sourceforge.net>. Last visited: 2016-04-09

- [13] Reactable. <http://mtg.upf.edu/project/reactable>. Last visited: 2016-03-15
- [14] SONA. <http://multimedia.uvic.cat/project/sona-2/>. Last visited: 2016-03-20
- [15] Reactivision pilot mixer (Copenhagen Institute of Interaction Design). <http://ciid.dk/education/portfolio/py/courses/toyview/projects/reactivision-pilot-mixer>. Last visited: 2016-04-27
- [16] Viladomat Arrò, R. (2009). Reactable Role Gaming (RRG): disseny, implementació i avaluació d'un entorn pel desenvolupament de jocs de rol per a taules interactives. <http://hdl.handle.net/10230/4739>
- [17] KIT Vision. http://webdiis.unizar.es/~jmarco/?page_id=925. Last visited: 2016-04-27

Drivers and services

- [18] Voice RSS - Free Text-to-speech (TTS) online service. <http://www.voicerss.org>. Last visited: 2016-05-23
- [19] MongoDB C# Driver. http://mongodb.github.io/mongo-csharp-driver/?jmp=docs&_ga=1.25658430.2086205881.1465744429. Last visited: 2016-05-18
- [20] Unity And MongoDB (SaaS). <http://answers.unity3d.com/questions/618708/unity-and-mongodb-saas.html>. Last visited: 2016-05-29

Games

- [21] Tangram Channel. <http://www.tangram-channel.com>. Last visited: 2016-06-10
- [22] Activity Village. Tangrams. <http://www.activityvillage.co.uk/tangrams>. Last visited: 2016-05-29
- [23] Oflazer, K. (1993). Solving Tangram puzzles: A connectionist approach. *International Journal of Intelligent Systems* 8(5):603 - 616.

Other

- [24] Unity manual: Event System. <https://docs.unity3d.com/Manual/EventSystem.html> Last visited: 2016-05-20
- [25] Unity manual: Pointer Input Module. <https://docs.unity3d.com/ScriptReference/EventSystems.PointerInputModule.html> 2016-05-23
- [26] VR Gaze Input. <http://talesfromtherift.com/vr-gaze-input/> Last visited: 2016-05-21