# A Case for Design Patterns supporting the Development of Mobile Mixed Reality Games

Richard Wetzel
The Mixed Reality Laboratory
University of Nottingham
UK NG8 1BB
psxrw3@nottingham.ac.uk

## ABSTRACT
This paper proposes a design pattern language applicable to mobile mixed reality games. While on the one hand the language covers direct game mechanics and therefore game design considerations, it also aims to provide similar for other aspects of mobile mixed reality games, namely authoring, content creation, interfaces, orchestration as well as testing and logging. Eleven patterns based on these aspects are described in more detail.

## Keywords
Mobile Mixed Reality Games, Pervasive Games, Location-based Games, Design Patterns, Authoring, Orchestration

## 1. INTRODUCTION
Mobile Mixed Reality Games (MMRGs) combine the virtual world with the real environment. Players wander (or run) around the city while computing devices support the gaming experience. The advance of modern smartphones has finally made it possible to develop such games outside the realm of research and for the first time a large base of potential users can be reached to make such games economically feasible (at least in theory). This rather young strain of game development has however not yet formed a cohesive and structured understanding of what makes these games fun to play, how they function and what needs to be taken into consideration when designing, developing and staging them. Design pattern languages have been used in the past in other areas and since a few years design patterns for games have also been established. They aim to present knowledge about the inner workings of a topic and guidelines for better application in a structured way based on well-understood and recurring characteristics of said areas.

In this paper a proposal is made for a design pattern language specifically catering for the affordances of MMRGs. These games have certain peculiarities that make them arguably more complex to develop and maintain than traditional videogames (e.g. reliance on inaccurate sensor data, close coupling to the real world context they are played in). Such design patterns should therefore not be limited to only analyze game mechanics but further extend themselves into areas like authoring, content, user interfaces, software development, testing and run-time orchestration.

For a game to fall into the scope of this paper the following definition of MMRG is used: *A Mixed Reality Game is a game that is manipulated by actions in the real world as well as in the digital realm and/or is represented in both. In addition, in Mobile Mixed Reality Games players (or other game entities) perform (or witness) a meaningful change of context. This change is typically movement through physical space but could also be constituted by a changing environment (physical, temporal, social) around stationary players.*

Terms with similar definitions are Location-based Games (which require spatial change) [20] or Pervasive Games (which do not necessitate technology) [17]. All three terms are often used interchangeably, and the majority of games from each group arguably also belongs to the other two.

This paper does not present finished research but is rather an introductory position paper. It should serve as an attempt in grasping the scope of the proposed design pattern language, including applicability and feasibility for use with MMRGs and built the basis for a discussion around these topics.

After this introductory section, the paper continues with a section of related work concerning MMRGs and design patterns. Section three further motivates the introduction of design patterns for different aspects of MMRG development. Section four looks at use cases for such design patterns, followed by the definition of the pattern language. Section six describes sample patterns derived from previous work utilizing the pattern language. The paper finishes with a discussion section and an outlook into future work.

## 2. RELATED WORK
### 2.1 Mobile Mixed Reality Games
Mobile Mixed Reality Games have a lot in common with location-based games and pervasive games and these terms are often used interchangeably. In general MMRGs create a game world that exists both in the real (non-digital) world and in a digital representation. Early examples of MMRGs include *GeoCaching* [14] or first forays into mobile augmented reality (AR) like *ARQuake* [23] and *Human Pacman* [9]. Alternate reality games like *The Beast* [26] also fall into the category of MMRGs as do remotely-played experience as *Can You See Me Now?* [5]. Locationing technology utilizes a wide variety of sensors like GSM cells, GPS, fiducial markers, natural feature tracking, NFC/RFID as well as WiFi and Bluetooth-based proximity sensing, all with their respective advantages and disadvantages that have to be taken into account when creating and running these games. MMRGs often embrace the environment they are played in and make it a specific element of the game play (e.g. *REXplorer* [4]). Such close coupling with the real world on the one hand arguably creates a stronger engagement but at the same time make it difficult or impossible to transport them to another

city. Some games circumvent this problem by encouraging users to create their own content (e.g. *Tidy City* [27]). Orchestration often plays an important role, sometimes due to live-action roleplaying elements (*Interference* [16]) and sometimes out of a desire (or necessity) to react quickly and appropriately to player actions and the general flow of the game [10]. The advance of smartphones has seen a recent rise of such games and greater variety with them being commercially available and no longer restricted to research prototype or event-based games. *Mister X Mobile* [12] translates the classic board game into a modern game of tag, and Grey Area's *Shadow Cities* [15] and Google's *Ingress* [19] probably sport the biggest amount of players of MMRGs to date with a single persistent game area spanning the globe.

## 2.2 Design Patterns

Design patterns were first introduced for the use in architectural and city planning contexts [1, 2]. These design patterns ranged from large-scale ideas about how to develop towns and cities in a country to setting-up lively neighborhoods and all the way down to minute details of room construction. Together they formed a *pattern language* aimed at offering insights and guidelines into how to design for everyday use as well as offering a common ground and vocabulary for discussion. They follow a problem-solution approach where each pattern answers one common (or easily overseen) issue when creating architecture. *Lighting on Two Sides of Every Room* for example talks about the tendency of people to prefer rooms that let in a lot of light and suggests to always design rooms with windows on two different sides.

Since then the idea of design patterns has been adapted into other areas. Design patterns for software engineering [13] look at typical challenges one has to solve when programming and offers generic and language agnostic solutions to these (e.g. *Singleton* proposes a class that can only be instantiated once during the runtime of a program).

In game design (often focused on video game design) patterns have also been proposed [22] and also quite intensively covered [6]. In contrast to software and architecture patterns, these game design patterns follow less of a strict problem-solution approach but rather describe identified game mechanics, their uses, occurrences and consequences. Examples include *Power-Ups* (strengthening your character in certain ways as seen in Pac-Man or Super Mario) or *Analysis Paralysis* (which happens when players are overwhelmed by potential move options as in Chess). Related to this design pattern language is the Game Ontology Project (GOP) [29] whose clear goal is the analysis and identification of established game elements and mechanics without including judgment of how they should be applied best. Design patterns for MMRGs are so far underrepresented although there have been some explorations focused on mobile gaming in general [11]. Arguably, many existing design guidelines for MMRGs [17] [28] share similarities with design patterns and could be transformed and restructured accordingly. These guidelines also are not limited to pure game mechanics but also look at the supporting infrastructure and other design decisions.

# 3. PROPERTIES OF MMRGS

As Mobile Mixed Reality games consist partially in the digital realm, a lot of their development is similar to that of traditional video games. Content in various forms needs to be produced (e.g. graphics, sounds), the game mechanics have to be evaluated, balanced and implemented, playtests are important as well as tracking user-behavior in order to learn lessons for future productions. Due to their nature of including the real world, MMRGs however have characteristics that go beyond videogame development - which arguably makes the development process more challenging. Design patterns for MMRGs should cater for these properties, analyze and support them. The following is an attempt to identify these specifics, and also compare them to their videogame counterparts if applicable.

## 3.1 Game Mechanics

In a videogame, all user input can be easily detected and translated into actions in the game (e.g. using a controller as input device), and the whole game is completely under control of the developers. In a MMRG, real world locations play an important role. How do players travel from one place to another? What happens between places? A MMRG might also include rules that cannot be fully controlled by the game itself (e.g. prohibiting players travelling on the bus in a game of *Mister X Mobile*). Or perhaps players need to handle a physical object as part of the game (e.g. the doll in *Interference*). Furthermore, as players are their own avatars, precisely locating them becomes a challenge due to flawed sensor technology. These are just some simple examples where game mechanics derived from videogames are not sufficient to model a MMRG. Design patterns should look at the ones that are common for MMRGs (and might also exist for videogames) or specific for MMRGs (and have no direct counterpart in videogames).

## 3.2 Content Authoring

Due to having to use mobile devices, content for MMRGs is usually simpler than for videogames - something which is obvious when comparing augmented reality games to first person shooters for example and the different levels of visual fidelity. However, as the game world is not completely represented in a controllable virtual environment, using real world locations as part of the game becomes an extremely important aspect of MMRGs. Choosing the right place for a dramatic scene is not a trivial task as one needs to take into consideration among other things: closeness to other places in the game, accessibility of the place during play times, change of characteristics at different times of the day. The more specific a place in such a game is, the more difficult it is to find a corresponding one in another city where the game should also be played. Some MMRGs circumvent this problem by randomly placing content in the environment; others take archetypes of locations (like *Ingress* where all places are related to culture or education which could therefore be generated automatically from existing databases). Scouting out these places is a demanding task that cannot be done fully from ones office but requires exploration of the city itself.

## 3.3 Interfaces

It is fair for a videogame developer to assume that the players have precise interfaces for input (controller, mouse, keyboard, etc.), have perfect view of the screen, good sound equipment as well as a fast and reliable internet connection. None of this is true for MMRGs. Different sensors have different strengths and weaknesses that completely change the way a game might work. The screen is small and the interface is hard to see in sunlight. It might be cold and rainy, so players put their devices in their pockets. Traffic makes it impossible to hear any audio output without headphones (which in turn prohibit the player from fully listening to the environment). Mobile data connections can at any moment turn slow or fail altogether. These pitfalls are what make the design of a user interface for a MMRG a crucial but difficult part of their development.

## 3.4 Development

Testing any piece of software is an imminent part of its development. MMRGs add another level of complexity to it. While some testing can be done with e.g. the phone attached to a development computer, it is necessary to also test the game on location. Only this way it is possible to see how the sensors behave, how the selected places feel and to evaluate the physical movement required by players (e.g. distances). Dealing with this requires a lot of time as the necessary switch of location prevents quick fixes. While logging the user input that happens directly when interacting with the digital world (e.g. pressing a button) is trivial, tracking real world interactions is again way more complicated and often relies on inaccurate sensor data or visual observations of the players.

## 3.5 Run-Time Orchestration

Videogames rarely have somebody behind the scenes observing player behavior and reacting accordingly by advancing the story or triggering certain events in the game. In MMRGs however this is a not uncommon occurrence. Game masters need to react on (unforeseen) player actions, and change planned scenes or events. These activities are very comparable to game mastering pen and paper roleplaying games where the almost unlimited options players have make it impossible to have predicted (and prepared) everything beforehand. Acquiring information about the state of the players (both for example their location but also their level of excitement) becomes important for making such decisions in an MMRG.

# 4. THE SCOPE OF DESIGN PATTERNS

When looking at the previously discussed and established design patterns languages they clearly share common goals. They want to provide a language that enables people to talk about typical issues and observations in a standardized way. By using the same vocabulary collaboration between different people is meant to become easier. Furthermore, design patterns allow breaking things down into smaller concepts, to potentially better compare different entities and search for commonalities and differences. Patterns can also be used to foster creativity when creating something new, and ease planning and outlining of ideas. Last but not least, design patterns are about identifying problems, solving them and/or preventing them from happening in the first place. In summary, these are the core goals of typical pattern languages: *Communication* (discussion/collaboration), *Analysis* (representation/standardization), *Creativity* (outlining/planning), and *Improvement* (problem solving/prevention).

These aspects of course do not exist separately from each other. Knowing the vocabulary enables not only good communication but provides the basis for a thorough analysis. Being more aware of possibilities sparks creativity and helps identifying problems and intuitively avoiding them. It therefore only seems logical to require the same attributes from a pattern language for MMRGs. These games have not yet reached a widespread adoption, both from a player as well as commercial developer perspective. Technologically simplistic games like *GeoCaching* have a large amount of players and circumvent many of the described problems due to their simplicity and the relative ease with which users can create their own content. Commercial games which fully exploit the power that modern smartphones enable are still few and far between with *Shadow Cities* and *Ingress* perhaps the only two with a large player base. This lack of successful games is on the one hand certainly caused by the aforementioned complications concerning the development of MMRGs, but arguably also because the games are still relatively new and therefore knowledge is lacking about how best to design and develop these games, what pitfalls and common errors to avoid and how to best engage players. This is where a design pattern language not only covering the game mechanics but the whole production process can potentially be of great help. Such a language should foster *Communication* to bring designers and developers onto the same level of understanding, with a widened vocabulary to be aware of common and uncommon qualities of MMRGs. A proper *Analysis* of these qualities should be supported to properly examine and compare existing games. What properties do they have in common, what are their effects on the players? What ways of authoring content are utilized and what kind of tools are needed to make this approach feasible? With increased potential for *Creativity* it becomes easier to quickly outline a desired design while being aware what kind of ripple effects choosing a certain pattern has on other parts of the design. What characteristics are needed for the orchestration tools to create the best connection between game masters and players? How does the choice of sensor influence the game play experience? How can unusual technology best be employed? Lastly, a design pattern language for MMRGs should help *Improvement* of existing games and development environments or avoid problems from occurring in the first place. What kind of play test set-up is useful or even required for the type of MMRG that is being developed? As MMRGs are difficult to set-up, develop and stage there is big potential of improving the current situation with a functioning pattern language.

# 5. PATTERN LANGUAGE FORMAT

In order to define a pattern language for MMRGs it is once again necessary to compare established design pattern languages in order to identify similarities and differences in their structure.

## 5.1 Existing pattern languages

### *5.1.1 Architecture*

Having been the first design pattern language [1] [2], these original ideas shaped the appearance of all pattern language that came after it. All patterns in the language are numbered and thus form a *Hierarchy*: The lower the number of a pattern, the grander is its scale. The patterns itself have a short, memorable *Name* that serves as a reminder of what the pattern is about and is supported by a P*icture* showing an example implementation of the pattern. An introductory paragraph puts the pattern in *Context To "Larger" Patterns*, i.e. patterns that are higher up in the hierarchy. A *Headline* (only one or two sentences long) defines the problem that the pattern aims to solve. The whole situation is then explained in intensive detail in the B*ody Of The Problem* - consisting of the empirical background, proof for its validity, variations on the pattern etc. This is followed by the *Solution* (phrased as an instruction to the reader). A *Diagram* visualizes the solution further before another small paragraph mentions the *Context To "Smaller" Patterns* that are relevant for completion of the current pattern.

### *5.1.2 Software Engineering*

The software engineering pattern language first proposed by the "Gang of Four" [13] utilizes a more complex template for pattern description. Each pattern has a succinct *Name* and a *Classification*. The problem the pattern tries to solve is called *Intent* and is a short statement. *Also Known As* provides alternative names that the pattern might have. The *Motivation* describes the problem in more detail and how it can be solved as part of a sample scenario. The *Applicability* gives more

information about the scope and usage of the pattern in different contexts and is followed by a graphical representation of the pattern's *Structure* in different types of diagrams. *Participants* describes software classes and/or objects that are used as part of the pattern while *Collaborations* describes their interactions with each other. *Consequences* presents the advantages of using the pattern as well as potential disadvantages. *Implementation* describes the actual coding part in more detail and is followed by a piece of *Sample Code*. *Known Uses* gives at least two examples of where this pattern has been applied in a real system. The pattern is concluded with *Related Patterns* that affect the use of the current pattern, might be affected themselves and general commonalities or differences between them.

### *5.1.3 Game Design*

When looking at patterns concerning themselves with game design, an additional template has been established [6]. Eleven *Categories* have been identified, and each pattern belongs to exactly one of them. Each pattern has a *Name* describing the concept of the pattern. A *Core Definition* summarizes the essence of the pattern in a concise way. The *General Description* gives a more detailed (but still short) overview of the elements that make this pattern and is supported by some *Examples*. *Using The Pattern* talks about the choices a designer has when applying this pattern and its variations. *Consequences* describes the effects that using the pattern has on the gameplay. *Relations* lists all other patterns that are somehow connected to the current one (and have potentially already been mentioned as part of *Using The Pattern* or *Consequences*). These patterns are grouped into five different categories: *Instantiates*, *Modulates*, *Instantiated By*, *Modulated By*, and *Potentially Conflicting With*. The pattern finishes with *References* listing related work that has been used for creation of the pattern or contains further descriptions.

## 5.2 Comparison

All pattern languages have in common that the patterns shall be memorable and intuitively understood. This is achieved by choosing a fitting name supplemented by short summaries of either a pattern's core or its problem and solution. Additionally, all pattern languages categorize the patterns in different ways, and all use real world examples for illustration purposes. Furthermore they all put the pattern into context with other patterns and talk about their relations to each other. The detailed description of the pattern is one large body of text (architectural patterns) or split up in a few different sections (game design patterns) or a larger variety (software patterns). The biggest difference between the pattern languages is the fact that the game design patterns do not follow the problem-solution approach utilized by the others. Here, the other two languages are focused more on providing guidelines for users of the pattern language instead of patterns to (mainly) analyze different core elements of games and how they interact with each other.

## 5.3 A Pattern Language for MMRGs

After considering the components of other established pattern languages, the following structure is proposed as a pattern language for MMRGs: *Name*, *Categories*, *Problem*, *Solution*, *Examples*, *Description*, *Effects*, *Connections*. The problem-solution approach is the core idea of the original pattern language and should make it easier to know which pattern to apply when a problem is encountered. *Effects* are chosen to complement this approach as patterns can have repercussions that might not be directly related to the problem, and might also cause other problems. In order to increase general readability, this warrants its own section in the pattern language. In general all sections except *Description* are brief to make each pattern more accessible.

**Name:** The *Name* of a pattern has to be concise, unique and already give an initial idea of the content of the pattern.

**Categories:** In order to make it easier to find patterns of a certain type, they should be categorized. The main groups have already been identified in section 3. In contrast to the other pattern languages, it should however be possible to assign several different categories of different granularity to a single pattern. This should make it easier to find relevant patterns, especially when they are presented in digital form and can be reordered at will.

**Problem:** As MMRGs are still a not a very popular type of game, one main motivation for the use of patterns is their potential to help improve games being developed. This is attempted by stating a concrete *Problem* to describe briefly when the pattern should be applied. Some *Problems* might be more generic than others and appear again in similar patterns.

**Solution:** The *Solution* section of a pattern briefly describes the essence of a pattern and how it is solving the issue at hand in broad strokes.

**Examples:** Real world examples of where the pattern has already been applied are used to illustrate the pattern. There could also be negative examples describing how a specific MMRG is suffering from the current problem.

**Description:** The *Description* gives a thorough overview of the pattern and discusses it in-depth. It provides more detail on the *Problem* as well as the *Solution*, describes common pitfalls, provides justification for its validity and discusses variants of the pattern.

**Effects:** *Effects* are a short summary of the positive and negative results that the application of a pattern has or might have.

**Connections:** *Connections* put the current pattern in relation to others. Sometimes patterns might contradict each other or work especially well together.

# 6. SAMPLE PATTERNS

The following patterns are not fully proofed patterns but should serve as an example and prototypes on how to apply the pattern language across the different aspects of MMRGs. In contrast to game design patterns they are not focused on the actual game design but instead also cover tools, implementation details, technical obstacles, location selection, content preparation etc. as described in section 3. This is also the reason why the sample patterns should not be seen as stand-alone set of patterns. They rather serve as a proof-of-concept: showcasing that there is a big enough potential for design patterns for MMRGs that are not restricted to pure game design and mechanics. It should be noted that patterns from other collections not specifically aimed at MMRGs but covering games in general certainly are also very valid for MMRGs. In order not to reproduce already existing work, the MMRG pattern language should strive to identify new patterns that are especially relevant and/or unique for this genre while referencing more generic ones from other sources. After having done more research in the area of design patterns for MMRGs, more patterns will be identified and the existing ones will be adapted, perhaps split up or completely removed. Furthermore the patterns presented here have been edited for brevity and – due to a lack of a yet large canon – connections to other patterns have been omitted (as most of them would be to as of now unidentified patterns).

## 6.1 Enforced Speed Limit

**Categories**: Game Mechanics, Player Movement

**Problem**: Athletic players have unfair advantages

**Solution**: Player movement is tracked and they are penalized when running too often / too fast.

**Examples**: *aMazing* [3] gives every player an energy pool that depletes faster when moving faster. Players who are running can no longer perform any actions in the game and have to wait standing still for the pool to refresh.

**Description**: Many MMRGs give an edge to players who are physically fit. It is often advantageous to get as quickly as possible from one location to another, and sometimes other players need to be caught. While physical exhaustion is often conceived as fun, it can also negatively influence the enjoyment of players who do not want to be running around the whole time (or are handicapped due to being very young, older or just not athletic). If not carefully designed such games turn into races with always the same players winning. One way of overcoming this is to implement a speed limit. The movement of a player is tracked and a speed calculated. When players run too fast they can be penalized in a variety of ways. As GPS data is inherently inaccurate there should be proper checks if the players really just ran or if the GPS is just suffering from jitter.

**Effects**: Gives players with different fitness levels equal chances (positive). Limits physical engagement (negative?).

## 6.2 Voluntary Movement Restrictions

**Categories**: Game Mechanics, Player Movement

**Problem**: Not all movement rules can be controlled by software.

**Solution**: Players agree before the game only to walk (and not run), not use certain modes of transportation (e.g. busses, trams) or not leave a certain area.

**Examples**: In sessions of *Mister X Mobile* it is usually frowned upon to take busses to travel around the game area faster or to hide inside buildings (where GPS does not work).

**Description**: Not everything can be regulated by technology as there is often a way for players to overcome such obstacles. Instead of trying really hard to keep players from cheating, an easier way is to trust them and to have players negotiate a social contract before a game session. This also allows players with different tastes to make up their own rules of what constitutes cheating.

**Effects**: Gives players great flexibility in what they want to restrict (positive). Saves development time (positive). Players might cheat unintentionally, out of excitement or because of unclear boundaries (e.g. when trying to get away from someone by “almost-running”) (negative).

## 6.3 In-situ Authoring Tool

**Categories**: Content Authoring, In-situ

**Problem**: Close coupling of content requires real world scouting

**Solution**: Provide a tool that can be used to create new content for a MMRG while being on location.

**Examples**: When creating new missions and riddles for *Tidy City* users can do so right on the spot with a tool for Android smartphones. The interface of the tool guides them on what kind of data is still needed and allows them to take pictures, GPS data and notes that are then transferred into the web-based main authoring tool.

**Description**: In-situ authoring is a powerful mechanism for content creation as it enables the users to immediately add to the game while being inspired by their surroundings [25]. Doing this in-situ also gives you valuable feedback about atmosphere of a location, accessibility or unforeseen obstacles (e.g. construction work). Instead of letting every contributor use their own methods to collect content providing a dedicated tool with clear boundaries helps to structure the content already in an early phase and saves conversion time. It also enables less tech-savvy people (who on the other hand might have more knowledge about the city) to participate in the authoring process.

**Effects**: Empowers users to easily create new content in the correct format (positive). Lets content creators experience location first hand while instantly allowing them to productively use their inspiration (positive).

## 6.4 Automated Generation of Interesting Locations

**Categories**: Content Authoring

**Problem**: A game should be playable around the world while still taking local structure into account.

**Solution**: A game auto-generates the locations that play an important role in the game by the use of a clever algorithm or based on intensive data.

**Examples**: In *Ingress* players are fighting over portals. These portals appear at culturally important locations like libraries, museums or galleries. This enabled the developers to automatically add them to the game.

**Description**: Creating content for a game without randomly spawning items but with relations to the real world environment is not trivial. One way is to create and place everything manually, but this is not feasible for larger games that are not limited to a single city. An alternative is to analyze existing data and create locations for the game out of this. This could mean that one looks at different categories of locations (like restaurants, bars, universities, bus stops) that can be used to populate the game world.

**Effects**: Easy to create a huge game world (positive). Not all data used for generation might be accurate or up-to-date (negative).

## 6.5 User-created Missions

**Categories**: Content Authoring, User Participation

**Problem**: The game needs very specific content for each location it should be staged at.

**Solution**: Players can create and add their own missions

**Examples**: *Tidy City* enables every interested player to create their own set of riddles that are referencing the real environment. Similarly *SCVNGR* [21] lets users add to the overall content of the game by adding single tasks or complete routes that others can follow. *GeoCaching* makes it equally easy for every player to hide their own Cache and publish information about it on one of the various community websites.

**Description**: One strength of MMRGs is the close coupling of digital content with the real world environment. While this adds a lot to the enjoyment of the game, it also makes producing new content tedious and impractical. New high quality content needs to be produced for every city (or even neighborhood) the game should be played in. With easy-to-use authoring tools that do not overwhelm inexperienced users, they can become authors of the game themselves. With their expertise and in-depth local

knowledge they become important contributors and enrich the game world.

**Effects**: User-created missions allow MMRGs to grow with the help of motivated users (positive). Not all user content might be high quality (negative). If a game only relies on user-created missions, it is difficult for new players get into the game in the first place as content might be lacking (negative).

## 6.6 Large-scale Augmented Reality

**Categories**: Content Authoring, Interfaces, Augmented Reality

**Problem**: Augmented Reality needs to be used effectively.

**Solution**: AR content should be several meters in size.

**Examples**: In *TimeWarp* players encounter a Roman Arch that is 15m tall and makes for an impressive sight.

**Description**: The use of Augmented Reality (AR) always needs to be justified. What is the advantage of using it over 2D graphics, illustrations or presenting it without overlaying it to the real world? One way to increase engagement with AR is to use large-scale models like towers, houses, statues etc. Players can already see them from far away, so they can act as guidance. When closer, players have to actually physically walk around the objects to fully explore them and also lean far back to see the top of the object. This adds physical engagement to their experience and gives them a much better feeling for the object in question than without AR [7].

**Effects**: Large-scale AR objects are more engaging for players (positive). Inaccurate tracking diminishes the effect as large-scale objects should typically be very stable (negative).

## 6.7 Audio Replay

**Categories**: Interfaces, Audio, Usability, Noise

**Problem**: Street noise makes it difficult to understand audio.

**Solution**: Players can replay received audio messages when and how often they want (or at least more than once).

**Examples**: In *Castle Crisis* [8] players get pre-recorded phone calls from certain characters that give them clues in order to find hidden bombs. The players can always call back in case they need to hear the clue again.

**Description**: MMRGs that are played in city streets often run into problems with audio. Street noise overwhelms speakers built into the phone, and the use of headphones can easily separate the players from their environment by shutting out these noises (which are often necessary to create a true mixed reality experience and are also helpful from a safety point of view). Allowing players to replay an audio message that they have received solves this problem as repeated replay assures players understand everything. Furthermore it makes it easier to share a message with co-players as they can listen to it at their own convenience.

**Effects**: Mitigates street noise and other problems for understanding audio (positive). Might lead to players listening to the same messages over and over again (or replaying them to all other players) slowing down the game (negative).

## 6.8 Audio as Main Media

**Categories**: Interfaces, Audio

**Problem**: Focus on visual interface limits real world engagement.

**Solution**: Using audio as the main means to provide information.

**Examples**: The only visual elements in the game *Castle Crisis* are two buttons for calling people. The rest of the game content consists of audio. In *Cargo* [18] the players receive phone calls with instructions by a computer-generated voice.

**Description**: MMRGs want to combine the real world with the digital one. Many games forget about the real world in this context and thus offer an overwhelming amount of visual information to the players. They track their position on the map, have to read on-screen texts and constantly press different buttons. Naturally this obstructs their engagement with the real environment as their focus always lies on the screen of the smartphone. This stops players from actively observing their surroundings and even endangers them when doing so while crossing streets. If the game uses a greater amount of audio, the players can focus more on the outside world.

**Effects**: Allows players to focus on real world while experiencing game content at the same time (positive). Audio might be difficult to hear due to noise (negative). Audio requires more effort to produce than simple text (negative).

## 6.9 Simulated GPS Jitter

**Categories**: Development, Testing, Sensors, GPS

**Problem**: Flaws of GPS can only be experienced when doing tests away from the development environment (i.e. outside).

**Solution**: When simulating GPS data for testing purposes the fluctuating accuracy of the signal is also simulated.

**Examples**: [No known examples of applying this technique.]

**Description**: While developing an MMRG based on GPS it has to be tested if proximity triggers are activated as expected and for example trigger zones do not overlap in a way that causes problems. When simulating player movement in a desktop environment the GPS accuracy is usually perfect, something that is not the case in the real environment. Problems caused by this can therefore only be found by in-situ testing - which is costly and time consuming. Therefore when using KML data or other means of simulating GPS, the development environment should support setting a randomized and changing accuracy to all data and thus having the simulated players not move in a straight line but rather having their positions "jump around". This is important in order to see how the game will react when e.g. proximity triggers are so close to each other that bad GPS accuracy causes false positives.

**Effects**: Easier discovering of how the MMRG behaves under "real" GPS conditions without having to go on location (positive).

## 6.10 Live Player Tracking

**Categories**: Runtime Orchestration, Live Data

**Problem**: Game masters are unaware of the player's whereabouts.

**Solution**: Broadcasting position of the players to game masters.

**Example**: The player group in *Interference* was equipped with a GPS sensor. The current position was constantly sent to the orchestration system, so that game masters were always aware of the physical location of the players.

**Description**: When orchestrating a MMRG it is important to have information about the state that the players are in. One of the crucial aspects is their physical location as this typically triggers story elements or enables them to interact with digital game content. A map overview that always shows the last known position of the players is therefore extremely helpful for game masters to e.g. interpret the flow of the game. They could trigger events if the players are standing still for a long time (and thus

might be engaged in overly long discussions due to confusion or boredom) or redirect players towards more exciting playing areas if they seem to be lost or wander off in the wrong direction. Tracking the player position works best when the game uses GPS as cell-id and WiFi locationing offer rather imprecise location data. In games built on relative location (e.g. the players interact with the game by scanning NFC tags) players can only be located at these "action points".

**Effects**: Game masters are aware of the physical whereabouts of the players (positive). Requires a working data connection and constant location updates draining the battery (negative).

## 6.11 Ingame Tech Support

**Categories**: Runtime Orchestration, 360 Illusion, Roleplaying

**Problem**: Fixing technical problems can limit player immersion.

**Solution**: Technical problems are considered part of the game and solved by an in character tech team.

**Examples**: When struggling with technical bugs in *Interference,* players could call headquarters and talk to engineers. These stayed in character the whole time while fixing the problems (which were blamed on using cutting edge technology).

**Description**: Technical problems happen all the time in MMRGs and they are not necessarily solvable by the players themselves. Instead of taking players out of the game by acknowledging these bugs out of character, players and support team stay in character while doing the debugging and fixing of the device. This can also be used to teach the characters on how to operate the devices in the first place and thus supporting a 360 illusion [24]. This pattern is best used in games with role-playing elements.

**Effects**: Despite technical failures, players stay immersed in the game (positive). Not all problems might be solvable by ingame technicians (negative).

# 7. DISCUSSION

The described patterns cover a wide range of different aspects of Mobile Mixed Reality Games and are based on observations from previous work. At the moment they are not connected to other patterns but exist only separately (although one could easily draw connections between the different ones dealing with content creation or player movement for example). This is unsurprising as MMRGs are on the one hand extremely diverse and on the other hand still underrepresented concerning the sheer amount of published games. In general this will pose a problem to deal with in analyzing existing games and tools for additional patterns: one important aspect of patterns is usually *recurrence*. In order for something to qualify as a pattern, it has to have been applied in several examples already. Otherwise one might argue that it does not constitute a real pattern.

Due to the nature of MMRGs this is a difficult problem to solve. It might work when looking at pure game mechanics as these can be easily observed by playing those games, watching other people play these games and reading reports about these games and thus deconstructing them into their atomic parts. This would result in a set of *established patterns*.

It gets more complicated about aspects like content, authoring or runtime orchestration. Here, even less examples are accessible. One possible way is of course to perform expert interviews with designers and developers and inquire about the tools that they are using. The question remains however if patterns for MMRGs should only look at already established and widely used patterns (which have just not been written down in a structured way) or whether it is also a valid approach to look for unique occurrences that have the potential to evolve into patterns if more MMRGs are developed utilizing them. For this to happen, however, these *emerging patterns* need to be collected and published nonetheless, despite their not yet validated state.

There is however another group of patterns that is interesting to be investigated: methods that have not yet been used at all. An example from this paper is *Simulated GPS Jitter* that would potentialy help immensely in the development process of MMRGs but seems to have not been adopted by anyone (probably due to a lack of native support for it by the popular development frameworks for smartphones). Here, again, research based on thoughts, opinions and wishes by current designers and developers could serve as a basis for these *hidden patterns*. It would then be a task to identify these patterns and construct prototypes that are applying them to prove their initial validity before they would then – hopefully – be taken up by more developers of MMRGs.

While *hidden patterns* are easily identified belonging inside the group, the line between *established patterns* and *emerging patterns* is hard to draw. When should a pattern be seen as fully accepted and widely used? If it appears in a certain amount of games? More than 10? More than 50? When exactly is a pattern validated? This will be a task for further thoughts and more strict definitions but can probably only be solved when looking at a larger amount of games and patterns than presented in this paper. For now, the discussed patterns might be sorted into the groups as follows:

**Established patterns:** *Voluntary Movement Restrictions*, *In-situ Authoring Tool*, *Automated Generation of Interesting Locations*, *User-created Missions*

**Emerging patterns:** *Enforced Speed Limit*, *Large-scale Augmented Reality*, *Audio Replay*, *Audio as Main Media*, *Live Player Tracking*, *Ingame Tech Support*

**Hidden patterns:** *Simulated GPS Jitter*

# 8. CONCLUSION AND FUTURE WORK

The presented framework constitutes a first attempt at a design pattern language for Mobile Mixed Reality Games. It is not limited to game mechanics but also looks into supporting tools, methods and infrastructures that have an important influence on designing, developing and staging MMRGs.

The next steps in the work will be to further frame the exact boundaries of the language and to continue the identification and collection of suitable design patterns. This will be done in part by looking at existing work (e.g. guidelines, evaluations) and in part by analyzing existing scientific and commercially available games and tools to filter out existing atomic actions, concepts and approaches. Another way of distilling these patterns will be interviews with experienced designers and developers of MMRGs. Furthermore, the question of when something can be called a pattern needs to be answered more thoroughly, making the distinctions between *emergent patterns*, *established patterns* and *hidden patterns* more prominent and well defined. A big challenge is posed by the overall limited amount of existing games. The importance and exact definition of recurrence as a core concept of patterns needs to be thoroughly analyzed. As a remedy, tools and games based around not widely spread patterns should be developed in order to test and validate them as well as making them more commonly known.

One of the most pressing questions for every design pattern language also needs answering: How best to apply the identified patterns? What kind of qualities do they need to posses so that people actively start using them and they provide an added value to their work? How do they fit into existing development workflows for creating MMRGs? Here, an application in an educational setting seems like a suitable first step to gather initial feedback as it is often the most accessible one when performing research. Rather sooner than later, however, the pattern language needs to be evaluated by in non-academic contexts. This is important so that it becomes more than just a "theoretical exercise" and instead proofs its overall value.

## 9. ACKNOWLEDGMENTS

This work is supported by EPSRC grant EP/I011587/1 as part of the ORCHID project. In addition, the author would also like to thank the members of previous research projects IPerG, IPCity and TOTEM.

## 10. REFERENCES

[1] Alexander, C. 1979. *The Timeless Way of Building*. Oxford University Press.

[2] Alexander, C., Ishikawa, S., Silverstein, M., Jacobsen, M., Fiksdahl-King, I. and Angel, S. 1977. *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press.

[3] aMazing: 2012. *http://phenix.int-evry.fr/totem/?q=aMazing*. Accessed: 2013-03-20.

[4] Ballagas, R.A., Kratz, S.G., Borchers, J., Yu, E., Walz, S.P., Fuhr, C.O., Hovestadt, L. and Tann, M. 2007. REXplorer: a mobile, pervasive spell-casting game for tourists. *CHI '07 Extended Abstracts on Human Factors in Computing Systems* (New York, NY, USA, 2007), 1929–1934.

[5] Benford, S., Crabtree, A., Flintham, M., Drozd, A., Anastasi, R., Paxton, M., Tandavanitj, N., Adams, M. and Row-Farr, J. 2006. Can you see me now? *ACM Trans. Comput.-Hum. Interact.* 13, 1 (Mar. 2006), 100–133.

[6] Björk, S. and Holopainen, J. 2005. *Patterns In Game Design*. Cengage Learning.

[7] Blum, L., Wetzel, R., McCall, R., Oppermann, L. and Broll, W. 2012. The final TimeWarp: using form and content to support player experience and presence when designing location-aware mobile augmented reality games. *Proceedings of the Designing Interactive Systems Conference* (New York, NY, USA, 2012), 711–720.

[8] Castle Crisis: 2012. *http://phenix.int-evry.fr/totem/?q=node/156*. Accessed: 2013-03-20.

[9] Cheok, A.D., Fong, S.W., Goh, K.H., Yang, X., Liu, W. and Farzbiz, F. 2003. Human Pacman: a sensing-based mobile entertainment system with ubiquitous computing and tangible interaction. *Proceedings of the 2nd workshop on Network and system support for games* (New York, NY, USA, 2003), 106–117.

[10] Crabtree, A., Benford, S., Rodden, T., Greenhalgh, C., Flintham, M., Anastasi, R., Drozd, A., Adams, M., Row-Farr, J., Tandavanitj, N. and Steed, A. 2004. Orchestrating a mixed reality game "on the ground". *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2004), 391–398.

[11] Davidsson, O., Peitz, J. and Björk, S. 2004. *Game Design Patterns for Mobile Games*.

[12] Gamesload 2010. *Mister X Mobile*.

[13] Gamma, E., Helm, R., Johnson, R. and Vlissides, J. 1994. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.

[14] Geocaching - The Official Global GPS Cache Hunt Site: *http://www.geocaching.com/*. Accessed: 2013-03-20.

[15] Grey Area 2010. *Shadow Cities*.

[16] Jonsson, S. and Waern, A. 2008. The art of game-mastering pervasive games. *Proceedings of the 2008 International Conference on Advances in Computer Entertainment Technology* (New York, NY, USA, 2008), 224–231.

[17] Montola, M., Stenros, J. and Waern, A. 2009. *Pervasive Games: Theory and Design*. CRC Press.

[18] Moran, S., Pantidi, N., Bachour, K., Fischer, J.E., Flintham, M., Rodden, T., Evans, S. and Johnson, S. 2013. Team reactions to voiced agent instructions in a pervasive game. *Proceedings of the 2013 international conference on Intelligent user interfaces* (New York, NY, USA, 2013), 371–382.

[19] NianticLabs@Google 2012. *Ingress*.

[20] Nicklas, D., Pfisterer, C. and Mitschang, B. 2001. Towards location-based games. *Proceedings of the International Conference on Applications and Development of Computer Games in the 21st Century: ADCOG* (2001), 61–67.

[21] SCVNGR 2008. *SCVNGR*.

[22] The Case For Game Design Patterns: 2002. *http://www.gamasutra.com/view/feature/4261/the_case_for_game_design_patterns.php*. Accessed: 2013-02-01.

[23] Thomas, B., Close, B., Donoghue, J., Squires, J., De Bondi, P., Morris, M. and Piekarski, W. Oct. ARQuake: an outdoor/indoor augmented reality first person application. *The Fourth International Symposium on Wearable Computers* (Oct.), 139–146.

[24] Waern, A., Montola, M. and Stenros, J. 2009. The three-sixty illusion: designing for immersion in pervasive games. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2009), 1549–1558.

[25] Weal, M.J., Hornecker, E., Cruickshank, D.G., Michaelides, D.T., Millard, D.E., Halloran, J., De Roure, D.C. and Fitzpatrick, G. 2006. Requirements for in-situ authoring of location based experiences. *Proceedings of the 8th conference on Human-computer interaction with mobile devices and services* (New York, NY, USA, 2006), 121–128.

[26] Weisman, J., Lee, E. and Stewart, S. 2001. *The Beast*. Microsoft.

[27] Wetzel, R., Blum, L., Feng, F., Oppermann, L. and Straeubig, M. 2011. Tidy City: A Location-based Game for City Exploration Based on Usercreated Content. *Mensch & Computer 2011*. M. Eibl, ed. Oldenbourg Wissenschaftsverlag GmbH. 487–496.

[28] Wetzel, R., McCall, R., Braun, A.-K. and Broll, W. 2008. Guidelines for designing augmented reality games. *Proceedings of the 2008 Conference on Future Play: Research, Play, Share* (New York, NY, USA, 2008), 173–180.

[29] Zagal, J.P., Mateas, M., Fernández-vara, C., Hochhalter, B. and Lichti, N. 2005. Towards an Ontological Language for Game Analysis. *in Proceedings of International DiGRA Conference* (2005), 3–14.