Chapter 3
Case Study: Virtual World Engine Staging a Pervasive Game

3.1 Choosing a Candidate Engine to Repurpose

Different types of game engines have been mentioned in Sect. 1.2 and a feature set to support pervasive games has been distilled and verified in Chap. 2. Because the feature set requires support for a virtual game world with world persistence, and a shared data space with data persistence, a virtual world engine is chosen as primary candidate for, being an engine in the same product line as a would-be pervasive game engine. As proof-of-concept, a specific virtual world engine implementation is extended to support the entire feature set and used to implement the pervasive game, called Codename: Heroes (CN:H). In this chapter, an explanatory case study (Johannesson & Perjons, 2014) is presented validating the chosen architecture and giving needed first-hand experience with the resulting architecture. Whereas features from the survey inform the architecture, the implementation of CN:H serves to highlight those features of particular importance and identify any open issues. Before dividing into technical details, the design aspects of CN:H are provided first.

3.2 Codename: Heroes

Codename: Heroes was to be developed in-house and due to limited resources needed the benefits of rapid development. From the outset, CN:H was specified to be a 'long term pervasive game', spanning months or years. The game world was said to overlap both the physical and virtual i.e., satisfying the sub-domain criteria. CN:H was designed to make explicit use of game mastering, both in day-to-day operation and specially designed weekend events. The game client, depicted in Fig. 3.1, was designed to be a prop 'in the mythos' (Jonsson & Waern, 2008) of the game. The

K.J.L. Nevelsteen, *A Survey of Characteristic Engine Features for Technology-Sustained Pervasive Games*, SpringerBriefs in Computer Science,
DOI 10.1007/978-3-319-17632-1_3

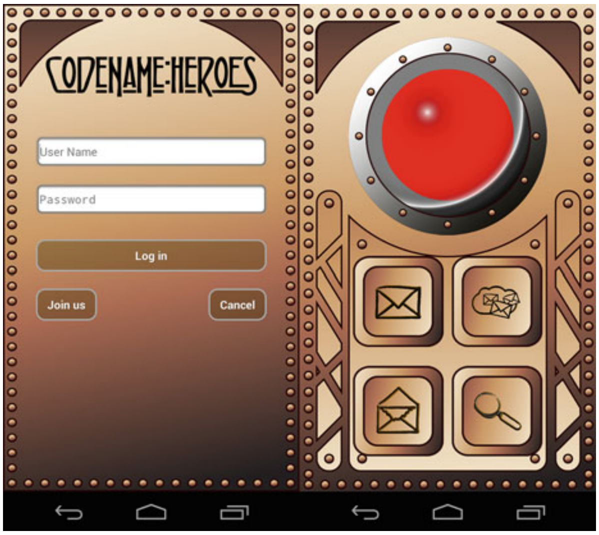

Fig. 3.1 Two screenshots of the game client, designed in the mythos of the game; the large circular area is designed to communicate the player's 'mana' level

game client was developed to the point of fully functional prototype. Further details on CN:H, from a cultural perspective, can be found in the works by Back and Waern (2013, 2014).

3.3 The Architecture

The specific virtual world engine implementation chosen, and centerpiece of the CN:H architecture (see Fig. 3.2), is the LambdaMOO (MOO) engine, by Pavel Curtis (MOO, 2012); a descendent in the family of MUD[1] architectures (collectively denoted as MU*/MOO), which can be traced back to the original virtual world implementation of 1978, called MUD1 (Bartle, 2003). The MOO engine stands as centralized server to all heterogenous devices. Only one game server was utilized, but with the idea that the centralized server could be expanded to multiple servers later. The centralized server is directly connected to non-volatile storage; for MOO this storage is a flat file, but a relational database is a common alternative approach.

[1]It is possible to trace the relation of MUD to pervasive games back to at least 2001. According to Nieuwdorp (2007), the first time the word 'pervasive' was used in conjunction with 'gaming' was by Falk (2001), where MUD was considered "a virtual counterpart to LARP". It is through the shared trait of world persistence that pervasive games have been said to be a direct descendant of MUD (de Souza e Silva & Sutko, 2009a).

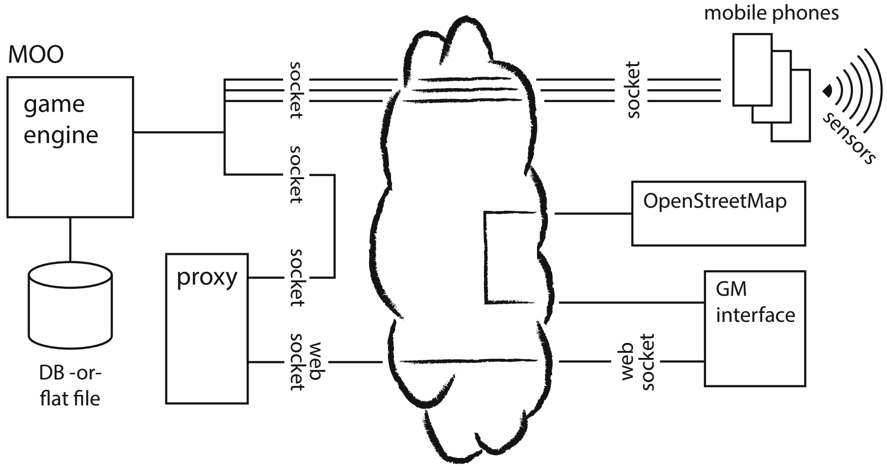

Fig. 3.2 CN:H software architecture: game engine connected to a database or flat file; networking of game engine with proxy and mobile phones through the Internet; mobile phones have sensors sensing the physical world; proxy connected via Web Sockets to the GM interface via the Internet; and, GM interface connected via the Internet to OpenStreetMap

To connect to the MOO server, clients can communicate via either the Telnet or MCP protocols (see Sect. 3.4.3 below). The game client was a fully functional Java-based mobile phone application, running on an Android-OS enabled smartphone, which is programmed to speak directly to the MOO engine. The mobile application opens a socket to and through the Internet to connect to the listening socket of the MOO server. Synchronization of local data with the server and possible disconnects are responsibilities of game client. Through sensors on the mobile phone (e.g., GPS, camera and accelerometer) the game client could interact with the physical world (e.g., the crowd-sourced artifacts, see Sect. 3.4.1 below).

An additional client to the game engine that was created, is the browser-based game mastering tool, for a GM to monitor player movement and activity. The GM tool is an OpenStreetMap WebMap implemented using HTML5 and Javascript, accessible via any web enabled computer or smartphone. The GM tool was a prototype precursor to the game master interface work by Guerrero Corbi (2014). To allow the web-based GM tool to connect to MOO, a small proxy server was created, which offered WebSocket (Fette & Melnikov, 2011) connectivity to the GM tool and simultaneously connected to MOO via Telnet and MCP (see Sect. 3.4.3 below). Essentially the proxy was an easy way to translate between HTML and MCP protocols.

3.4 Pervasive MOO in Relation to the Feature Set

MOO has rudimentary support for many of the features e.g., game mastering; roles, groups, hierarchies and permissions; content creation and scripting in run-time; access to the game state in run-time; and bidirectional communication that spans the virtual world. Obviously, by selecting an outdated engine, there were concerns pertaining to 'performance efficiency' (ISO, 2011). In response, it was estimated that one MOO server, at least initially, would be sufficient for CN:H; MUD1 in 1978 supported 36 simultaneous players and the MUD-based Gemstone after 1987, 650–1000 (Hall & Novak, 2008) players. Because the MOO codebase was created to run on now outdated hardware, it was estimated that around 3,000 text-based players could be supported on modern hardware, with a outer maximum of about 20,000, due to network port limitations. All visualization in the game is handled by the game client, leaving the MOO engine to process game events, albeit under heavier network load; rather than human produced text commands, the MOO would process game client events with sensor data.

3.4.1 Virtual Game World with World Persistence

MOO maintains a spatiotemporal world instance that retains player data seemingly indefinitely. CN:H has three different types of game elements that exist virtually in the MOO instance. First, crowd-sourcing was used to generate artifacts i.e., players built their own physical game elements, using virtual 'blueprints', which were assigned a unique identifier, in the form of an optically readable code (QR-code). These QR-codes could be read by the game client, to link the physical game object to its virtual counterpart (see Fig. 3.3). In this way, the game world overlapped with both the virtual and physical. Second, the players themselves, were also assigned a virtual game object that is linked to their game client, giving them presence in the virtual world. And third, a primary mechanic in CN:H was for players, or teams of players, to carry virtual messages towards specific goals. Messages were virtual game objects, without a physical counterpart, that were either virtually contained in an artifact or carried by players virtually. Both virtual messages and artifacts had an 'implied location' (Rashid, Mullins, Coulton, & Edwards, 2006) i.e., the location of a message or artifact had a location relative to, the player carrying or container holding, the artifact.

During the months or years that CN:H was planned to run, the game world would need to be ubiquitously available to the player, supporting temporal expansion. At any time of the day, a player would be able to turn on the game client and access the virtual game world i.e., the client could access the game server ubiquitously, requiring game engine reliability. MOO is proven to be reliable (i.e., mature, available, fault tolerant and recoverable (ISO, 2011)), through a long standing heritage of open-source community maintenance (Bartle, 2003).

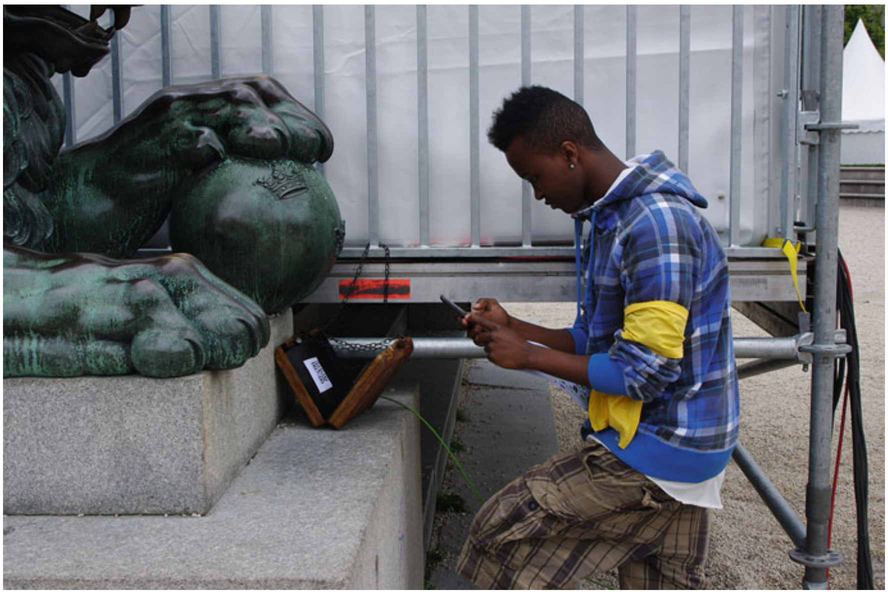

Fig. 3.3 Player scanning a physical object via its QR-code, to access its virtual counterpart

3.4.2 *Shared Data Space(s) with Data Persistence*

To provide a shared data space, the architecture for CN:H was initially conceived as client software running on a smartphone connecting to one or more centralized servers. MOO provides a shared data space for CN:H, and coordinates network communication from different clients to it. MOO periodically persists the world's data space, which resides in memory, to non-volatile storage, reloading the world in the event of system failure. Some issues with MOO are that: Holding all world data in memory simultaneously and periodically persisting it to a flat file is an outdated practice. Clients are responsible for their own data persistence. In the event of client failure, it is the client's responsibility to synchronize game state with the game engine. Another disadvantage of MOO is its inability to scale over more than one server, but considering the limited number of players initially, this was not considered an immediate problem. The MOO architecture could be expanded later e.g., the MUD-based engine running EverQuest was extended to handle 400,000 players distributed across at least 40 servers (Bartle, 2003).

3.4.3 Heterogeneous Devices and Systems

The game client and the GM tool were both created to provide ubiquity of access to the virtual game world while participants were on the move. Heterogeneity between the game client and the GM tool was extensive, differing in hardware type, operating system, programming languages and network protocols. In itself, MOO does not support heterogeneous devices. Originally MOO was designed to be accessed simultaneously by many players, each through a networked computer running a Virtual Terminal program and the Telnet protocol (Postel & Reynolds, 1983). As MUD clients became more diverse and elaborate, an extension to Telnet, called MUD Client Protocol (MCP) (MOO, 2012), was devised to support more elaborate and differing game clients.

MCP makes use of a concept called 'out-of-band data' (MOO, 2012) which seemingly splits the network communication channel in two, by escaping control messages[2] between client and server, allowing for asynchronous remote procedure calls on a channel hidden from the player. To effectively communicate via MCP, client and server must first negotiate a common interface beforehand. A disadvantage of the outdated MCP extension, is that although the engine can support many custom interfaces, all are linked to a single player login object, with MOO asserting the player is logged-in from only one device i.e., no support for crossmedia. This turned out to be a problem for game masters who wanted to access the GM tool, but were still also logged in via Virtual Terminal to access the game state. To work around this issue, a `ghost` player was created to handle a second incoming connection from a game master.

In CN:H, MCP formed part of the device abstraction layer to support the game client and GM tool. MCP proved sufficient for the game client, because the client could be programmatically controlled not to drop connections during times of uncertain connectivity. For the GM tool, however, because the HTML5 protocol is commonly connectionless, the proxy running alongside MOO provided a constant connection to the MOO. Via the proxy, MOO could access service-oriented architectures or provide its own services. In this sense, the proxy is part of the device abstraction layer, translating web protocols into MCP calls. To populate the WebMap on the GM tool, geographical data from OpenStreetMap is accessed as a web service and location data is accessed as MCP calls routed via WebSockets. The same communication technique was being considered to add social networking to CN:H, in the next iteration of development.

Although flexible, MCP does not fully resolve interoperability issues between heterogeneous devices and services e.g., one protocol has been agreed upon in advance, connectionless transmission is not supported and the need for stateless

[2]Escaping (or control signaling) is prefixing a message with a special marker so that it can be identified and handled differently i.e., as a control message.

transactions is not questioned. In this case, the implementation of CN:H raised awareness to assumptions that were made during the design phase, highlighting how critical the problem of interoperability is.

It is difficult to say with which service-oriented systems a pervasive game engine should be combined. It could be argued that all pervasive games make use of geographical data and therefore would benefit from being combined with a GIS or map data from OpenStreetMap. Context information is needed, so a constant connection with a wireless sensor network could also be argued for. Because of the prevalence of the Internet, it can also be argued that a game engine needs to be combined with a web server by default as well. In any case, the ability for a game engine to interface with other systems is important.

3.4.4 Context-Awareness

To obtain a degree of context-awareness in CN:H, each participant carried a smartphone running a copy of the game client. The game client could access the smartphone sensor and actuator hardware (e.g., 3G, GPS, Bluetooth, accelerometers, vibration motor), and communicate with the game engine via mobile networking, using asynchronous remote procedure calls. Note that 3G and Bluetooth are also sensors, because they can be used for position triangulation (de Souza e Silva & Sutko, 2009b) and proximity detection, respectively. Additionally, because the client maintained a nearly constant connection with the game server, when the player was online, player presence and activity could be detected, with inactive players being marked as 'idle'. Usage of context-awareness in CN:H was rather limited, using only position tracking and proximity. To allow for GPS-based positioning the MOO engine needed to be modified to support a different spatial model (Nevelsteen, 2014). The original MOO virtual world consists of a number of 'room' nodes with a directed graph between them. In CN:H, GPS position coordinates were added to relevant game objects, rather than use the room-based nodes. Player GPS coordinates were updated through position localization and the implied location of other objects equated to the player's position, whenever a player interacted with an object. Each GPS snapshot links a virtual game object to a particular location (Nevelsteen, 2014). Although the engine supported it, the design of CN:H explicitly avoided the need for detecting a player in a bounded area. Additional context information in CN:H, was the geographical data surrounding active players, provided for by the GM tool by OpenStreetMap. Other potential sources of context information, not used in CN:H, were GIS data, social media, or any information on the Internet e.g., see (Suomela, Räsänen, Koivisto, & Mattila, 2004).

Due to the mobile nature of the game, uncertainty had to be dealt with, both in position localization and degradation of mobile networking. The Telnet protocol (Postel & Reynolds, 1983) used by MOO was an advantage in the mobile setting, because as long as the connection was not explicitly closed, long periods of inactivity did not negate the connection.

3.4.5 Roles, Groups, Hierarchies, Permissions

MOO already supported roles for participants, including the ability to sort them into groups and hierarchies; a group being a 'collection object' holding other objects and each new hierarchy being a branch on the main MOO game object hierarchy. MOO provides three different participant roles by default: `player`, `programmer` and the all powerful `wizard` role. Groups were used to allow collections of players to venture through quests collectively i.e., a collection object, holding the group of players, could be tied to different quest stages and different timestamps (Nevelsteen, 2014). Hierarchies were used for classification e.g., determining the required permission level needed in order to interact with the various game objects. Each game object has associated owner and permission flags, to control if and how other roles are able to interact with the object e.g., the combination of roles and permissions were used to limit access to MCP functionality. Roles and permissions determined what actions game participants could take, including actions taken through GM tools. Aside from object properties, MOO allows one or more run-time scripts to be attached to each game object, each with their own associated owner and permissions, allowing for almost any additional functionality to be added to a game object. The all powerful `wizard` role ignores permission flags, granting wizards the ability to make any modification to the system, even those leading to catastrophic events in MOO. The three basic MOO roles, plus the additionally created role of `game_master`, were used in CN:H. Although not ideal, the permission system was sufficient. Implementing additional permission flags would have required invasive modification to the engine.

The game design included plans to outsource some GM responsibilities to advanced players (called 'Sages'), effectively crowd-sourcing the human resources needed to stage the game. But, due to time constraints, the feature was not implemented. Creating the role of `sage` would have meant deriving the role from the `game_master` role and reducing it's permissions.

3.4.6 Current and Historical Game State

The minimal world configuration (called a 'core' (MOO, 2012)) that can be loaded into the MOO data space, does not require any player information; each game object, including the player object, is assigned and identified by an auto-incremented number during creation. The core that was expanded on in CN:H is called JHCore (MOO, 2012) and requires, as player information, a name and password. Additional player information specific to CN:H was added to the player game object e.g., bluetooth identifier, email address, ‘mana’, ‘available rituals’ and ‘available blueprints’.

Each game object (including it’s identifier, permissions, parent object, properties and associated scripts) is encoded in a text format in the MOO data space. The data space is stored in memory and directly accessible to those with permission, via the command line interface of a Virtual Terminal. Thus, a wizard can access the entire world state in the data space and optionally export it to storage i.e., all game data could be logged. Logging provided a historical view of game state e.g., turning GPS coordinates into GPS trails. Unfortunately, support for advanced logging, such as streaming data, was lacking in MOO. After each staging of CN:H, a post-game analysis was performed, including questionnaires and log analysis. Results of the analysis was for research purposes and to incrementally improve the game design.

MOO supports a system for documenting that was sufficient for CN:H, but with the drawback that, it did not support any data types other than text e.g., binary data such as images or sound. Documentation detailing how to use questing was created in the documentation system, but since the stagings of CN:H were with a relatively small number of participants, the documentation system was not used to pass information between game masters.

3.4.7 Game Master Intervention

Because providing game mastering requires a large amount of resources (Thompson, Weal, Michaelides, Cruickshank, & Roure, 2003; Flintham et al., 2003), most virtual worlds are designed to run fully automatic, with only a minority of worlds being semi-automatic or fully game mastered role-playing worlds (Bartle, 2003). Therefore the MOO architecture features only rudimentary game mastering tools e.g., to aid players who have technical issues or social conflicts within the world.

CN:H was designed to run semi-automatic; mostly automatic during day-to-day operation, but with certain events being flagged for game master intervention, and the possibility for fully game mastered role-playing events during weekends. For a game master to be able to effectively intervene in a running stream of events, a mechanism needed to be in place to: stop an event and all progression dependent on that event; signal a game master that a decision was needed; and resume progression according to the GM’s decision or according to a default value, after a timeout. In CN:H, progress in the game was represented as progression through stages of a quest. MOO room nodes were used to represent quest stages and the graph between them represented all possible progressions through the quest. To implement GM intervention, an ‘intervention bit’ was added to each room game object i.e., each quest stage. If the bit was set, a script associated with the quest stage would execute, checking conditions to see whether a game master needed to be notified. The GM intervention bit was implemented and tested as proof-of-concept, but was never tested during play.

No specialized game master interface was considered at design time. A GM interface could be either implemented directly in MOO (with access to the entire game state e.g., including any roles or permissions), or MOO could provide a selection of the game state through web services for a third party GM interface, using the proxy described previously. Initially MOO's text-based command interface was used for all game mastering and MOO allows direct alteration of the entire game state in run-time. Advanced CN:H specific GM commands were created in MOO's run-time scripting language e.g., performing a series of basic commands or translating event data into a consumable form. After the initial play testing, the versatility of MCP was fully understood and the minimalistic GM tool (the OpenStreetMap WebMap) was implemented for the role of `game_master`, as a visualization of virtual objects and their GPS locations. After the second staging of CN:H, a Master's Degree project was carried out to "develop a generic architecture for interfaces of game-masters of pervasive games that allows adaptability" (Guerrero Corbi, 2014).

3.4.8 Reconfiguration, Authoring and Scripting in Run-Time

The MOO virtual world engine is designed to run continuously, so the phases of pre- and in-game are one and the same; any functionality available pre-game was available in-game also. By tying the game mechanics to game object locations, rather than specific physical locations, CN:H strived to obtain location adaptability. In run-time, but still pre-game quests could be created or altered to suit a specific staging. Once in-game, modifications could be made, but care had to be taken not to disrupt the game in progress i.e., special weekend events could still be created leaving the day-to-day mechanics undisturbed.

A major reason to choose the MOO implementation was specifically because it fulfilled the requirement of run-time content creation and scripting, described by Bartle (2003) as highly dynamic. The run-time scripting language in MOO is called MUD Object-Oriented; the 'fully expressive' scripting language, created by Stephen White in 1990, was created enabling users to create beyond what was originally imagined by the original MOO developers (Bartle, 2003). Content authoring in run-time, was used in CN:H, to allow for content to be crowd-sourced e.g., players were allowed to create personal artifacts that could be imported into the game. One caveat encountered was that, content creation that requires change in the physical world, without an actuator to bring about that change, is impossible. This became apparent at the design time of CN:H e.g., how can CN:H create virtual game objects with a physical counterpart, with potentially massive amounts of players spread all over the world? A solution is to "hack into reality" (Jonsson & Waern, 2008); tie existing phenomena from the physical world into the game world. Crowd-sourced player artifacts were coupled to a virtual object via generated and printed QR-codes. 3D scanning of physical objects was considered, but QR-codes were chosen because they were simpler to implement. A similar caveat applies to run-time content creation in combination with heterogeneous devices. Although, the game engine supports run-time content creation, game clients wanting to make use of new content, have to support dynamic content also. MCP solved inconsistencies between client and server when dealing with an all text-based content. But, if content has dependencies that can only be resolved at compile time (e.g. thick clients), new run-time content on the client side is limited to what the existing framework supports (Bell, 2007). An option is to use HTML or other markup language, but it is not a complete solution since not since all types of content can be represented. This again highlights an interoperability problem.

CN:H was implemented entirely using scripting language, without the need to recompile the engine code. The scripting language supports autonomous agents and was used to create the NPC called `void_walker`. Because scripting is done in run-time, so is debugging. This made the entire virtual world a continuous simulation where the WOz technique could be used to simulate game play prior to staging. All game elements in the physical world had a virtual counterpart that could be manipulated to simulate player interaction.

3.4.9 Bidirectional Diegetic and Non-diegetic Communication

Unless players deliberately choose another medium (e.g., to limit communication disclosure (Bergström, 2011)), all communication in MOO is intended to stay within the virtual world. The chat communication channel is bi-directional and can be used for both diegetic and non-diegetic purposes. MOO features a mail system for delayed communication and a news channel for uni-directional communication to the players in the virtual world. Since players in CN:H are playing in the physical world and not continually behind a stationary computer screen, communication mediums in MOO were not sufficient. The graphical user interface, sensors and actuators on the game client/smartphone, provided for partial diegetic communication (e.g., through blinking lights, accelerometer readings or haptic vibration feedback), but the main uni-directional diegetic channel was achieved by extending the MOO mail system into the game client. Although MOO supports a bi-directional chat system, the game client did not provide access to it. Using the client as an non-diegetic bi-directional communication channel was decided against, as not to break the mythos of the game i.e., the presence of a non-diegetic channel would most likely reduce the fiction surrounding the game client. Due to limited development resources, no alternative was created for non-diegetic communication, and so this defaulted to email and phone conversations i.e., soft events that could not be picked up by the engine.

3.5 Discussion

To summarize, MOO and its extensions (including MCP) supported the feature set from the survey, with most of the work revolving around engineering interoperability. All changes implementing CN:H, including interoperability via MCP, were scriptable in the run-time scripting language; no engine code was modified i.e., indicating that the engine was an appropriate choice. It was, however, felt that more commonly used scripted functionality (e.g., those routines responsible for geodesic distances and triangulation) should be moved to the engine, providing easier access to common functionality and better performance, by being implemented in a compile time language.

As of this writing, Codename: Heroes has been publicly successfully staged twice in the area of Stockholm, Sweden, with no issues from the game architecture. CN:H as proof-of-concept seems to indicate that a virtual world engine, supporting the resulting feature set, could be successfully repurposed to stage a pervasive game. In no way is MOO an ideal engine, given the problems, caveats and disadvantages outlined in each of the feature sections above, but the case study seems to underline that MOO is at least in the same product line as a would-be pervasive game engine.

References

Back, J., & Waern, A. (2013). ‘‘we are two strong women” – designing empowerment in a pervasive game. In *DiGRA 2013 - defragging game studies*. Atlanta, GA, USA: DiGRA.

Back, J., & Waern, A. (2014). Codename Heroes–designing for experience in public places in a long term pervasive game. In *Proceedings of the 9th international conference on the foundations of digital games*. Ft. Lauderdale, FL, USA: Foundations of Digital Games.

Bartle, R. (2003). *Designing virtual worlds*. Indianapolis, Indiana, USA: New Riders Publishing.

Bell, M. (2007). *Guidelines and infrastructure for the design and implementation of highly adaptive, context-aware, mobile, peer-to-peer systems*. (Doctoral dissertation, University of Glasgow, Faculty of Information and Mathematical Sciences, Department of Computing Science).

Bergström, K. (2011). Framing storytelling with games. In *Interactive storytelling* (pp. 170–181). Lecture Notes in Computer Science. Vancouver, Canada: Springer Berlin Heidelberg.

de Souza e Silva, A., & Sutko, D. M. (2009a). Digital cityscapes. (Chap. Merging Digital and Urban Playspaces: An introduction to the Field, pp. 1–20). USA: Peter Lang.

de Souza e Silva, A., & Sutko, D. M. (Eds.). (2009b). *Digital cityscapes*. USA: Peter Lang.

Falk, J. (2001). MUD nexus: the world as game board for computer games. In *CHI2001, workshop on distributed and disappearing user interfaces in ubiquitous computing*.

Fette, I., & Melnikov, A. (2011, December). *IETF tools: the websocket protocol: RFC6455*. Retrieved from http://tools.ietf.org/html/rfc6455

Flintham, M., Benford, S., Anastasi, R., Hemmings, T., Crabtree, A., Greenhalgh, C., ... Row-Farr, J. (2003, April). Where on-line meets on the streets: experiences with mobile mixed reality games. In *Proceedings of the SIGCHI conference on human factors in computing systems* (pp. 569–576). CHI ’03. New York, NY, USA: ACM.

Guerrero Corbi, V. (2014, July). *Development of a web based interface for game-masters of pervasive games*. (Master’s thesis, Universitat Politècnica de Catalunya).

Hall, R., & Novak, J. (2008, April). *Game development essentials: online game development.* Clifton Park, NY, USA: Delmar Cengage Learning.

ISO. (2011). *ISO/IEC 25010:2011 systems and software engineering – systems and software quality requirements and evaluation (SQuaRE) – system and software quality models.*

Johannesson, P., & Perjons, E. (2014). *An introduction to design science.* Springer International Publishing Switzerland.

Jonsson, S., & Waern, A. (2008). Art of game-mastering pervasive games, the. In *Proceedings of the 2008 international conference on advances in computer entertainment technology* (pp. 224–231). ACE '08. New York, NY, USA: ACM.

MOO. (2012). *LambdaMOO (official site).* Retrieved from http://www.moo.mud.org

Nevelsteen, K. J. L. (2014, September). Applying GIS concepts to a pervasive game: spatiotemporal modeling and analysis using the Triad representational framework. *International Journal of Computer Science Issues, 11* (5).

Nieuwdorp, E. (2007, April). The pervasive discourse: an analysis. *Computers in Entertainment (CIE), 5.*

Postel, J., & Reynolds, J. (1983, May). *IETF tools: telnet: RFC854.* Retrieved from http://tools.ietf.org/html/rfc854

Rashid, O., Mullins, I., Coulton, P., & Edwards, R. (2006). Extending cyberspace: location based games using cellular phones. *Computers in Entertainment (CIE), 4* (1), 4.

Suomela, R., Räsänen, E., Koivisto, A., & Mattila, J. (2004). Open-source game development with the multi-user publishing environment (MUPE) application platform. In *Entertainment computing – ICEC 2004* (pp. 308–320). Eindhoven, The Netherlands: Springer Berlin Heidelberg.

Thompson, M. K., Weal, M. J., Michaelides, D. T., Cruickshank, D. G., & Roure, D. C. D. (2003). *MUD slinging: virtual orchestration of physical interactions.* ECSTRIAM03-007.