RWA - A Game Engine for Real World Audio Games

Thomas Resch
Research and Development, School of Music
University of Applied Sciences Northwestern Switzerland
Audio Communication Group, TU Berlin
thomas.resch@fhnw.ch

ABSTRACT
Audio guides and (interactive) sound walks have existed for decades. Even smartphone games taking place in the real world are no longer a novelty. But due to the lack of a sufficient middleware which fulfills the requirements for creating this software genre, artists, game developers and institutions such as museums are forced to implement rather similar functionality over and over again. This paper describes the basic principles of Real World Audio (RWA), an extendable audio game engine for targeting smartphone operating systems, which rolls out all functionality for the generation of the above-mentioned software genres. It combines the ability for building location-based audio walks and -guides with the components necessary for game development. Using either the smartphone sensors or an external sensor board for head tracking and gesture recognition, RWA allows developers to create audio walks, audio adventures and audio role playing games (RPG) outside in the real world.

Keywords
Sound walk, audio guide, state machine, game engine, LibPd, Pd, binaural, positioning system, sensor fusion, head tracking, speech recognition, HRTF, augmented reality, OpenAL

ACM Classification
H.5.1 [Information Interfaces and Presentation] Mutlitmedia Information Systems, H.5.2 [Information Interfaces and Presentation] User interfaces, K.8.0 [General] Games.

1. INTRODUCTION
At Nime 2014 we introduced a server-based architecture for creating audio walks with simple gaming elements [1]. Server and corresponding Android app are the basis for the RWA engine. RWA consist of three different programs: The RWA-Creator, the RWA-Runtime for iOS and Android (and possibly for Windows Phone), and the RWA-Server. For the time being, the focus lies on the development of the RWA-Creator and the Android runtime. The Creator is a graphical user interface which allows the user to program the basis of an RWA game, mostly through drag and drop. More complex functionality than utilized by a simple audiowalk might need some editing using RWA's scripting language. Similar to other game engines, it has different views for data creation and manipulation. The aim is to provide an environment that enables the developer to combine GPS or other location data like Bluetooth or RFID tags with an event, where the term event might refer to almost anything: Starting up game logic, changing the current game state, sending messages to the server or another client, loading a Pure Data (Pd) patch on the smartphone or simply playing an audio file.

The examples in this paper are based on a pen and paper RPG fight sequence. While this is a rather unlikely scenario for an audio game, it is the most complex case for applying the rule set and therefore includes every other game situation.

2. RELATED WORK
During recent years, many commercial and non-commercial tools for creating sound walks have been published, for example [2, 3, 4, 5]. Both walks and guides may be created easily with these tools through a graphical interface by combining GPS data with audio or video files, but they all lack the functionality for implementing game logic. A related approach to the RWA engine without a follow-up realisation was proposed in [6] by Timothy Roden and Ian Parberry. The ideas described in their paper are close to RWA, however they do not incoorporate location based interaction. A good resource for audio games – although still in front of a computer display – can be found in [7]. Examples for already realized games with a real world character are for example Google's Ingress [8] and Zombies, Run! [9]. While Ingress takes place in the real world but is not an audio game, Zombies, Run! combines position data and binaural audio content into a post apocalyptic real world jogging game. The recently published Blowback [10] also includes binaural audio but lacks the location based interaction.

Although they usually do not integrate any wearable technical devices, alternate reality games (ARG) must be considered a close relative to RWA games as they also use the real world as their setting. A comprehensive introduction for ARG including some of its most successful representatives has been published in [11]. A very detailed description of augmented audio reality including several use cases using HRTF's and head tracking is provided in [12].

3. OVERVIEW
3.1 Workflow
An RWA game is composed of at least one RWA scene which is automatically generated when the Creator application is started up. A scene is a data structure which represents a state machine with an arbitrary number of states and transitions, and is either GPS based, settled in a user-defined coordinate system, or not relying on any coordinate system at all. A simple city guide could be described by several states within one GPS scene. State transitions would only rely on the client's location (meaning the smartphone's location) and be completely independent from the client's former state. Using the Asset-View, audio files or Pd patches for a certain state can be added via drag and drop. If a state is not edited any further by using

the State-Editor, all assets will be loaded and executed automatically in the runtime on arriving within the radius of the state's GPS location and stopped and removed on leaving the corresponding area. By default, audio content is rendered as a regular mono/stereo source. Selecting *Export* from the main menu saves the game script and allows distribution to clients and server. This can be done manually or – at a later point in RWA development – automatically, provided server and clients are reachable via the network.

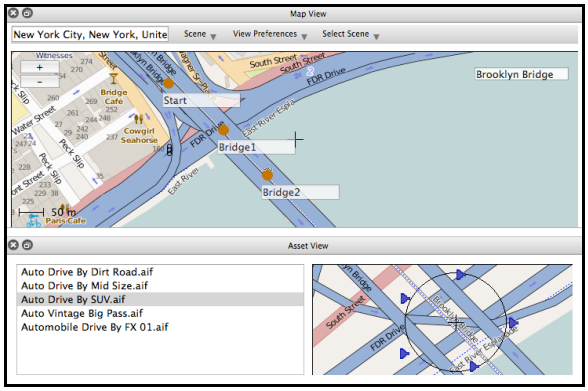

Figure 1. Screenshot of the Map- and Asset-View

3.2 Underlying Game System

The engine implements role playing game concepts and rules. RPG systems have been sufficiently analysed in [13]. Additionally, RWA introduces the concept of the entity which is either a subject – a hero or Non-player character (NPC) – or an object. Entities may have an arbitrary number of predefined attributes, they can influence attributes of other entities, and are capable of performing specific tasks at certain times with these attributes. They are also capable of owning other objects and subjects. Every entity has at least a name attribute.

A comprehensive task analysis for electronically enhanced board games has been done in [14]. Its results can be directly applied to RPG tasks by simply replacing the instructions in a board game – for example "Go to Jail" – with the RPG concepts of the game master and NPC's.

4. RWA-CREATOR

4.1 Requirements

The Creator application is written in C++ using the Qt framework [15]. Binaries for OS X and Windows are available at [16]. Linux, iOS and Android are also viable targets, and corresponding binaries will be published at a later point, as will the source code. Until then, RWA needs a Qt installation due to the mandatory dynamic linking required by the LGLP license. For the time being, the Max/MSP object fhnw.state [17] is used for simulation purposes. In order to test RWA games in this environment, game scripts have to be exported in the RWA-Script format. For the Android runtime the XML format is required.

4.2 Usage

4.2.1 Basic Setup, Managing Scenes

On startup, RWA automatically creates a default set of views and editors – The Map-View, the Asset-View, the State-Editor – and the first scene. More scenes can be added by choosing *New Scene* or *Duplicate Current* from the scene menu, and deleted by choosing *Delete Current*. For more elaborate scenarios, it might be necessary to use the Manifest and the Scene-Editor which allow for the use of global variables and functions.

4.2.2 Map View

Available tools for editing and manipulating scenes within the Map-View are arrow and rubber, implementing the same functionality commonly used in other applications: Using the arrow tool, GPS states can be generated by clicking into the map, or moved by dragging an already existing state. The rubber deletes states. Entryconditions, messages and connected assets are editable in Asset-View and State-Editor, which usually display the last touched state.

4.2.3 Asset-View

The Asset-View can connect certain file types through drag and drop with the currently edited state. For the moment, supported file-types are .wav and .aif audio-files and Pd patches. Support for other audio and video formats will follow. The map to the right of the asset file-list as shown in Figure 1. allows the user to put the currently selected source in a particular position. When a participant arrives within the radius of a state, the localisation of the sound source should remain stable at the chosen place(s) using binaural encoding – independent from the orientation of the participant´s head, and the participant's distance from the audio source. On entering a state, the default behavior for assets is to trigger playback or load a Pd patch. In order to modify this behavior and bind the execution of assets to other conditions, the state script has to be edited by using the State-Editor.

4.2.4 State-Editor

The State-Editor is a text based code editor and describes the behavior of assets and possible tasks of the hero, and interactions with other entities visible in the state. Currently, RWA-Script is the only supported language for game logic, supplying a limited instruction set with the ability to replicate common RPG rules.

4.3 Implementation Details

Thanks to Kai Winter's qmapcontrol [18], RWA integrates Google Maps as well as Open Maps support for the Map- and Asset-View. Using Qt's signals and slots, the application architecture is based upon a Model-View-Controller Pattern – with a mirrored set of data for rendering – realizing a communication scheme as shown in the figure below.

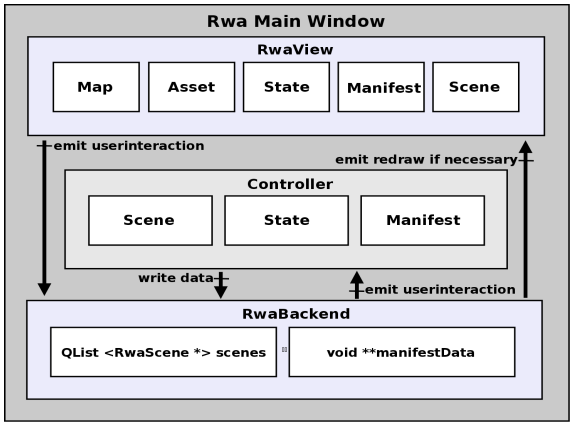

Figure 2. Communication diagram for the RWA-Creator

The base class for all views and editors is the RwaView-Class. On calling its constructor, a pointer to the backend is passed in order to interconnect user interaction signals with the corresponding slots in the backend. Whenever data is modified in any view, a signal containing the changes will be emitted. The backend then sends a signal to the appropriate controller, which writes the data into the model, notifies all currently

visible views, and forces them to redraw if necessary. Due to Qt's widget implementations which already contain a model, data is also directly updated internally within the widget. However, this data is only a mirrored version of the currently visible state or scene, and is merely used for rendering. Both scenes and states use Qt's QList data structure, internally represented by an array of pointers. This allows for fast random access and therefore equally fast jumps to any state in the game.

5. RWA-RUNTIME

5.1 Setup

The Android RWA-Runtime prototype is based on the Android FHNW AudioWalk app, which has been programmed for the Indoortracking project. For details on the underlying architecture, please refer to [1]. In contrast to the original application, the RWA-Runtime is self-aware of its own current state, and therefore has to load the game script into memory in order to be server-independent. On startup the RWA-Runtime creates its own entity by looking up the instructions described in the Manifest, and (usually) sets its state to the beginning of the first scene. In the easiest case, the hero's only attribute is a name. In a more complicated scenario, she might have a set of attributes, for instance strength, intelligence, or health points, and is already in possession of several objects.

```
@hero
{
   @name=entername();
   @strength=d6()+7;
   @attack=11;
   @defense=9;
   @health=30;
   @currentweapon=Sword;
}
```

```
@object
{
   @name=Sword;
   @eval() = return d6()+4;
}
```

Figure 3. A Manifest describing a hero and one object

5.2 State Transitions

A single state is capable of holding a set of instructions and messages in the @onentry-block and another set within the @pathway-block. This allows for the representation of a typical RPG cycle, using a minimal number of states, for example a fight sequence as shown in the diagram below. Scene variables and functions on the top and the first state are represented in RWA-syntax.

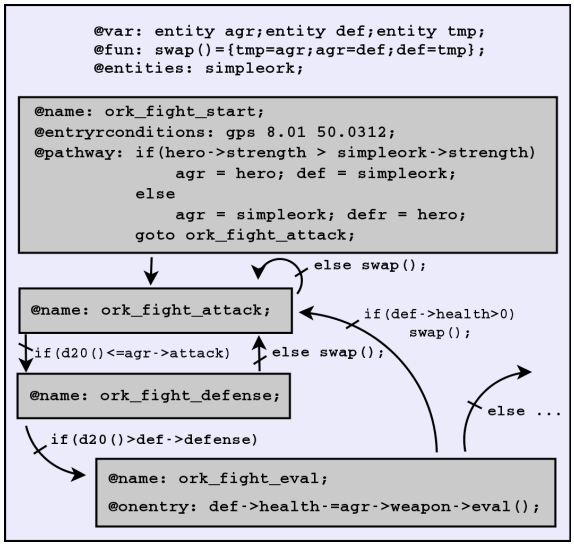

Figure 4. Fight sequence in RWA

The scheduler and executing interpreter for evaluating state transitions, tasks, and interactions works with a rate of 10 Hz which is completely sufficient for dialogue-based game situations, as well as for location-based state transitions. Explicit state transitions can be achieved by using the *goto* statement and are done instantly. They also allow for designing a well ordered game hierachy by interconnecting states and scenes as illustrated in the following figure, where the fight sequence is now completly isolated in its own scene.

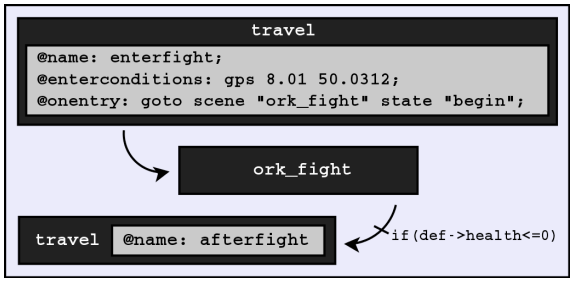

Figure 5. Isolated RWA scene "ork_fight"

6. ADVANCED FUNCTIONALITY

In order to set up scenarios as described in the previous chapter, it might be useful to predefine not only entities but also reappearing functions, for example for the fight sequence. This can be done either in the Scene-Editor for scene-specific actions as layed out in Figure 4., or in the Manifest, which provides global access of functions and variables from every state in the game.

```
@fun:fight(entity agr, entity def)
{
   if(d20() <= agr->attack)
   {
      if(d20() <= def->defense)
      {
          swap(agr, def);
          fight(agr, def);
      }
      else evalHit(agr, def);
   }
}
@fun:evalHit(entity agr, entity def)
{
   def->health -= agr->currentweapon->eval();
   if(def->health > 0)
   {
      swap(agr, def);
      fight(agr, def);
   }
}
```

Figure 6. Functions in RWA-Script in the Manifest

7. RESULTS

So far, only a few walks have been created and tested within the provisional Max/MSP simulation or as a single user game without any client-server interaction or binaural audio. But even without applying all the possibilities provided by an RPG rule set and other (not yet implemented) technologies, the existing system enables the user to program more complex apps than do the existing tools mentioned above. The time it takes to produce content not included, applications may be generated within minutes. The possibility of transferring an existing scene to a different location with only a modicum of additional editing – a sound walk created originally for Berlin's Tiergarten will need some adjustments for New York's Central Park – enables the developer to create and ship applications with the same sound material and functionality for a variety of locations at once.

LibPd is stable and runs reliably on iOS and Android, and makes it possible to integrate signal processing without having to alter the actual audio callback.

First tests have been done with calculating binaural sources on a Smartphone using the OpenAL Mob Library [19] and the Pd-Object earplug~ [20]. Depending on the library used and the additionally needed reverb for improving the distance simulation, an iPhone 4s was capable of computing three sources simultaneously. Although this number does not seem much at first glance, benchmarks like the Geekbench [21] suggest, that the currently available generation of smartphone processors is at least three or four times faster.

8. CONCLUSION

The success of games like Ingress, Run, Zombies! and Blowback, proves, that there is a definite need for a middleware like RWA. Through the usage of an RPG rule system it is possible to program a wide variety of applications, ranging from position based audio information systems, to audio adventure games, to language training applications, to complex audio-based role playing games. The resulting software can react to differences in daytime, weather, gender, or the participant's age, as well as to fictional attributes, objects, and NPC's, and therefore allow users to generate a completely dynamic audio augmented reality. The code editors for describing state- and scene-behavior are already easy enough to use – in future development, specialized editors and templates for certain situations will be included in order to allow for even more rapid game development. In combination with the already existing, very intuitive graphical interfaces, artists and game designers will be able to create RWA games by simply dragging and dropping – without the need for advanced programming skills.

9. ACKNOWLEDGMENTS

Thanks to Dr. Michael Kunkel, Prof. Dr. Stefan Weinzierl, and the Electronic Studio Basel for their support of my research.

10. REFERENCES

[1] Matthias Krebs, Thomas Resch (2014): "A simple Architechture for Server based (Indoor) Audio Walks". In: *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME 2014). London.* P.269-272.

[2] Authentic Tours Limited (n.y.), *myTours* [online]. URL: http://www.mytoursapp.com [accessed January 21st 2014].

[3] Espro Acousticguide Group (n.y.), *acousticguide* [online]. URL: http://www.acoustiguide.com [accessed January 21st 2015].

[4] Toozla (n.y.), *toozla* [online]. URL: http://www.toozla.com/ [accessed January 21st 2015].

[5] audioguideMe (n.y.), *Audioguideme* [online]. URL: http://www.audioguide.me/ [accessed January 21st 2015].

[6] Ian Parberry, Timothy Roden (2005): "Designing a Narrative-Based Audio Only 3D Game Engine". In: *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology (ACE'05)*. New York: ACM P. 274-277.

[7] Richard van Tol, Sander Huiberts (n. y.): *AudioGames, your resource for audiogames, games for the blind, games for visually impaired!* [online]. URL: http://www.audiogames.net [accessed January 22nd 2015].

[8] Google Inc. (n. y.), *Ingress* [online]. URL: https://www.ingress.com/ [accessed January 21st 2015].

[9] Six to Start with Naomi Alderman (n.y.), *Zombies, Run!* [online]. URL: https://www.zombiesrungame.com/ [accessed January 21st 2015].

[10] Deutschland Radio (2015), *Blowback* [online]. URL: http://blogs.deutschlandradiokultur.de/hoergame/ [accessed January 21st 2015].

[11] Jeffrey Kim, Elan Lee, Timothy Thomas, Caroline Dombrowsky (2009): "Storytelling in new media: The case of alternate reality games, 2001-2009" [online]. URL: http://firstmonday.org/ojs/index.php/fm/article/view/2484/2199 [accessed April 15 2015]. In: First Monday, Volume 14, Number 6.

[12] Aki härmä, Julia Jakka, Miikka Tikander, Matti Karjalainen, Tapio Lokki, Jarmo Hiipakka, Gaëtan Lorho (2004): "Augmented Reality Audio for Mobile and Wearable Appliances". In: *Journal of the Audio Engineering Society 52 (6)*. P 618-639.

[13] Anders Tychsen (2006), "Role Playing Games - Comparative Analysis Across Two Media Platforms". In: Wong, Fung, Cole, Pisan (Edt.): *Proceedings of Third Australasian Conference on Interactive Entertainment (IE 2006)*. Perth. P.75-82

[14] Daniel Eriksson, Johan Peitz, Staffan Björk (2005): "Enhancing Board Games with Electronics". In: Gellersen, Want, Schmidt (Edt.): *Proceedings Series: Lecture Notes in Computer Science, Vol. 3468.* Berlin: Springer-Verlag GmbH.

[15] Digia plc (n.y.), *Qt* [online]. URL: http: http://qt-project.org/ [accessed January 23rd 2015].

[16] Thomas Resch (2015): *Github repository for RWA* [online]. URL: https://github.com/funkerresch/rwaengine/ [accessed 2015, April 15].

[17] Matthias Krebs, Thomas Resch (2014): *Github repository for Large Scale Indoortracking* [online]. URL: https://github.com/fhnw-imvs/fhnw-audiowalk/ [accessed 2015, April 15].

[18] Kai Winter (2008), *qtmapcontrol* [online]. URL: http://www.medieninf.de/qmapcontrol/ [accessed January 23rd 2015].

[19] Jawbone (n.y.),Github Repository for the OpenAl Mob Library [online]. URL: https://github.com/Jawbone/OpenAL-MOB [accessed January 23rd 2015].

[20] Pei Xiang (2004), *earplug~* [online]. URL: http://sourceforge.net/projects/pure-data/files/libraries/earplug~/earplug~-0.2.tar.gz/download [accessed January 23rd 2015].

[21] Primate Labs (n.y.), *Geekbench* [online] URL: http://www.primatelabs.com/geekbench/ [accessed January 23rd 2015].