A collaborative and location-aware application based on augmented reality for mobile devices

Candidate: Alessandro MULLONI
Supervisor: Dr. Roberto RANON
Daniel WAGNER

Academic Year 2006/2007

Università degli Studi di Udine
Via delle Scienze, 206
33100 Udine
Italia
Reality is merely an illusion, 
albeit a very persistent one.

[Albert Einstein]

I believe in everything until it’s disproved. 
So I believe in fairies, the myths, dragons.

It all exists, even if it’s in your mind. 
Who’s to say that dreams and nightmares 
aren’t as real as the here and now?

Reality leaves a lot to the imagination.

[John Lennon]
To my mum Angela, my dad Roberto and my sister Chiara
for turning dreams into possibilities,
for supporting me when possibilities become reality.

To Maddalena,
for when this thesis started
things didn’t look so bright,
happy and clear.
Acknowledgments

Thanks to my supervisor, Daniel Wagner, for the input during the creation of a game concept and for helping me with precious hints during the whole project, especially when the source code was looking so dark. Thanks to Professor Dieter Schmalstieg for his advices and for being so friendly and open-minded. Thanks to Albert Walzer for creating such brilliant 3D models, the graphics in this project wouldn’t look this cool without his work. Thanks to Erick Méndez, Christian Pirchheim, Gerhard Schall and all the other people from ICG for treating me like a member of the research group and like a friend since my first moment in Graz.

Thanks to Roberto Ranon for his suggestions all through the development of this thesis project and for the effort he puts on promoting courses and projects on 3D computer graphics in Udine. Thanks to Luca Chittaro and all the members of HCI Lab in Udine for giving me the chance to work on interesting projects and to grow both as a student and as a researcher.

Thanks to all the good friends in Udine, in Milan, in Sheffield, in Graz and around the world. Friends will be friends, right till the end.
# Contents

<table>
<thead>
<tr>
<th>Section Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contents</td>
<td>v</td>
</tr>
<tr>
<td>List of Figures</td>
<td>vii</td>
</tr>
<tr>
<td>Introduction</td>
<td>xi</td>
</tr>
<tr>
<td><strong>1 Related work</strong></td>
<td>1</td>
</tr>
<tr>
<td>1.1 Augmented Reality</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Handheld Augmented Reality</td>
<td>3</td>
</tr>
<tr>
<td>1.3 Augmented Reality Computer Supported Cooperative Work</td>
<td>7</td>
</tr>
<tr>
<td>1.4 Augmented Reality Gaming</td>
<td>9</td>
</tr>
<tr>
<td>1.5 Handheld Geospatial AR applications</td>
<td>14</td>
</tr>
<tr>
<td><strong>2 The Studierstube ES framework</strong></td>
<td>17</td>
</tr>
<tr>
<td>2.1 Introduction</td>
<td>17</td>
</tr>
<tr>
<td>2.2 Studierstube ES</td>
<td>18</td>
</tr>
<tr>
<td>2.3 Tracking and ARToolkitPlus</td>
<td>20</td>
</tr>
<tr>
<td>2.4 Networking, database and Middleware</td>
<td>22</td>
</tr>
<tr>
<td>2.5 Graphics and StbSG</td>
<td>23</td>
</tr>
<tr>
<td>2.6 Developing applications with Studierstube ES</td>
<td>24</td>
</tr>
<tr>
<td><strong>3 Design and implementation</strong></td>
<td>27</td>
</tr>
<tr>
<td>3.1 Project goals</td>
<td>27</td>
</tr>
<tr>
<td>3.2 Game concept</td>
<td>30</td>
</tr>
<tr>
<td>3.2.1 Game setting and gameplay</td>
<td>30</td>
</tr>
<tr>
<td>3.2.2 Reality and Augmented Reality</td>
<td>31</td>
</tr>
<tr>
<td>3.2.3 Game setup</td>
<td>33</td>
</tr>
</tbody>
</table>
## Contents

3.3 Architecture of the system and hardware setup ........................................ 33
3.4 Database design .......................................................................................... 37
3.5 User Interface .............................................................................................. 42
3.6 Client-side application ................................................................................. 47
  3.6.1 Input ...................................................................................................... 48
  3.6.2 Graphics ............................................................................................... 49
  3.6.3 Database ............................................................................................... 53
  3.6.4 Gameplay ............................................................................................... 54
3.7 Server-side game controller ......................................................................... 56

4 Conclusions ..................................................................................................... 59
  4.1 Conclusions and future work ..................................................................... 59

Bibliography ....................................................................................................... 63
### List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Milgram’s Reality-Virtuality continuum [20].</td>
<td>2</td>
</tr>
<tr>
<td>1.2</td>
<td>Different impact of various AR hardware solutions: backpack-laptop with HMD, Tablet PC, PDA, Smartphone.</td>
<td>3</td>
</tr>
<tr>
<td>1.3</td>
<td>Examples of handheld devices. From left to right: Ultra Mobile PC, Personal Digital Assistant (PDA), Smartphone, portable gaming console (Gizmondo). The scale of devices with respect to each other is not the actual one.</td>
<td>4</td>
</tr>
<tr>
<td>1.4</td>
<td>Screenshots of an early handheld Augmented Reality application (AR-PDA) using a thin-client approach; here AR-PDA is used to modify the properties of an oven.</td>
<td>5</td>
</tr>
<tr>
<td>1.5</td>
<td>Example of the magic lens approach: in the AR game The Invisible Train [37] a wooden railway track (big picture) is augmented with virtual trains and switches, but these are visible only when the user looks at the railway track through the PDA device (small picture).</td>
<td>6</td>
</tr>
<tr>
<td>1.6</td>
<td>Examples of CSCW AR applications on HMD-based setups: Studierstube-based movie editing [32] (left) and the WearCom [5] conference system (right).</td>
<td>8</td>
</tr>
<tr>
<td>1.7</td>
<td>Examples of CSCW AR applications on handheld devices: TransVision [29] (left) and AR Pad [21] (right).</td>
<td>8</td>
</tr>
<tr>
<td>1.8</td>
<td>Examples of HMD-based AR games. From top to bottom: ARQuake (outdoor playing mode, the environment is real but the opponents and the gun are virtual objects), Human Pacman (the environment is real but the game objects and the overlay data are all virtual), AR Worms (the game board is fully virtual, the real environment is used only for 3D registration) and MonkeyBridge (the water, the game characters and part of the paths are virtual, but some objects like the bright white tiles are real).</td>
<td>10</td>
</tr>
<tr>
<td>1.9</td>
<td>Examples of augmented board games: public and private views in Knight-Mage [18] (left) and game board and shared display for Wizard’s Apprentice [27] (right).</td>
<td>11</td>
</tr>
</tbody>
</table>
### List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.10</td>
<td>Examples of games where AR is used only for tracking purposes and the content of the game is actually fully virtual: Mobile Maze [7] (left) and AR Tennis [14] (right).</td>
</tr>
<tr>
<td>1.11</td>
<td>Examples of games where AR is used both for tracking purposes and for real environment augmentation, but user interaction is performed only through a virtual interface: Penalty Kick [30] (left) and The Invisible Train [37] (right).</td>
</tr>
<tr>
<td>1.12</td>
<td>Examples of games where AR is used both for tracking purposes and for real environment augmentation and user interaction is performed through both virtual and real interfaces: AR Soccer [11] (left) and AR Kanji [36] (right).</td>
</tr>
<tr>
<td>1.13</td>
<td>Examples of Geospatial AR applications. From top to bottom: Reitmayr and Schmalstieg’s navigation system [28], INSTAR [23], Signpost [38] and Vidente [31].</td>
</tr>
<tr>
<td>1.14</td>
<td>Map augmentation: Schöning’s work [16] (left) and (b) Vidente [31] running on a miniature model (right).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Component-based architecture of the framework developed by ICG at TU Graz: the Studierstube ES framework and the libraries that form the hardware abstraction layer.</td>
</tr>
<tr>
<td>2.2</td>
<td>Example of the binary-encoded fiducial markers used by ARToolkitPlus. The shown markers are those having an encoded ID between 0 and 9.</td>
</tr>
<tr>
<td>2.3</td>
<td>ARToolkitPlus pipeline: a device is aimed at a given square marker; a picture is taken from the camera of the device; the image is ported to black-and-white format; a marker is identified and a coordinate system based on the centre and orientation of the marker is computed; 3D geometry is rendered on top of the marker and the camera image using the computed coordinate system; the final image is shown on the display of the device which therefore acts now as a see-through augmenting display.</td>
</tr>
<tr>
<td>2.4</td>
<td>Example of an AR application that can be obtained by only configuring Studierstube ES through XML files (without writing source code).</td>
</tr>
<tr>
<td>3.1</td>
<td>Gizmondo handheld console.</td>
</tr>
</tbody>
</table>
3.2 Example of a game setup where each pasture location is represented by a circle and each stable is drawn with a small icon. Walls are drawn in black while blue lines represent connections between different locations. 33

3.3 Architecture of the system implemented for the present thesis project. 34

3.4 Entity-Relationship diagram for the database. 38

3.5 Implementation of the first user interface design; the screenshot shows the application view while looking at a marker. 3D arrows are pointing to the connected markers. An overview map showing an orthographic projection of the whole game environment can be visualised on-demand. 42

3.6 First application to test the user interface. 3D arrows are used to augment the real environment and a helper map is visualised as a semi-transparent overlay. The currently visible marker is stressed on the map with the aid of a special icon, and the map orientation is kept coherent with the position and the view direction of the user. 43

3.7 Second application to test the user interface. 3D arrows and marker IDs are used to augment the real environment and a helper map is visualised as a semi-transparent overlay. The currently visible marker and its connection are colour-coded and the map orientation is kept coherent with the position and the view direction of the user. Marker IDs are shown both on the overview map and in the AR view. 45

3.8 Colour-coding to identify the team and the objects locked to the team. All the overlays assume the colour of the team and team colours are also used to identify the objects which are currently locked and what team locked them. In these images the player is playing for the blue team. In the right image the UFO and a cow are locked to the blue team while two cows are locked to the red team. 46

3.9 Mapping of the state of arrows on their graphical aspect. Dark-coloured objects are not valid selections while those with a bright tint are valid ones. The currently selected arrow assumes a red colour, other arrows assume a grey colour. The validity of arrows is also mapped on the 3D models used. From left to right: valid arrow non selected, valid arrow selected, non valid arrow non selected, non valid arrow selected. 46
3.10 Diagram representing all the classes designed and implemented for the present thesis project, grouped by their functionality. On the bottom the parts of the Studierstube ES framework which are exploited by each class are also shown. ............................................. 49

3.11 Example of a 3D object rendered with the normal OpenGL shading (left) compared to the same object rendered with cartoon shading (right). .... 51

3.12 Example of a 1D texture used for toon shading. Leftmost values have texture coordinates close to 0 and rightmost values have texture coordinates close to 1. This texture converts smoothly-changing texture coordinates into ranges having a constant colour. ............................................. 52

3.13 2D overlays used during a game session: a text box (top) tells the name of the player using the device, two sliders (side) are used during mini-games, another text box (bottom) tells the game action currently selected by the player. Other text boxes (centre) are popup windows with a timeout and give extra information on the progress of the game. .................. 52

3.14 Overview map shown as a 2D overlay. ............................................. 53

3.15 Menu to select a player shown by the PlayerSelection class. ............... 55

3.16 Screenshot of the server-side game controller. .......................... 57
Introduction

The project presented in this thesis was carried out at the Institute for Computer Graphics and Vision of the Technische Universität Graz\(^1\) inside the research group directed by Professor Dieter Schmalstieg\(^2\) and under the supervision of Daniel Wagner\(^3\). Professor Schmalstieg’s research group is currently developing and maintaining a framework for multi-user Augmented Reality applications running on mobile devices called Studierstube ES; this framework is specifically targeting multi-player handheld Augmented Reality games.

Augmented Reality applications augment the real environment with virtual embedded information that appears to be consistently registered in 3D with real spatial locations; Augmented Reality has an advantage over Virtual Reality because the real environment contains a richness of information and user-interaction techniques that cannot yet be reached by synthetic environments. Several Augmented Reality systems have been developed but most of them are based on bulky, expensive and intrusive hardware; a cheaper, less intrusive and more portable hardware solution is given by modern handheld devices that are gaining in performance every year (thanks mostly to dedicated CPUs and embedded peripherals) and are now effectively an off-the-shelf compact solution for Augmented Reality applications.

The purpose of the present thesis project is to design and implement a multi-user collaborative Augmented Reality game which is based on the Studierstube ES framework and that exploits the capability of users to move inside the environment. If the gameplay is compelling game players tend to be more forgiving than users of professional application when hardware or software issues occur; games are therefore an ideal class of applications for research projects focused on developing new technological solutions. The overall goal of this project is to develop a case-study

---

\(^1\)Institute for Computer Graphics and Vision
Technische Universität Graz
Inffeldgasse 16, A-8010 Graz, Austria
http://www.icg.tu-graz.ac.at/

\(^2\)http://www.icg.tu-graz.ac.at/Members/schmalstieg/

\(^3\)http://www.icg.tu-graz.ac.at/Members/daniel
application to prove the feasibility of merging Computer Supported Cooperative Work, location-awareness and Augmented Reality in a system based on standalone handheld platforms and exploiting user mobility in an environment. The proposed solution can be partially or totally reused for non-entertainment applications: the system developed for user orientation in the physical game environment could be used for indoor navigation systems, the whole system could be used to coordinate groups of workers in a cooperative context and a large environment (e.g. workers on a building site).

The development of the present thesis project required hardware and software decisions that brought to the design and the implementation of a client-server architecture based on Bluetooth connections. Handheld game consoles with a rich set of built-in peripherals (including Bluetooth connection, a VGA camera and an nVidia 3D accelerator) have been chosen as client devices for the present project while a desktop computer with a Bluetooth adapter has been adopted as the central server. A database has been designed with the purpose of holding the state of the whole system in the central server to share it with all client devices through Bluetooth. A custom application for client devices has been implemented using the Studierstube ES framework; this application grants a set of custom features that are not present in the original framework: such features include easier handling of game data through custom data structures, graphics methods to visualise the state of the game, 3D geometry processing for special graphical effects and handling player-triggered changes in the state of the game. The user interface of the client application has been designed and refined through two informal sessions of user testing. A further application has been implemented to handle on the server some game events that cannot be assigned to a client device.

The present thesis document is structured as follows: Chapter 1 describes the recent work carried out in Augmented Reality, focusing mostly on projects that exploit handheld devices; Chapter 2 is a small introduction to the Studierstube ES framework which has been used for the present project; Chapter 3 presents the goals for this project, how they were transferred into a concept and how a hardware and software solution for such concept has been designed and implemented; Chapter 4 draws some conclusions about the present project and describes possible future directions of this work.
1 Related work

1.1 Augmented Reality

Augmented Reality (AR) is a branch of computer science research and Virtual Reality (VR) which deals with augmenting the real world with computer-generated information. VR and AR are both focused on providing information to users inside an immersive 3D environment; while VR totally immerses the user inside a fully virtual environment, AR exploits the already existing user immersion inside the real environment and provides additional computer-generated information to obtain an augmented environment. The real environment around us is so rich in both provided information and interaction techniques available that it is difficult replicate it in a computer; synthetic environments are therefore still very simplistic if compared to real ones. An advantage of AR over VR is therefore the richness of the exploited environment; one of the major disadvantages is the complexity of merging together and consistently align these two separate environments: this problem is called 3D registration and will be discussed later.\[1\]

There is no unique definition of AR, although some definitions have appeared in literature. Milgram defines AR on the basis of the so-called Reality-Virtuality continuum \[20\], shown in Figure 1.1. A virtual environment is regarded to be a fully synthetic environment in which users are completely immersed; the real environment is considered to be at its opposite, composed only by real objects and inevitably bound to (or limited by) the laws of physics. These two environments are surely strong antitheses because of their contrasting properties, but such differences can also be considered as marking the two extremes of Milgram’s Reality-Virtuality continuum.

\[^{1}\text{http://www.se.rit.edu/~jrv/research/ar/introduction.html}\]
1. Related work

Mixed Reality is located all along Milgram’s continuum and represents all systems that exploit elements from both the real and the virtual environment at the same time. As seen in Figure 1.1 within Mixed Reality one can distinguish between Augmented Reality and Augmented Virtuality, depending on which is the primary environment and which is the secondary one used only as a supplement. When a system is close to the central part of the continuum it becomes more arbitrary whether to define it as Augmented Reality or Augmented Virtuality because it is less clear which environment is preponderant over the other. AR is located closer to the real environment than to the virtual environment on Milgram’s continuum; AR can therefore be seen as an extended version of the real environment, one supplemented by virtual objects.

Ronald Azuma identifies three fundamental characteristics for AR and gives his own definition of an AR system without binding it to specific hardware solutions:

- an AR system must combine real and virtual;
- an AR system must be interactive in real-time;
- an AR system must be registered in 3D.

3D registration is one of the basic problems in AR: virtual 3D objects and real objects must appear consistently aligned with each other. Several techniques can be used to track the position of AR devices and to perform registration (from vision-based pattern recognition to sensor-based tracking) providing different levels of accuracy and usually requiring more expensive hardware as the adopted solution becomes more professional. Registration errors in AR can be easier to detect than in a virtual environment because visual-visual conflicts (when the real object and a virtual one registered to it – both usually visible in an AR system – are not properly aligned) can occur and they are highly noticeable. High accuracy in the registration process...
is especially demanded by certain application (e.g. applications to support surgery) and the real-time interactivity requirement makes registration even more difficult because it must occur within a small time-frame. [2, 3]

1.2 Handheld Augmented Reality

Many early and current AR hardware setups are based on static desktop computers with stationary cameras or on Head Mounted Displays (HMDs) and backpack laptops. While such hardware setups provide high performance and – in the backpack laptop case – usually hands-free operation, these solutions also present serious drawbacks (e.g. high costs, low social appeal and limitations to users’ dexterity) which prevent them from targeting a wide audience of non-technical users. An example of the impact of different AR hardware solutions can be seen in Figure 1.2.

![Figure 1.2: Different impact of various AR hardware solutions: backpack-laptop with HMD, Tablet PC, PDA, Smartphone.](image)

Drawbacks of desktop and backpack-based AR systems are especially evident when AR is exploited in mobile computing; a mobile AR system can assist users in-place whenever needed and is therefore a particularly compelling research area. When thinking for mobility, devices falling inside the handheld class (e.g. Ultra Mobile PCs, PDAs, smartphones and portable gaming consoles, all shown in Figure 1.3) appear to be valid hardware solutions for AR applications. Since the present project
has been designed and developed for a handheld platform, this chapter will intentionally focus mainly on related work based on handheld setups.

Figure 1.3: Examples of handheld devices. From left to right: Ultra Mobile PC, Personal Digital Assistant (PDA), Smartphone, portable gaming console (Gizmondo). The scale of devices with respect to each other is not the actual one.

Until a few years ago AR projects targeting handheld devices used them just as front-end displays, while all computations were performed remotely and transmitted back to the devices (exploiting wireless network infrastructures and connections). The AR-PDA project [10] (shown in Figure 1.4) and the Batportal system [24] are examples of such thin-client approach, where a front-end client with basic I/O functionalities runs on PDA devices while all remaining processing is performed remotely on a desktop computer and data is transmitted from and to the PDA through a wireless connection. A similar architecture has also been adopted for the AR-Phone project [1] running on mobile phones.

Handheld devices have recently advanced in computing power and also in 3D graphics processing, mostly thanks to the introduction of embedded Graphics Processing Units (GPUs); furthermore the latest generations of handheld devices were released with built-in wireless capabilities and cameras (also because of commercial pressure from communication companies). Handheld CPUs – because of power consumption limits – are not expected to improve much in the next years; current research focuses instead on embedding specialised processors (like GPUs, video encoders and decoders, vector floating-point processors and programmable processors (FPGAs) into devices, to improve performance for certain classes of applications while maintaining power drain within acceptable limits (e.g. by disabling some processors

\[\text{http://www.arm.com/products/CPUs/families/ARM11Family.html}\]
\[\text{http://www.arm.com/products/CPUs/ARM_Cortex-M1.html}\]
1.2. Handheld Augmented Reality

Figure 1.4: Screenshots of an early handheld Augmented Reality application (AR-PDA) using a thin-client approach; here AR-PDA is used to modify the properties of an oven.

when they are not used); an example of an embedded processing unit with these characteristics is Texas Instrument’s Omap 2420.

While early thin-client approaches were necessary because of the low computational power available on old handheld devices, hardware advances made it possible to use handheld devices as standalone AR platforms (therefore with video processing, rendering and any other computation directly performed locally on the hardware of handheld devices). Recent handheld devices are therefore effectively compact and self-contained off-the-shelf AR hardware setups. Having standalone AR systems is very important for the scalability of applications because when the computational burden is performed locally on each single client device then the overhead on central servers is considerably reduced, or even central servers can be removed because clients do not need to communicate with each other or because they can communicate peer-to-peer.

Together with challenging mobility capabilities and new hardware potentials, handheld devices have also some inherent drawbacks. Although new devices are released with embedded hardware for 3D graphics acceleration, low power consumption requirements make it impossible for them to compete with the graphical quality achieved by desktop systems. Furthermore handheld devices are usually not equipped with a Floating-point Processing Unit (FPU) and are therefore only capa-

http://focus.ti.com/pdfs/wwbu/TI OMAP2420.pdf
ble of performing fixed-point computations in hardware while floating-point computations are emulated in software (this performs up to 50 times slower if compared to a hardware floating-point implementation): this makes it impossible to use algorithms strongly based on floating-point computations. Wireless network connections are still not optimal on some devices (e.g. those equipped only with Bluetooth, because of its short range and limited bandwidth).

Figure 1.5: Example of the magic lens approach: in the AR game The Invisible Train [37] a wooden railway track (big picture) is augmented with virtual trains and switches, but these are visible only when the user looks at the railway track through the PDA device (small picture).

Small display size and narrow field of view constrain the type of user interaction techniques that can be adopted. The most used user interface technique in handheld AR is the magic lens, a metaphor borrowed from works on User Interfaces and 3D graphics [4, 35]. When adopting the magic lens approach the handheld device is employed as a see-through augmenting lens and aiming it at a portion of the real environment will result in an augmented view of that part of the world. One major characteristic of this approach is that many physical movements are required if the user wants to examine a huge AR environment, because only a small portion of the environment can be seen at a single time; this can be a drawback (e.g. in an AR application which requires the user to have a global overview of the whole AR environment) but also an advantage (e.g. if the exploration of the environment is
strongly integrated in the gameplay of an AR game). An example of the magic lens approach can be seen in Figure 1.5.

One disadvantage when developing for handheld devices is a general lack of standard software interfaces, although some recent devices have abandoned proprietary solutions in favour of more standardised operating systems (e.g. Microsoft Windows, Symbian, Linux) and development libraries (e.g. Windows Mobile API\(^5\), OpenGL ES, OpenVG, OpenSL ES, OpenMAX\(^6\)). A framework to help in developing AR applications for handheld devices is presented in [36]; to the best of our knowledge this is also the first example of a standalone AR system running on a handheld device. Subsequent developments on the same framework are presented in [37]. This framework is also used for the present thesis project and will be further described in Chapter 2.

Some recent works pioneered the use of mobile phones as standalone AR platforms. Henrysson recently ported the vision-based tracking library ARToolKit [17] to Symbian smartphones [15], a further step towards the creation of reusable software libraries for developing handheld AR applications. Möhring developed an alternative custom tracking library [22] also based on computer vision techniques.

1.3 **Augmented Reality Computer Supported Cooperative Work**

Computer Supported Cooperative Work (CSCW) systems based on AR merge the advantages of virtual visualisation (adding virtual information to the real environment) and the potential of real (or video-based) face-to-face communication. The key features of a collaborative AR environment can be identified in **virtuality** (objects that are not available or not existent can still be examined), **augmentation** (real objects can be augmented with virtual spatially-aligned information), **cooperation** (normal user interaction can still be exploited), **independence** (each user can control an independent point of view) and **individuality** (data can be shared but it can also be presented in different ways to different users). [33]

Works on this field often adopt approaches based on desktop computers or HMDs and backpack laptops while there is little work exploiting handheld devices. Shared

\(^5\)http://msdn.microsoft.com/mobility
\(^6\)http://www.khronos.org/
Space \[6\] and Studierstube \[32\] pioneered the use of AR for CSCW applications. Studierstube has been used to develop the prototype for a collaborative storyboard design application for movies, running on a setup composed by various workstations and HMDs, and for many other collaborative AR projects; a backpack-based setup with HMD-based visualization is also available. Shared Space has been used for an AR conferencing system called WearCom \[5\]. Some pictures are shown in Figure 1.6.

Another pioneering project – but a handheld-based one – is TransVision \[29\] which employs portable see-through devices to help collaborative design in an AR environment; a sensor-based tracking system performs the registration of virtual 3D objects for the collaborative design process, and these objects are superimposed over the

Figure 1.6: Examples of CSCW AR applications on HMD-based setups: Studierstube-based movie editing \[32\] (left) and the WearCom \[5\] conference system (right).

Figure 1.7: Examples of CSCW AR applications on handheld devices: TransVision \[29\] (left) and AR Pad \[21\] (right).
real environment. AR Pad [21] is a handheld display equipped with a joystick and a camera; it has been exploited as a hardware interface for face-to-face collaborative AR applications.

A particular group of collaborative AR applications are AR games, both in their collaborative and competitive forms. This research area is particularly interesting because players of games tend to forgive small software and hardware issues when these issues do not hinder the game experience; this is usually not valid for professional applications (where application costs tend to levitate and end-users therefore expect the system to be free of flaws), or when a single issue can cause serious damage (e.g. in an AR-based surgical application). This can be probably seen as a justification for the proliferation of AR games, also on handheld devices.

1.4 Augmented Reality Gaming

Handheld AR games can be classified considering three main factors:

- **number of users**: multiplayer games appear to be the most popular, because they do not require developing AI routines for the virtual opponents and at the same time exploit the human-to-human communication already present in the real environment;

- **physical scenario**: this can range from a single object or a coloured patch used only for tracking purposes (therefore without any active role in the gameplay), to static tabletop boards with support for registration, to a full real environment with movable objects and properly set up to complement the AR contents of the gameplay;

- **game scenario**: this is a factor in common also with non-AR games and refers to the type of game (action, turn-based, adventure): since of technical limitations on handheld AR systems slow-paced game mechanics are preferred (where gameplay is developed through storytelling more than through real-time action).

Several AR games with HMD-based setups have been developed in the past years. ARQuake [34] is an AR implementation of the famous game Quake[^1] where the

[^1]: http://www.idsoftware.com/games/quake/quake/
1. Related work

Figure 1.8: Examples of HMD-based AR games. From top to bottom: ARQuake (outdoor playing mode, the environment is real but the opponents and the gun are virtual objects), Human Pacman (the environment is real but the game objects and the overlay data are all virtual), AR Worms (the game board is fully virtual, the real environment is used only for 3D registration) and MonkeyBridge (the water, the game characters and part of the paths are virtual, but some objects like the bright white tiles are real).

original virtual maps are replaced by the real environment while enemies and other game objects are still virtual and superimposed on the real environment; ARQuake can be played both outdoors and indoors but it’s a single-player game (therefore there is no social interaction). Human Pacman \[^8\] is an AR implementation of the well-known game PacMan\[^8\] where PacMan and the ghosts are played by two separate human teams; the game space is constituted by the real environment while special PacMan objects are virtual. Monkeybridge is a collaborative tabletop AR game which partially draws its inspiration from the game Lemmings\[^9\] like in the

\[^8\]http://en.wikipedia.org/wiki/Pac-Man

\[^9\]http://en.wikipedia.org/wiki/Lemmings_(video_game)
Lemmings game, the virtual game characters inexorably walk through a path that is floating on water: the goal of the game is to build this path with both real and virtual elements before the game character walks into the water and drowns. AR Worms \cite{25} is an AR implementation of the game Worms\cite{10}; it uses an HMD-based architecture to visualize a fully virtual game board with virtual characters on it; players can only interact with the board through an adapted GUI (a gamepad or a PDA), therefore no physical interaction with the game board is possible. AR Tankwar \cite{26} is a real-time strategic board game using an architecture very similar to the one used in AR Worms. Some pictures of these games can be seen in Figure \ref{fig:ARGames}.

A very peculiar branch of AR gaming concerns the augmentation of tabletop board games; these are not ordinary AR games because 3D registration is not really performed: the whole board is usually augmented (e.g. with a projector or a large display) together with the game experience, but there are no 3D virtual objects superimposed on the real environment. False Prophets \cite{19} uses a custom game board augmented with IR sensors and emitters: players are encouraged to play concurrently while exploiting face-to-face communication; game information is both public (on the game board) and private (shown to each player on handheld computers). CATS \cite{18} is a software framework for AR games and has been used to develop some games called KnightMage, Candyland and an AR adaptation of Monopoly; all these games support physical interaction while the game state is visualised through

\footnote{http://worms.team17.com/}
custom game boards and physical objects. Wizard’s Apprentice [27] is an AR board game where interaction is performed physically through a custom game board; game information is completely public and it is displayed through a shared monitor. The Eye of Judgement[11] is a card-based battle game where each card is augmented with the corresponding 3D avatar; it has recently been released for Sony Playstation 3. Some pictures of these AR board games can be seen in Figure 1.9.

There’s a huge number of recent AR games based on handheld setups. The following paragraphs will present and classify them by the type of physical scenario adopted; the number of players and the game scenario are also presented for each of them.

Figure 1.10: Examples of games where AR is used only for tracking purposes and the content of the game is actually fully virtual: Mobile Maze [7] (left) and AR Tennis [14] (right).

A first group of handheld AR games is those who use the real world just for synchronisation purposes (Figure 1.10). These games usually exploit a recognizable object or a coloured patch inside the real world and use it to track handheld position and movements and present on their display only virtual content: this is a weaker version of the traditional AR paradigm because the gameplay is augmented (by adding real world movements) but not the real environment. Mobile Maze [7] is a purely virtual single-player maze game where the ball must be moved through tilting movements of

the devices therefore with an interaction technique borrowed from real world experience. Marble Revolution\[^{12}\] is another single-player maze game with an interaction technique based on tilting. Symball \[^{13}\] is a two-player competitive table tennis game for Symbian devices; the handheld devices are not registered to each other but they have their movements tracked to control the tilting of the racket. AR Tennis \[^{14}\] is another competitive two-player tennis game but here the game space is consistently registered between the two devices and the devices themselves can be tracked and shown with visual feedback on the opponent’s device.

![Fig. 1.11](http://www.bit-side.com/)

Other handheld AR games present to users an augmented view of the real environment but the interaction with the environment is limited to a virtual interface on the device while interaction with real objects is not relevant for the gameplay (Figure 1.11). Penalty Kick \[^{30}\] is a simple single-player game where the aim of the player is to shoot a ball into a goal printed on a product package and registered through a marker. Impera Visco \[^{30}\] is a competitive multi-player turn-based board game where marker-based tiles are augmented with virtual data and enemies. The Invisible Train \[^{37}\] is a multi-player AR game where players control virtual trains running on a real miniature railroad track; players collaborate by modifying train

\[^{12}\]http://www.bit-side.com/
speeds and operating track switches through an AR user interface running on PDAs. Virtuoso [41] is a collaborative multi-player educational game where AR is used to provide players with explanations about artworks so that they gain the knowledge necessary to sort all virtual artworks on a real timeline.

Figure 1.12: Examples of games where AR is used both for tracking purposes and for real environment augmentation and user interaction is performed through both virtual and real interfaces: AR Soccer [11] (left) and AR Kanji [36] (right).

Finally some games do not only augment the real environment graphically but also exploit direct user interaction with real objects in their gameplay (Figure 1.12). AR Soccer [11] is a single-player game where the aim is to kick a virtual ball into a virtual goal using the real foot: the edge of the foot is detected to perform proper speed and direction calculations. AR Kanji [36] is a competitive two-player memory card game where players are asked to recognise a certain symbol on the back of a playing card: by turning a selected card players can obtain an AR confirmation of the correctness of their answer. Museum Augmented Reality Quest (MARQ) [13] is a collaborative multi-player educational game whose game space is a whole museum; players are required to complete quests if they want to progress in the gameplay, and some quests must be solved through the interaction with real objects located in different areas of the museum.

1.5 Handheld Geospatial AR applications

Geospatial AR is a branch of AR where the informative potential of AR is exploited directly in-place (thanks to the mobility of the adopted devices) to superimpose

relevant geo-referenced information on the corresponding area of the real world.

Several Geospatial AR systems have been developed to help indoor and outdoor navigation. Reitmayr and Schmalstieg [28] present an HMD-based system that exploits AR for user navigation and information browsing in an urban environment; navigation can also be collaborative when more than one person uses the system at the same time. The INSTAR system [23] is based on handheld devices and uses various sensors to superimpose navigation AR cues on the real environment. Signpost [38] guides users inside an unknown building through AR navigation hints. Vidente [31] is an application for visualizing in real-time subsurface features: a handheld device is used to visualize hidden subsurface elements like pipes and wires through AR.

Figure 1.13: Examples of Geospatial AR applications. From top to bottom: Reitmayr and Schmalstieg's navigation system [28], INSTAR [23], Signpost [38] and Vidente [31].
A particular branch of Geospatial AR is focused on augmenting maps to ease the process of content browsing. Schöning et al. [16] present a work that combines the advantages of a large scale paper-based map with the informative capabilities of AR: a magic lens approach is used on handheld devices to augment the map with interactive information. Vidente [31] has also been implemented for augmentation of urban miniature models.

Figure 1.14: Map augmentation: Schöning’s work [16] (left) and (b) Vidente [31] running on a miniature model (right).
2

The Studierstube ES framework

2.1 Introduction

Currently available handheld devices are heterogeneous in both the operating systems adopted and the libraries available for application development. Standardised operating systems (e.g. Microsoft Windows, Symbian, Linux) and development libraries (e.g. Windows Mobile API, OpenGL ES, OpenVG, OpenSL ES, OpenMAX) are now emerging, but they are still not widespread enough; furthermore, libraries usually offer just a low-level interface to developers.

There is a general need for tools to help on high-level development of handheld AR applications. When considering the development of multi-user applications, fundamental requirements for such tools are decent client-side performance (even on target handheld devices with very limited resources) and server-side robustness (especially because of often unstable network connections). Tools are also needed for rapid prototyping and authoring because there are still no safe and solid prototyping paradigms for handheld AR applications. Considering the good levels of computing power available on recent handheld devices the possibility to run standalone client-side AR applications has now become an important chance for having very scalable applications.

A component-based framework for AR applications targeting mobile devices has been developed by the Institute for Computer Graphics and Vision at the Technische Universität Graz. The component-based architecture of this framework is shown in Figure 2.1. This framework has been used for the present project and the aim of this chapter is to give a brief overview on the capabilities of the framework and of its main components before the actual thesis project is presented in the following
chapters.

2.2 Studierstube ES

The core of the framework is Studierstube ES (Embedded Subset). After an attempt to make a lightweight port for handheld devices of the original Studierstube framework used for desktop and HMD-based setups and given the bad performance of such porting, the development of STUDIERSTUBE ES started from scratch during 2006. The earliest Studierstube ES is presented in while its second evolution is described in; the current version is anyway quite different from the first ones: a big part of Studierstube ES has been rewritten and is now legacy-free to perform better on handheld devices with limited hardware.

Studierstube ES is a cross-platform library so it can target devices running various operating systems; currently supported operating systems are all versions of Microsoft Windows Mobile (and Windows CE), Symbian and Linux; Studierstube ES

Figure 2.1: Component-based architecture of the framework developed by ICG at TU Graz: the Studierstube ES framework and the libraries that form the hardware abstraction layer.
can also run on Microsoft Windows XP to allow for quicker (and high-performing) prototyping and for easier client-side debugging. The main target platform is at the moment Windows Mobile, therefore Windows-based devices are the best supported ones.

An important feature of Studierstube ES is the capability to run standalone applications on handheld devices: applications using the Studierstube ES library are able to run fully on target devices without relying on remote servers, thus reducing the server-side computing burden and allowing for high scalability of developed applications. A persistent connection with a central server is anyway maintained by Studierstube ES itself when available: the role of a central server is crucial for sharing the global state of an application within all clients and for the robustness of the whole developed system against client-side failures.

Studierstube ES is able to run multiple applications concurrently but in a single thread (Studierstube ES is single-threaded); this is at the moment not a limitation because current handheld devices do not have multi-thread or multi-core hardware. All Studierstube ES applications are kept in dynamic libraries (DLLs) and there is no need for the framework to know anything about each application before run-time. Applications are dynamically loaded and unloaded on-demand helping for an optimal use of the scarce memory usually available on handheld devices.

As seen in Figure 2.1, Studierstube ES runs on top of a hardware interface layer and offers all fundamental functionalities needed in a multi-user handheld AR application:

- scenegraph handling to define and render 3D objects; this is based on OpenGL ES (a library offering 3D rendering with support for hardware acceleration);
- position tracking based on the ARToolkitPlus library;
- tools for a rapid creation of 2D Graphical User Interfaces (GUIs);
- audio and video playback to insert multimedia contents in the applications;
- handling of network connections with a remote database, using the Middleware client library.

All functionalities exploited in the present project will be described in the next sections: tracking functionalities in Section 2.3, database and networking functionalities
in Section 2.4 and all graphics functionalities (for both 3D scene graph handling and 2D GUIs) in Section 2.5; no multimedia content is used in the present application therefore multimedia functionalities are not discussed here.

2.3 Tracking and ARToolkitPlus

Handheld devices are often compact and closed platforms that do not expose hardware interfaces which would be necessary to connect external sensors typically used for tracking purposes; furthermore the addition of external sensors itself goes against the common desire of having off-the-shelf compact and robust AR platforms. Finally a big part of modern handheld devices is released with a built-in camera. All these considerations inevitably point towards vision-based tracking solutions.

ARToolkit [17] is a common tracking library based on computer vision techniques. The ARToolkitPlus project [39] first started with the goal of porting the code of ARToolkit to handheld devices [38] but was later subject to a revision and evolved into the current library which shares almost no code with the original desktop-based library; the peculiarities and the limits of handheld devices guided this revision process. Mobile devices typically lack for hardware floating-point processing, which is usually emulated in software with obvious performance drop-downs: this is just changing right now and the first FPU-equipped devices are emerging (e.g. Texas Instruments OMAP 3430\textsuperscript{1}) but floating-point hardware on handheld devices is still very rare; the parts of ARToolkit which were based on floating-point computations have been therefore rewritten and are now using fixed-point. Memory is a scarce resource on handheld devices, together with usually very limited memory bandwidth: this arose the necessity to support native pixel-formats from device cameras, without adopting computationally expensive pixel-format conversions. Another limitation given by scarce memory bandwidth is that video streams are often low-resolution although modern cameras can reach good resolutions when taking still pictures.

Tracking in ARToolkit is based on square patterns called fiducial markers; the markers used in ARToolkit are drawn using custom black-and-white image templates and are recognised using computer-vision techniques exploiting pattern-matching. ARToolkitPlus uses a similar kind of marker but the algorithm based on templates (and requiring computationally expensive pattern-matching algorithms) has been

\textsuperscript{1}http://focus.ti.com/pdfs/wtbu/ti_omap3family.pdf
2.3. Tracking and ARToolkitPlus

replaced by an algorithm based on binary-encoded fiducial markers; these binary-encoded markers represent up to 4096 unique IDs as $6 \times 6$ patterns which have embedded error correction and are robust to $90^\circ$ rotations (a common source of problems when using square markers). Some of the markers used by ARToolkitPlus are shown in Figure 2.2 while the whole marker-based tracking pipeline of ARToolkitPlus is presented in Figure 2.3.

Figure 2.2: Example of the binary-encoded fiducial markers used by ARToolkitPlus. The shown markers are those having an encoded ID between 0 and 9.

Figure 2.3: ARToolkitPlus pipeline: a device is aimed at a given square marker; a picture is taken from the camera of the device; the image is ported to black-and-white format; a marker is identified and a coordinate system based on the centre and orientation of the marker is computed; 3D geometry is rendered on top of the marker and the camera image using the computed coordinate system; the final image is shown on the display of the device which therefore acts now as a see-through augmenting display.

ARToolkitPlus includes further robustness capabilities and performance-driven optimisations. A lightweight heuristic based on values taken from the previous frame is used to compensate for variations of lighting conditions in the environment; this is fundamental given the mobility characteristics of handheld devices and therefore the rapidly changing illumination. When images taken from device cameras contain a radial luminance fallout (common with low-quality built-in cameras) ARToolkitPlus
gives the possibility to correct the problem using a radial correction value. Finally, utilities have been developed to ease the process of camera calibration.

2.4 Networking, database and Muddleware

When developing handheld AR applications there is a necessity for a networking system to support communication and to grant at the same time enough robustness to compensate for the lack of reliability in connections (which are typically loosely coupled and unstable); when dealing with multi-user handheld AR applications this robustness translates also into an application state fully held in a shared and consistent central repository. Exploiting different heterogeneous devices is common in handheld AR: this implies that the networking system must be able to run on different hardware and software platforms and to perform decently even on the less powerful device.

Muddleware [40] is a communication platform that targets handheld devices in a mobility contest and has a strong focus on supporting multi-player AR games. It slightly draws its inspiration from the concept of Tuplespaces [12], an associative memory which is represented as collection of tuples and is used for storing values and executable code. Muddleware differs from Tuplespaces because it is based on an XML Document Object Model which allows recursive definitions of tuples; inside Muddleware all XML processing is performed exploiting a modified version of TinyXML2. Queries to the database are performed using XPath3, a declarative language that does not contain advanced programming constructs (e.g. looping) and is therefore easily and quickly parsed and executed. Synchronous and asynchronous callbacks (watchdogs) can be specified so that both client-side and server-side modifications incurring on selected data are consistently and automatically visible by any other Muddleware client which is using the same server.

Developers have various alternatives to access a Muddleware database. C++ and Java methods are available for low-level access: this assures an immediate execution of the specified queries but normally requires many lines of code to perform the desired actions. C++ shared-memory data structures are available and can be declared and handled like typical C++ data structures: these structures exploit a

\footnote{2http://sourceforge.net/projects/tinyxml}

\footnote{3http://www.w3.org/TR/xpath}
set of macros to transparently and automatically synchronise data from and to the central database on the Muddleware server. A more advanced database interaction paradigm is also available: Muddleware scripts consent to express data-driven queries with an XML-based scripting language and to register them on the server for a quicker execution. Finally all transitions can be controlled by an external (typically server-side) Muddleware controller which is defined as a Finite State Machine with data-driven transitions.

Muddleware traditionally exploited a hardware interface layer supporting TCP-IP based network connections. A Bluetooth connection module for Muddleware has been recently developed; this consents to run Muddleware clients and to exploit their potentialities even on handheld devices which do not have WLAN networking capabilities because of their limited hardware.

2.5 Graphics and StbSG

Studierstube ES exploits OpenGL ES\textsuperscript{4} for 3D rendering thus guaranteeing transparent access to 3D graphics hardware on all handheld devices that have a GPU, and software rendering on devices that do not have one. OpenGL ES emerged a few years ago as a cross-platform API for 2D and 3D graphics on embedded devices and consists of a subset of the original OpenGL library for desktop systems. The first versions of the OpenGL ES specifications (identified by 1.x version numbers) were released targeting devices without programmable graphics hardware (fixed-pipeline); the OpenGL ES 2.0 specifications have been recently released and they represent a great improvement because they are explicitly targeting programmable GPUs and therefore also grant support for vertex and fragment shaders.

When creating AR applications with Studierstube ES, developers do not necessarily have to access graphics hardware through low-level OpenGL ES primitives. The OpenInventor-based scenegraph exploited by the original Studierstube for desktop systems has been replaced in Studierstube ES by an XML-based scenegraph; this scenegraph is handled through an embedded scenegraph handler called StbSG. StbSG scenegraphs make intensive use of fields and connections (by reference) between fields, therefore massive data (e.g. geometry data) can be shared between nodes instead of being replicated, thus saving as much memory space as possible;

\footnote{http://www.khronos.org/opengles/}
2. The Studierstube ES framework

furthermore any node can be easily referenced (by asking for a pointer to the desired field) and its inner values can be modified using a C++ API. The tracking module of Studierstube ES is also based on fields: this allows for seamless field connections between the tracking system and the scenegraph handler. A set of tools is available to convert 3D models from various common file formats (Autodesk Scene Export, Wavefront Object, VRML) into XML-based StbSG scenegraph files.

Finally Studierstube ES also helps on the creation of GUIs by providing graphical widgets (i.e. images, text labels and buttons) which are easy to setup and position on screen.

2.6 Developing applications with Studierstube ES

With the Studierstube ES framework it is possible to obtain an AR application even without writing any line of source code. Studierstube ES can in fact be configured through XML files to render 3D objects registered with the real environment. When in the Studierstube ES configuration file we insert the node

```xml
<TrackingTarget name="Marker1" id="1" size="80"/>
```

we define a tracking target on a fiducial marker named Marker1, having ID 1 and a size of 80 units (in the virtual environment, but these can be mapped to real-world units e.g. millimetres). If we also insert in the scenegraph the two nodes

```xml
<MatrixTransform matrix="REF Marker1.matrix" />
<Cube width="80" height="80" depth="80"/>
```

we can then run Studierstube ES (without loading any specific application) to obtain the AR view shown in Figure 2.4.

When developing a complex application there are anyway things which can be naturally described with a Studierstube ES XML file (e.g. structured contents like 3D models) but there are also things that are more naturally implemented with source code (e.g. 3D geometry pre-processing, processing of user input, database updates, transitions in the state of the application) or are completely independent from the framework (e.g. handling a server-side database held on a desktop system). Furthermore when developing multi-user applications which are based on
2.6. Developing applications with Studierstube ES

Figure 2.4: Example of an AR application that can be obtained by only configuring Studierstube ES through XML files (without writing source code).

several interconnected clients and a server it is often necessary to implement server-side applications which perform global maintenance of the application state. The following chapter will describe in detail the design and the implementation of the present thesis project; it will be possible to see how the development of an actual multi-user AR application with Studierstube ES does not consist only on writing XML files but involves also hardware considerations for a distributed architecture, accurate database design and the development of various applications.

As a final note it is necessary to point out the limitations of the Studierstube ES framework. Most of the limitations of the framework are imposed by the underlying hardware (e.g. low-performance CPUs and GPUs, unstable network connections, small-sized displays) and are therefore intrinsic limitations of any framework running on the same platforms. It is anyway necessary to keep in mind that this framework is specifically designed for multi-user handheld AR applications, in particular gaming applications, and is therefore not optimised for the execution of applications which require features that do not pertain to this class of applications (e.g. there is no API for the processing and rendering of volumetric data, although it might be necessary for some cooperative Information Visualisation AR applications); before employing the Studierstube ES framework for a handheld AR project it is therefore necessary to verify that the application we want to develop is within the scope of the framework and that functionalities that do not concern multi-user AR gaming applications are not required.
2. The Studierstube ES framework
Design and implementation

3.1 Project goals

Before the present thesis could start a brainstorming session took place to find the right concept for the project. Related work in handheld AR (Chapter 1) and the functionalities and the potentialities of the Studierstube ES framework (Chapter 2) were both considered. The intention (shared with the supervisor in Graz, Daniel Wagner, and with other researchers from the same Institute) was to design and implement an application that would have exploited the Studierstube ES framework to bring a multi-user experience and which would have emphasised the possibility of users to move inside the environment. These requirements seemed to merge perfectly the mobility and networking characteristics of modern handheld devices with the capability of AR applications to create a shared virtual world registered in 3D with the real one.

As already stated in Section 1.3, AR games are an interesting class of applications for research projects which are focused on testing new technologies because game players are more forgiving than users of professional applications when hardware and software issues occur if they do not hinder the game experience. The present project was therefore decided to be a multi-player AR game running on handheld consoles. Starting from these first considerations, further technological constraints were applied; they are listed in the following paragraphs.

The mobility requirements presented before pointed to a gameplay which forces users to explore the real environment and allows them to perform certain actions only on precise physical locations. It was also decided that gameplay would have exploited not only physical locations (by augmenting them with virtual content)
but also physical adjacency of locations (by mapping them to connections inside the virtual game space).

The target hardware was chosen in advance: it was decided to use Gizmondo consoles\(^1\) (Figure 3.1) because they are compact and solid off-the-shelf systems with several interesting features. They offer GPRS and Bluetooth network connectivity, GPS support and an SD-card slot for memory expansion. They have an embedded nVidia 128-bit GoForce 4500 GPU for 3D acceleration with a hardware transform engine and 1280 KB of built-in graphics memory therefore they grant good 3D rendering performance. Since they are handheld gaming consoles they have a physical user interface specifically designed for gaming applications. A final and very important characteristic of Gizmondos is that they run a Windows CE operating system therefore they are able to execute applications based on the Studierstube ES framework for Windows CE without the necessity to port code to a different software platform.

\[\text{Figure 3.1: Gizmondo handheld console.}\]

It was decided that the game would have been a multi-player game between two opposing teams; this decision made it possible to develop a game concept for a competitive game (between teams) but with strong elements of collaboration (between players from the same team). Since the game is an AR game, it was assumed that

\(^1\)http://en.wikipedia.org/wiki/Gizmondo
both competition and collaboration would have exploited also natural face-to-face user interaction in the real environment.

Another feature that was included in the requirements for the game concept, given the support for networking of Gizmondos, is the capability to maintain a centralised state of the whole application and to share it consistently between all client devices involved in the game. It was therefore decided that the game would have exploited a central server to hold the complete game state at any instant of time, while Gizmondo devices would have acted as clients sending any change in the game state to the central server and relying on it for any update of the game state made by other client devices or by the server itself; this feature had necessarily to exploit Gizmondo’s Bluetooth connection because no other local network technology (e.g. WLAN) is available on this device.

Since players have own private handheld devices that they carry around the game space in any instant of the game session it was decided to introduce gameplay features exploiting private views; different users share a common basic knowledge about the game state but they are also presented with additional private information which is only visible to them. A variation of the private view that was also decided to be included in the gameplay is knowledge-sharing between players from the same team; knowledge-sharing inside the team seemed to be a feature which would have increased the involvement of people in the collaborative part of the gameplay.

A final constraint that was regarded as fundamental was the requirement to base the gameplay on a physical setup easy to build and to replicate in different places. Heavy and bulky hardware and complex registration procedures were not considered as an option because it would have taken a big amount of time and effort to replicate the setup used for testing and this was seen as an obstacle to the diffusion of the present project and of AR technology in general.

All the features identified for the present thesis project are summed up in the following list.

- The project encourages collaborative and competitive behaviours within a multi-player gaming contest, mixing computer-mediated interaction techniques and real-world face-to-face communication.
- The project employs an overview map of the whole game space which presents
a shared game state concurrently updated by all handheld devices involved in the game session; furthermore, some information is visible only to single players (private view) or to single teams (shared-knowledge).

- The project is a distributed application that highly exploits the mobility factor of handheld devices; this implies mixing AR techniques with location-aware Human-Computer Interaction paradigms.

- The project adopts a setup tested in a research lab but easy to build in other game spaces.

- The project is based on a game setup which maps real-world locations and real-world line-of-sight interconnections between locations to a virtual world registered in 3D with the real environment.

The overall goal of this project is to prove the feasibility of implementing all these features in an AR application for handheld devices using the Studierstube ES framework. This project implements some known application paradigms – mainly CSCW and location-awareness – in a handheld AR system and can be considered as intersecting various research areas (CSCW, Location-Based Systems, handheld AR, mobile computing). Projects that merge together all these research topics are still scarce (as seen in Chapter [1]): from a research point of view the present project can therefore be regarded as innovative because it conveys known application paradigms to a quite novel research area like handheld AR.

All these identified features were used as the input for a subsequent brainstorming session focused on creating a game concept that would have included all of them. The outcome of this brainstorming process is the concept presented in the following section.

### 3.2 Game concept

#### 3.2.1 Game setting and gameplay

The game is a multi-player game where players are divided into two competing teams. The game setting is a group of locations which are pastures where cows are quietly grazing; there are also two special locations where a stable for each team is
located. Adjacent locations are connected to each other and form a graph. Cows are not aware of the fact that an alien invasion is currently taking place and have no chance to survive without the help from players; UFOs are present on some of the locations and they are ready to shoot any cow that will try to run from that location.

At the beginning of a game session all cows and UFOs are placed randomly all over the pastures. It is possible for players to lock cows and UFOs so that their team gains exclusive control over their movements and actions. Locked cows and UFOs which have not been used by any player for a certain amount of time will anyway be unlocked and made available again to the other team to prevent obstructive game strategies based on locking the most cows.

The final goal of the game is to save as many cows as possible; cows can only be considered safe when they are located inside a stable: the goal of players is to collect cows from around the game space and to take them safely to the stable that belongs to their team. Players can lock cows and ask them to move between connected locations (any pasture can hold a maximum of four cows). Cows should try to avoid facing UFOs otherwise they might not make it alive to the stable; cows are forced to fight when they move from a pasture that contains a UFO controlled by the opponents: the outcome of the fight can be the death of all cows or their transfer to a hopefully safer location. A secondary goal of players is to prevent the opponents from bringing cows to their stable by obstructing them with the aid of UFOs: UFOs can also be locked and moved (although they cannot be moved to a location containing a stable) to allow the development of strategies that obstruct the paths used by the opponents to move cows.

The winner of the game is the team that saves the most cows: the game therefore ends when there are not enough cows left on pastures for the losing team to catch up with the winning team. A game session can obviously also end in a draw when both teams save the same amount of cows.

3.2.2 Reality and Augmented Reality

The physical game setup consists of a set of fiducial markers (presented in Section 2.3) each representing a different location. A graph of interconnections between fiducial markers is already present in the real environment and depends on line-of-
sight and physical adjacency relations. This is the real game environment that is augmented during the game: each player carries around the game space a Gizmondo that acts as a magic lenses (see Section 1.2).

The interaction between players and the virtual game environment is mediated by the real markers: user actions which concern a specific location of the game can only be performed in the proximity of the corresponding fiducial marker in the real environment (i.e. when such marker is visible to the camera of the device). To perform actions on different locations players must therefore physically move inside the real environment.

When players look at markers with their Gizmondos they can see an augmented 3D environment composed by computer-generated graphics registered in 3D with the real environment. Every marker corresponds to a different virtual location of the game; on each marker it is possible to see the cows and the UFO which are currently positioned on such location. 3D arrows are also visualised in AR on top of a marker; arrows are always pointing towards the direction of markers connected with the currently visible one. If the visible marker corresponds to a stable then a 3D stable is visualised instead of cows and UFOs.

A virtual overview map showing the physical position of each pasture and of the stables inside the real game environment is then available to all players and is visualised directly on their Gizmondos as a 2D overlay; players are able to see on their overview map how many cows and UFOs are positioned on each location, if such location is currently visible by at least one player from their team (this mechanism implements both the private view and the shared knowledge requirements). The overview map is also registered with the environment, albeit in 2D: as soon as a marker becomes visible by the Gizmondo, the map is automatically oriented to be coherent with the view direction of the player (this will be explained in Section 3.5).

Other virtual content is presented to users on their Gizmondos but it is not registered with the real environment. Overlay text boxes are used to inform about the current state of the game. Two sliders are also employed during fights between cows and UFOs; during such fights the team controlling cows must select a value from a moving slider and the opponents must do the same for the UFO: the values selected by both teams are then compared and the higher value wins the fight (cows win also if values are equal).
3.3. Architecture of the system and hardware setup

The whole AR view is presented extensively in Section 3.5 and in Section 3.6.2

3.2.3 Game setup

A game environment must be set up before starting a game session. Fiducial markers must be stuck on walls and an overview map for the environment must be defined; the location of each marker on walls must then be specified, together with interconnections between locations: these should ideally be mapped on line-of-sight and physical adjacency relations already present in the real environment. There is no necessity to be very precise because game mechanics do not require high accuracy. An example of a map for the game setup can be seen in Figure 3.2.

![Figure 3.2: Example of a game setup where each pasture location is represented by a circle and each stable is drawn with a small icon. Walls are drawn in black while blue lines represent connections between different locations.](image)

A tool to help this map definition process is presented in Section 3.7. This same tool is also used to randomly initialise the position of cows and UFOs inside the virtual game space.

3.3 Architecture of the system and hardware setup

As said in Section 3.1, the devices used by the present project are Gizmondo handheld consoles. A necessity to be considered when designing the architecture for this
project was the need for a shared global game state consistently held in a safe repository: this requirement was fulfilled with the adoption of a central server running a Muddleware server (see Section 2.4). The architecture that was adopted is therefore a client-server architecture where the central server is run by a desktop computer while all clients are Gizmondo consoles exploiting Bluetooth connections. The server is also running a custom game controller application which is responsible for the execution of transitions in the game state: some changes are in fact requested by client devices but performed server-side by this game controller, as explained in Section 3.7. The global architecture of the implemented system can be seen in Figure 3.3.

![Architecture of the system implemented for the present thesis project.](image)

The server must be able to accept Bluetooth connections, but this is handled seamlessly with the aid of a Bluetooth USB adapter and of the Bluetooth bridge which is part of Muddleware. At the present time there is a limitation given by the maximum number of connection accepted by a USB Bluetooth adapter (seven devices...
at most$^2$ that limits the number of players which can join a game session; this limitation can be theoretically solved adding more USB Bluetooth adapters although practical problems like interference between devices and driver issues prevent from doing this effectively. Another limitation is given by the short range of Bluetooth connections (around 10 metres but depending also on the power of transmitters and on the occlusions in the environment); future versions of the framework might try to address this limitation by allowing the use of many Bluetooth access points to create various Bluetooth cells in the environment and by implementing roaming over such Bluetooth cells (this would still allow a maximum of 7 devices per USB adapter).

The current version of the framework limits the present thesis project to small physical setups (no more than 10 metres of radius from the Bluetooth transmitter) and to a maximum of 7 concurrent players.

As can be seen from Figure 3.3, several programs are involved in the present thesis project: a part of them is composed by custom programs designed and implemented specifically for this project while the remaining part consists of third-part frameworks, servers or tools. The central server is running

- **Muddleware server**: it is a Database Management System developed internally and maintained by the ICG in TU Graz and it has already been presented in Section 2.4; it is fundamental for handling the global game state in a central database and for sharing it with all client devices; the design process for the database used in this project will be described in Section 3.4.

- **Bluetooth bridge**: it is a program which accepts connections from Bluetooth-equipped devices and acts as a bridge connecting such devices to the Muddleware server; this program was also presented in Section 2.4.

- **Server-side game controller**: it is a custom program implemented to execute transitions in the game state which cannot be performed directly by a single client because they are not dependent on any player (e.g. handling timeouts) or because they are dependent on more than one player (e.g. computing the outcome of a fight); this program will be presented in Section 3.7.

Each Gizmondo client is running an instance of

• **Studierstube ES framework**: it is a framework for handheld AR applications and it has been described in Chapter 2; when compiled for Gizmondos it already includes an embedded Bluetooth module and it uses such module as the default peripheral for network connections;

• **Thesis project**: the client-side AR thesis application; it is stored as a DLL and dynamically loaded by the Studierstube ES framework; this application will be described in detail in Section 3.6

A hardware solution suggested by the use of the Studierstube ES framework is the use of fiducial markers (presented in Section 2.3) to physically represent each location in the real game environment. In the present project all fiducial markers have been stuck on walls but the solutions proposed in this thesis do also allow the use of markers mounted on other types of structure (e.g. on a movable rack); the adoption of movable markers that can be put in the middle of rooms could indeed augment their visibility and the flexibility of the system.

Given the fact that the gameplay strongly exploits virtual connections between locations (and at the same time physical adjacency of markers) care must be taken when mapping physical relations on virtual ones; this mapping should be immediate to understand. An assumption that was made is that when markers are not positioned too close to each other an imprecise representation of the game space on the server does not cause ambiguities if proper spatial hints are given by the user interface (which will be presented in Section 3.5). An interesting outcome of proving these considerations would be the small setup time for the proposed system: due to the low precision needed and the lack of calibration procedures all fiducial markers can be quickly installed anywhere; with the aid of a graphical map editor a coarse map of any game environment could then be easily drawn and markers could be positioned and interconnected with a few mouse clicks. This would mean that the present projects exploits a highly portable AR setup.

The work that was performed for the present thesis project is summed up in the following list; all the listed points will be described in detail in the following sections.

• A database to hold the complete state of the application on the Middleware server was designed.
A user interface to help the navigation of players inside the game environment was designed and then refined using some feedback taken from two informal sessions of user testing.

A client-side application to be run on each Gizmondo console was designed and implemented in C++ using the Studierstube ES framework. Several classes were created adding custom features to those already granted by the framework; these features include methods for an easy access to the state of hardware buttons, 3D geometry pre-processing, visualisation of the state of the system through graphical overlays, a layer of custom data structures and methods to easily handle the data stored in such structures, handling of all state transitions in the game.

A server-side application was implemented in C++ to perform all game transitions that cannot be done directly on client devices (transitions that do not depend on any player and transitions that depend on more than one player).

### 3.4 Database design

The complete game state for the present thesis project is held on a database in a central Muddleware server and is consistently updated and shared between all client devices involved in the game session. All entities and relationships involved in the game were identified and classified as either static or dynamic; static entities and relationships are considered to be those that do not change during the course of a single game session, while dynamic ones are continuously changing also during a game session. Since database synchronisation is performed through wireless Bluetooth connections it was necessary to have a low number of dynamic entities to reduce the overhead required on the central server to synchronise all client devices and to reduce the Bluetooth network traffic. The resulting Entity-Relationship diagram is presented in Figure 3.4; all entities and relationships are listed and described in detail below.

- A **Wall** is a direct representation of a physical wall of the game environment. A wall is identified by a unique ID number and is defined by specifying its two extremes as 2D points because the environment is considered to be located on
Figure 3.4: Entity-Relationship diagram for the database.
one single floor and its representation consists of an orthographic projection from above.

- A Marker represents one of the fiducial marker employed to identify virtual game locations inside the real environment; in this project every marker is assumed to be stuck on a wall. Every marker is identified by a unique ID, the same ID also used by ARToolkitPlus (presented in Section 2.3). A marker is always positioned on a wall (at a certain percentage of the wall and on a specific side of it). Given their tight coupling markers and game locations are treated as one inside the database; they are effectively the connection point where the real environment (containing the physical markers) becomes merged with the virtual game space (based on virtual locations which can be either pastures or stables). Markers (and therefore virtual locations) are connected to each other by links.

- A Movable Object represents any game object that can be locked by players and then transferred from one location to a connected one. Movable objects generalise two other more specific entities, Cow and UFO, which represent the two types of object actually employed during gameplay. All movable objects are identified by a unique ID. They all have a status which varies during a game session depending on the actions performed by players. The status assumed by a cow can be dead (it disappears from the game space), waiting (unlocked and waiting on a location), locked (locked by a team and waiting on a location), moving (locked by a team and waiting to be transferred from a location to another) and fighting (locked by a team, waiting to be transferred from a location to another and engaged in a fight); UFOs can assume the same state values apart from being dead. When a fight between cows and a UFO is engaged players controlling the two parties will be forced to select a value from a moving slider and the comparison between values for cows and for the UFO will decide the outcome of the fight: the value selected during a fight is stored as the Value attribute of a movable object. A movable object is always positioned on some marker (location) and, when required by the controlling player, it can be asked to move to another marker; the transfer is requested but not performed by client devices because transfers between markers are checked and validated by the server-side controller before being
executed. Finally a cow also has a slot attribute necessary for the StbSG rendering engine: four cow slots are present in the scenegraph and each of them corresponds to a separate 3D model therefore every cow positioned on a marker gets one of these 3D models assigned and it must maintain the number of this slot as an own attribute; further details on the use of the scenegraph will be presented in Section 3.6.2.

- A **Team** represents one of the teams involved in a game session. A team is identified by a unique ID and it has a name to represent it in a more natural way for players. Each team is assigned a colour which identifies it and which is used in the user interface as a visual cue for locked movable objects and other graphical widgets: details on these graphical cues will be further in Section 3.6.2. Every team has a status attribute which tells if the game session is not yet over and the team is still playing or if the game session is over; when the game is over the status attribute also specifies whether the team has lost or won, or if the game ended in a draw. All movable objects can be locked to a team (but no more than one team at once) and they must be locked before any action is performed on them by players; the locking team gains exclusive control over the movable object. A team also has one and only one marker where its own stable is located; this location will be the safe place where all players belonging to the team should try to transfer as many cows as possible.

- A **Player** represents a physical player involved in a game session. Players are identified by unique IDs and also have names to represent them more naturally. Every player plays for one and only one team. During a game sessions players can be looking at markers and this information is exploited to implement the private view and knowledge-sharing mechanisms discussed in Section 3.5; players therefore maintain in an attribute the marker they’re currently looking at (or a special value to indicate that there is currently no marker visible to a given player).

Walls, markers and the relationships between them are considered to be static because they constitute the physical setup where game sessions take place. The remaining entities and relationships are dynamic because they represent virtual game content which is manipulated by players and which therefore changes during a game
3.4. Database design

The ER diagram shown in Figure 3.4 has been translated to the following relational database:

- **WALL**(ID, StartX, StartY, EndX, EndY)
- **MARKER**(MarkerID, WallID, WallSide, Percentage)
- **LINK**(MarkerID1, MarkerID2)
- **Cow**(ID, Status, MarkerID, Slot, LockedTo, MovingToMarker, Value)
- **UFO**(ID, Status, MarkerID, LockedTo, MovingToMarker, Value)
- **PLAYER**(ID, Name, Team, LookingAt)
- **TEAM**(ID, Name, StableID, ColorR, ColorG, ColorB, Status)

The relational database has then been implemented as an XML-based Muddleware database, resulting to XML code that has the structure presented below.

```xml
<DB>
  <Wall id="" start_x="" start_y="" end_x="" end_y="" />
  <Marker marker_id="" wall_id="" wall_side="" percentage="" />
  <Link marker_id_1="" marker_id_2="" />
  <Cow id="" status="" marker_id="" slot="" locked_to=""
     moving_to_marker="" value="" />
  <UFO id="" status="" marker_id="" locked_to="" moving_to_marker="" value="" />
  <Player id="" name="" team="" looking_at="" />
  <Team id="" name="" stable_id="" r="" g="" b="" status="" />
</DB>
```

The XML structure presented above is anyway only an indication of how the database is stored on disk and it was not created manually. Database initialization in the present thesis project is performed by the server-side game controller at startup and before every gaming session; this will be discussed in detail in Section 3.7.
3.5 User Interface

Since the gameplay for the current thesis project consists mostly on moving objects between connected locations it was necessary to create a user interface capable to inform users of the available connections from and to a given marker through AR clues and other generic computer-generated graphical aids.

The first user interface design was based on AR arrows drawn on top of the visible marker and oriented in 3D space towards the markers connected to it; AR arrows were merged and registered in 3D with the real environment. A further available clue was given by an overview map of the whole game environment (a static orthographic projection of the environment from above) visualised to users as a semi-transparent overlay. Screenshots of the implementation made for this first user interface design are presented in Figure 3.5.

![Figure 3.5: Implementation of the first user interface design; the screenshot shows the application view while looking at a marker. 3D arrows are pointing to the connected markers. An overview map showing an orthographic projection of the whole game environment can be visualised on-demand.](image)

The overview map in the first user interface design was not immediate to understand because it was not clear which marker was the currently visible one; furthermore the orientation of the map was static and this was introducing well-known problems that arise when using a static 2D map while actively navigating a 3D environment [9], problems which are usually related with errors occurring during mental rotation of the map. The clearness of the overview map was therefore improved by dynamically centring it on the currently visible marker; this was further stressed drawing the currently visible marker with a different icon. By dynamically changing also the

orientation of the map depending on the position of the viewer the burden of mental rotation was removed from users; the orientation of the vertical axis of the map (pointing from the bottom to the top of the screen) was kept coherent with the view direction of the user, thus obtaining a so-called forward-up map (or track-up map).

It would have also been interesting to exploit the distance from the marker to center the map on the actual position of the user instead of centring it on the currently visible marker. This was anyway not possible because there is no correlation between the measurement units exploited in the virtual representation of the environment and the actual measures in the real world; this is a direct consequence of the adoption of an easy setup system: since no calibration is needed it is not possible to obtain a correct mapping between the representation of walls and their actual size in the real environment.

Figure 3.6: First application to test the user interface. 3D arrows are used to augment the real environment and a helper map is visualised as a semi-transparent overlay. The currently visible marker is stressed on the map with the aid of a special icon, and the map orientation is kept coherent with the position and the view direction of the user.

This second user interface design was implemented in a small test application; this application was exploited to perform an informal user test internally within the personnel of the research lab: users involved included also researchers working on user interfaces. A picture from a user testing session and the relative screenshot...
of the application are shown in Figure 3.6. Users were asked to explore a game environment setup covering three rooms of the institute; during the exploration users were explicitly asked to look at a marker and to physically identify the connected ones inside the real environment. Informal testing was conducted in the form of cooperative evaluation\(^3\); this technique was chosen because the test task (explore the environment) was deliberately vague to encourage subjective behaviours to emerge and because the level of technical expertise of users involved in testing allowed them to discuss the user interface while remaining focused on the task. The second user interface design proved to be generally intuitive to use; it is anyway not possible to make strong claims because results were taken from a group of technologically expert users which already knew the physical environment were tests took place. Users were able to observe markers and look to the directions pointed by AR arrows to roughly identify the areas where connected markers should have been located. The overview map was particularly appreciated as a refining tool to understand what was the exact position of connected markers in the direction already identified using the AR arrows; another feature that was considered very helpful was the semi-transparency of the overview map which allowed users to use both the AR clue given by 3D arrows and the more high-level view given by the overview map. A few criticisms were raised on the overview map, all pointing to two main problems:

- while the mapping between the currently visible marker and the highlighted marker in the overview map was clear, there was no evident mapping between the AR arrows and the links visualised in the map;
- there was no way for users to understand whether the marker they believed to be the connected one was really the connected one or not; since all markers stuck on walls are on paper sheets containing also their unique IDs, it was suggested to draw marker IDs also on the overview map so that users could get an immediate confirmation of the correctness of their guess.

The user interface design was modified to introduce improvements which would have contrasted the issues raised by the first user test. The icon representing the currently

\(^3\)In the cooperative evaluation user testing technique users are asked to think aloud to explain what they are doing and why they are doing it; users are also encouraged to actively participate into the evaluation process by asking questions whenever they have any doubt and by immediately presenting criticisms to the evaluator.
visible marker and the connections from and to it were colour-coded on the overlay map and next to each marker the corresponding unique ID was visualised. Furthermore the unique ID of connected markers was also adopted in the AR visualisation and shown above the 3D arrows. A second user testing session was then performed with the same modalities as the first one (explore the environment, physically identify markers, cooperative evaluation) but with a broader audience including also users that were not computer experts and which had no previous knowledge of the physical environment were tests took place.

![Figure 3.7: Second application to test the user interface. 3D arrows and marker IDs are used to augment the real environment and a helper map is visualised as a semi-transparent overlay. The currently visible marker and its connection are colour-coded and the map orientation is kept coherent with the position and the view direction of the user. Marker IDs are shown both on the overview map and in the AR view.](image)

The second informal user test showed that even non technical users understand the user interface and are quickly able to physically identify what markers are connected to the currently visible one. During this test the most common strategy was similar to the one observed during the first user test; users tend to look at the arrows to identify the spatial direction where connected markers can be found and then refine the spatial search with the aid of the overview map. With the final user interface anyway marker IDs are also extensively exploited as a proof of correctness.

Colour-coding has also been extensively exploited in both the AR view and the bidimensional overlays to inform players about what team they are playing for and about what cows and UFOs are currently locked to their team. In the AR view 3D models for unlocked cows and UFOs are rendered with a neutral grey colour but as
soon as they are locked to a team they immediately assume the colour of that team. At the same time all overlays on a device belonging to a player assume the colour of the team that player is playing for, thus eliminating ambiguities about which is the own colour. Finally on the overview map and in the AR view stables are coloured with the colour of the team that owns them so that they can be easily identified by players. This graphical mapping is shown in Figure 3.8.

Figure 3.8: Colour-coding to identify the team and the objects locked to the team. All the overlays assume the colour of the team and team colours are also used to identify the objects which are currently locked and what team locked them. In these images the player is playing for the blue team. In the right image the UFO and a cow are locked to the blue team while two cows are locked to the red team.

Figure 3.9: Mapping of the state of arrows on their graphical aspect. Dark-coloured objects are not valid selections while those with a bright tint are valid ones. The currently selected arrow assumes a red colour, other arrows assume a grey colour. The validity of arrows is also mapped on the 3D models used. From left to right: valid arrow non selected, valid arrow selected, non valid arrow non selected, non valid arrow selected.
AR arrows are also colour-coded to identify their varying state. When choosing a
direction to transfer cows or a UFO it is necessary to identify in the user interface
what arrow is the currently selected one; the selected arrow always assumes a shade
of red, while all other arrows maintain a blue-grey colour. Some arrows are not valid
selections because a cow cannot be transferred to a location that already contains
four other cows and UFOs cannot be moved to locations containing a stable or
another UFO; in the user interface valid arrows are identified by their bright colour
while invalid arrows always have a dark tint. Since mapping two values (if an arrow
is valid and if it is selected) on a single attribute is not a good practice, the validity
of an arrow is also mapped on its geometry: a valid arrow is visualised as a normal
arrow while a non-valid arrow is represented only by a rectangle (an arrow without
the point). This mapping can be seen in Figure 3.9.

3.6 Client-side application

The design of the client-side application started by dividing all needed functionalities
into several functionality groups which in part mirror the subdivision adopted inside
the Studierstube framework and presented in Figure 2.1. The functionality groups
which were identified are listed below.

- **Input**: this group contains functionalities to handle the hardware interface
events; these functionalities exploit the hardware abstraction layer already
provided by the Studierstube ES framework to store flags which tell whether
a button on the Gizmondo is pressed or not.

- **Graphics**: this group includes all functionalities to draw some computer-
generated graphics on the screen or to pre-process graphics that will be vi-
ualised later. It is mainly composed by functionalities for 3D geometry pre-
processing (because the actual rendering is handled directly by the StbSG
handler presented in Section 2.2) and for on-screen visualisation of informa-
tion through overlays.

- **Database**: this group contains all functionalities necessary to seamlessly handle
the connection with the central Muddleware database. Functionalities in this
group include storing data taken from the database and accessing it through
custom data structures; other functionalities are those to maintain these data structures consistent after local modifications (therefore sending the updated values to the database) and after remote modifications (therefore asking for any updated value and receiving it).

- **Gameplay**: this group contains all high-level functionalities which are necessary to handle the gameplay flow. Functionalities are also available to store the current application state and to query the game state handler for state-dependent data. Gameplay functionalities also redirect player actions to the appropriate bottom layer; player actions can send outputs to the graphics group (when an action causes a change of the geometry of the scene or of the colour of an object) and to the database group (when an action causes the variation of one or more values which have to be then sent to the central server) and they can generate input from the input group (when the current state of a button is needed) or require updates from the database group (when certain data is required).

A set of classes was then designed for each functionality group to perform the given functionalities. Figure 3.10 shows all the classes designed and implemented for the present project and groups them by their functionality group. The parts of the Studierstube ES framework which are exploited by the different classes are stressed in the diagram; these parts are also connected to the classes which are using them. All functionality groups and the relative classes will be described more in detail in the following sections.

### 3.6.1 Input

The *Input* functionality group contains a single class also called *Input* which is responsible for granting an interface for events raised by the hardware abstraction level contained in Studierstube ES. The Input class maintains in memory the state of every hardware button on the Gizmondo, and it can be queried to know whether a certain button is pressed or not; when the button is pressed the class can also be queried to know if the button has just been pressed or if it has been kept pressed since previous frames.
3.6. Client-side application

Figure 3.10: Diagram representing all the classes designed and implemented for the present thesis project, grouped by their functionality. On the bottom the parts of the Studierstube ES framework which are exploited by each class are also shown.

3.6.2 Graphics

The Graphics functionality group contains all classes which are necessary to visualise computer-generated graphics in the present application; graphics functionalities include 3D geometry pre-processing and visualisation of the game state through 3D geometry and 2D overlays. The main classes falling inside this functionality group are Graphics, Overlay and Overview; all other classes are used only as an internal support by one of the three main graphics classes and will be discussed later when the corresponding main class is presented.

The Graphics class is necessary to perform all computations concerning 3D geometry. This class is based on the StbSG scenegraph handler of Studierstube ES (described
in Section 2.5 and is used to perform pre-processing on the 3D geometry contained in the scenegraph before it is rendered through the StbSG handler. Several pre-processing steps are executed on the scenegraph before it is rendered.

Some nodes of the scenegraph must be activated and some others must be deactivated: the scenegraph contains in fact all 3D models that are needed by the application (i.e. 3D models for a UFO, four cows, four directional arrows, a stable) and the geometry that must be actually rendered on a given marker is just a subset of the whole scenegraph. A selection must be therefore performed to activate only nodes corresponding to 3D models which must be rendered and to deactivate the remaining nodes; this is done by querying StbSG for pointers to each node representing a 3D model and then activating or deactivating it depending on the current game state.

Another pre-processing step is then performed on nodes which have been activated to perform colour-coding as described in Section 3.5. All objects in the scenegraph come coloured in a shade of grey and the colour actually used for their rendering is decided at run-time. The ScenegraphUFO and ScenegraphCow classes are used to store pointers to the nodes of the scenegraph containing 3D models; these pointers are exploited during pre-processing to change the colour of the UFO and of each cow depending on whether they are locked (their colour becomes the colour for the team that locked them) or unlocked (they maintain the original gray colour).

Arrows are also subject to pre-processing; like for the UFO and for cows a support data structure called ScenegraphArrow is used to store pointers to scenegraph nodes related with each 3D model and to quickly access their attributes. As said in Section 3.5 arrows are colour-coded depending on whether or not they represent the currently selected direction to transfer cows or the UFO. Unselected arrows maintain a blue colour while selected ones are rendered in a shade of red; if the output direction is a valid one then the colour is bright while if the connected marker cannot receive any more cows (if we are transferring cows and it already contains four cows) or a UFO (if we are transferring the UFO and the marker is a stable or it already contains a UFO) then the colour is dark. Mapping two values (if the arrow is selected and if it is a valid direction) on the same attribute (the colour of the arrow) is not a good practice therefore two separate 3D models are also used to distinguish a valid arrow from a non-valid one; depending on the type of each arrow this pre-processing
phase enables the corresponding 3D model in the scenegraph. Further arrow pre-
processing is used to set the arrow rotation according to the vector that goes from
the currently visible marker to the connected one. The corresponding marker ID is
finally set for rendering on each arrow.

Figure 3.11: Example of a 3D object rendered with the normal OpenGL shading (left)
compared to the same object rendered with cartoon shading (right).

A final pre-processing step, used to obtain a funnier look of the rendered scene,
consists of applying a toon shading algorithm to the 3D geometry. Shading is the
process of computing colour values for a 3D geometry and is usually based on pa-
rameters describing the lighting conditions in the environment. Toon shading is a
particular shading technique that generates cartoon-like renderings instead of pho-
torealistic ones; an example of the difference between normal OpenGL shading and
toon shading can be seen in Figure 3.11. While normal OpenGL shading computes
colour values for each vertex and these values are then smoothly interpolated when
rendering the object, toon shading computes in real-time texture coordinates for
each vertex and then applies a special 1D texture to the object\textsuperscript{4}. These computa-

\textsuperscript{4}The use of toon shading on already-textured models requires multitexturing with support for
at least two textures. Since multitexturing was not originally supported by the Studierstube ES
framework, an extension of the framework was implemented as a part of the present project; this
extension was implemented modifying the framework source code and maintaining its structure
(multitexturing can be specified directly inside the scenegraph). The implemented extension is now
part of the framework and all other users of Studierstube ES can use it when multitexturing is
required.
tions are in both cases based on the cosine of the angle between the light vector hitting the 3D object and the normal vector corresponding to each vertex. Given a light vector $\mathbf{L}$ from a given vertex to a light source and a normal vector $\mathbf{N}$ for the same vertex, the cosine of the angle $\alpha$ between $\mathbf{L}$ and $\mathbf{N}$ can be easily computed as the dot product $\cos \alpha = \mathbf{L} \cdot \mathbf{N}$ (if both vectors are normalised); when $\cos \alpha$ is clamped to $[0, 1]$ it can be used to represent the so-called *diffuse component* of shading which is 1 when the light vector is parallel to the normal vector, 0 when the two vectors are orthogonal and assumes intermediate values in between. While normal OpenGL shading uses the diffuse component directly for vertex colours, toon shading uses it as a texture coordinate for the texture shown in Figure 3.12 which is used to translate smoothly-changing cosine values into ranges having constant colours: this gives the cartoon-style look of an object drawn with sharp-edged colour patches.

Figure 3.12: Example of a 1D texture used for toon shading. Leftmost values have texture coordinates close to 0 and rightmost values have texture coordinates close to 1. This texture converts smoothly-changing texture coordinates into ranges having a constant colour.

Figure 3.13: 2D overlays used during a game session: a text box (top) tells the name of the player using the device, two sliders (side) are used during mini-games, another text box (bottom) tells the game action currently selected by the player. Other text boxes (centre) are popup windows with a timeout and give extra information on the progress of the game.

The *Overlay* class is necessary to visualise all bidimensional overlays (apart from the overview map) during a game session. The overlays employed by the present project
are two text boxes that inform about the name of the player and the currently selected action (player actions are described in Section 3.6.4), popup text boxes (which exploit the Popup class) and two sliders (implemented by the Slider class) which are used during fights between cows and a UFO. The Overlay class uses both low-level rendering (through OpenGL ES) and higher-level Graphical User Interface (GUI) methods of the Studierstube ES framework.

![Figure 3.14: Overview map shown as a 2D overlay.](image)

The Overview class is necessary to visualise the overview map of the game environment as a bidimensional overlay. It checks the currently visible marker and then updates the position and orientation of the overview map accordingly so that it fits with the actual view of the player; the colour of the currently visible marker and of its connections are also modified. This class also implements local-awareness and shared-knowledge by presenting the players with additional information for each marker; if the marker is a stable then the number of cows on it is visualised and the stable icon is colour-coded with the colour of the team that owns it; if the marker is a pasture then all cows and UFOs on it are shown, but only if the marker is currently visible to the player looking at the map or to one of the players in the same team.

### 3.6.3 Database

The main class falling inside the Database functionality group is called DataManager. All remaining classes in the same functionality group are support classes which
store and offer access to the corresponding type of data present in the database. Support classes that represent a dynamic database entry (presented in Section 3.4) also handle per-frame synchronisation with the database on the central server; the synchronisation is seamlessly handled exploiting shared-memory data structures offered by Muddleware (presented in Section 2.4).

To ease the access to data all support classes implement methods to perform often needed queries; this greatly simplifies all the other classes of the application because they can query Database classes to directly obtain dynamically-computed values instead of receiving database values and being forced to do computations on their own. A single call of a method can be used to get from a marker the number of cows on the location it represents, the number of unlocked cows, the maximum number of cows that it can receive before the four-cow limit is reached and so on. A team can be queried to know if any of the players belonging to it are currently looking at a specific marker. Other support classes offer analogous methods to perform complex queries.

3.6.4 Gameplay

Classes which are inside the Gameplay functionality group are high-level classes which are responsible for the correct flow of gameplay. The MainApplication class is responsible for starting the application, creating the Studierstube ES window and looping through rendering and event processing (for both user-triggered and system-triggered events). The application can be in various modes; depending on the current mode different classes are exploited to store the game state and to perform mode-dependent computations.

At startup, when users are asked to select a player name before they can start to play, the application is in SelectingPlayer mode; a class called PlayerSelection is used to show a menu where users can select their own player name and their team before they can join the game session. The menu visualised by this class is shown in Figure 3.15.

When the application is in Playing mode (all the time during a game session) the GameState class is used to hold the global game state, to handle transitions between game states and to obtain game-state dependent data values. The GameState class offers a very rich interface to all other classes involved in the present application;
other classes can query GameState to quickly obtain many state-dependent values. This class stores pointers to the currently visible marker (if there is any) and to the player which is using the application, therefore queries can be made to know e.g. how many cows and UFOs are on the currently visible marker (or on any other marker) and whether they have been locked by the player or not, how many cows and UFOs are locked on any marker by the opponents. The possible actions that a player can perform during a game session are SelectingCows, MovingCows, SelectingUFO, MovingUFO, Fighting and WatchingStable; the availability of a certain action depends on the current game state (e.g. if the player is looking at a location containing a stable she can only perform the WatchingStable action while a player who has not locked any cow cannot perform a MovingCows action). The GameState class is therefore also used to validate actions selected by players before activating them. A final duty of this class is to ask the central database to perform all transitions in the game state that must be handled server-side by the game controller presented in Section 3.7; these actions are transferring cows and UFOs and beginning and ending fights.

When the game is over (may it be a victory, a loss or a draw) the application switches to GameOver mode; in this mode players cannot perform any action and the visualisation shows a static message box with the outcome of the game. No support class is therefore needed for this state because there is no rendering performed and
no action or game transition to handle.

### 3.7 Server-side game controller

Some state transitions involved in the gameplay are not directly related to a specific player because they do not depend on any player or because they depend on more than one player. Even when such transitions are triggered by a single player they rely on a broad knowledge of the whole game state and they modify data structures which are not controlled by the player who started the transition; the transitions that cannot be handled by a single client fall within the classes listed below.

- **Timeouts** to release locked objects and to end fights cannot be controlled by a client because they do not depend on any player.

- **Transitions** that transfer cows and UFOs from one marker to a connected one should not be performed client-side. Although cows and UFOs are locked to a team and a transfer action is always triggered by a single client it is fundamental to validate the transfer action before actually moving objects; the validation step involves a check of the current game state: a UFO cannot be moved to a marker which contains a stable and when cows are moved it is necessary to verify the presence of a UFO locked by the opponents before deciding if cows can be moved or if both the cows and the UFO should be engaged in a fight.

- **Transitions** that regard fights cannot be performed client-side because all clients are involved as peers in such transition. The outcome of a fight should be decided by an independent and impartial external game controller.

A server-side game controller has been implemented to perform these game transitions whose responsibility cannot be assigned to a single client. The implemented game controller is based on a Muddleware client and in the test setup used for the present project runs directly on the same desktop computer as the Muddleware server (therefore all connection are local and fast).

The game controller includes also functionalities to initialise the database. At the present time it contains methods to insert into the Muddleware server a hard-coded
overview map for the game environment used for testing and to randomly position cows and UFOs inside this game environment. It also presents to users a view of all game state transitions through a text-based console, as shown in Figure 3.16. As explained in Section 4.1, this text-based console will ideally be replaced by a graphical interface that will allow a quick definition of the overview map (before game sessions) and a more intuitive overview of the game state (during game sessions).
4.1 Conclusions and future work

The present thesis project is a handheld AR game but the game itself was not the main target of the project: the overall goal of this project was to use the Studierstube ES framework to prove the feasibility of implementing and running well-known software paradigms (CSCW, location-based presentation of data) and new one (a novel solution for user navigation) on mass-market handheld devices using AR.

As shown in Section 1.3 CSCW applications exploiting AR are still not common on handheld devices: the present project implements both collaborative and competitive computer-mediated user interaction on a distributed client-server architecture; furthermore collaborative and competitive behaviours are enforced by face-to-face interactions which can be naturally achieved thanks to the adoption of AR and of a non-intrusive hardware. Location-based AR applications that present users with spatial data are emerging (they have been presented in Section 1.5) and are a compelling research area because of the informative potential in augmenting the already very instructive real environment with further computer-generated knowledge; this project implements location-based presentation of information because of the mapping that subsists between the real world (markers and connections between them) and the virtual game world (locations and connections between them) and because of the central role of user mobility in the implemented application.

A new software solution which has not been implemented before (to the best of our knowledge) is the navigation system designed for the present project. Computer-aided user navigation is not a novel topic and it has already been implemented in the past, also with the aid of AR clues (some projects have been presented in Section
Location-based systems usually require registration with precise models of the environment (e.g. systems for GPS-based navigation, but also the Signpost application presented in Section 1.5 which requires an architectural model for the building). In the proposed system there is no necessity to position markers very precisely: the setup used for testing inside the research lab (which covers three rooms of the institute) and its definition in the database for the virtual environment were both built in a few minutes (time requirements for larger environments are supposed to increase linearly with the size of the covered area). The proposed system can be considered potentially useful for giving coarse navigation instructions inside buildings (e.g. to give visual instructions about directions to take when looking for a room in a university).

The developed solution for user navigation was subject to two informal user testing sessions. Users performed very well when asked to explore the game environment and to identify a marker using only the AR clues presented on an adjacent marker. Users involved in the tests stated that it was easy to understand the AR arrows and the direction pointed by them; with the aid of the overview map and the marker numbers it was then possible to quickly identify a searched marker inside the real environment. Formal user tests are left as a future work: only after those tests it will be possible to claim the good usability of the implemented solution for user navigation; the preliminary results are anyway very encouraging because of the appreciation received from both technical and non-technical users. Possible future work also includes testing the whole game with a representative group of potential users; this testing is regarded to be fundamental before any strong claim on the usability and the effectiveness of the proposed system and the fun factor of the game can be made.

A final future work which is being considered at the moment is to evaluate the market potential of the implemented project. As seen in Section 1.4 many handheld AR games are emerging; some of them are commercial products currently available on the market. It is therefore regarded to be very interesting to understand if the developed game concept and the implemented system could both converge to a product appreciated by the current market of gamers. The marketability of the project will also depend on the development of an easy-to-use and more intuitive game controller (to replace the one presented in Section 3.7) and the inclusion of
a graphical tool to create the overview map; the implementation of these tools is considered to be a possible future work.
4. Conclusions
Bibliography


[8] Adrian David Cheok, Siew Wan Fong, Kok Hwee Goh, Xubo Yang, Wei Liu, Farzam Farzbiz, and Yu Li. Human pacman: A mobile entertainment system with ubiquitous computing and tangible interaction over a wide outdoor area.


[18] Carsten Magerkurth, Timo Engelke, and Maral Memisoglu. Augmenting the
time domain with physical and social elements: towards a paradigm shift
in computer entertainment technology. In ACE ’04: Proceedings of the 2004
ACM SIGCHI International Conference on Advances in computer entertain-

[19] Regan L. Mandryk and Diego S. Maranan. False prophets: exploring hybrid
board/video games. In CHI ’02: CHI ’02 extended abstracts on Human factors

reality: a class of displays on the reality-virtuality continuum. Telemanipulator

[21] Dmitry Mogilev, Kiyoshi Kiyokawa, Mark Billinghurst, and Jarrell Pair. Ar
pad: an interface for face-to-face ar collaboration. In CHI ’02: CHI ’02 extended
abstracts on Human factors in computing systems, pages 654–655, New York,

[22] Mathias Möhring, Christian Lessig, and Oliver Bimber. Video see-through ar
on consumer cell-phones. In ISMAR ’04: Proceedings of the Third IEEE and
ACM International Symposium on Mixed and Augmented Reality (ISMAR’04),

[23] Wolfgang Narzt, Gustav Pomberger, Alois Ferscha, Dieter Kolb, Reiner Müller,
Jan Wieghardt, Horst Hortner, and Christopher Lindinger. Pervasive informa-
tion acquisition for mobile AR-navigation systems. Mobile Computing Systems
2003.

IEEE Computer Society.


