Integrated Project on Pervasive Gaming

FP6 - 004457

Work package WP12: *Showcase – City as Theatre*

Deliverable D12.5: Delivery of the Second City as Theatre prototype

Editors: Martin Flintham, Steve Benford and Mauricio Capra

Authors: Chris Greenhalgh, Martin Flintham, Michael Wright, Jonathan Green,
Keir Smith, Adam Drozd, Matt Adams, Ju Row Farr,
Nick Tandavanitj, Hanna Talbot, Irma Lindt, Johan Peitz, Steffan Björk, Alain Becam

Version: 1.0

Release date: September 2006

Status: *public*

Executive Summary

This document describes the second major iteration of the design of Day of the Figurines that has been realised in the second phase of the iPerG project in the City as Theatre showcase.

Day of the Figurines is a long-term text messaging game for mobile phones that has been designed by professional artists. The game follows a day in the life of an imaginary small town. Day of the Figurines takes the form of a massively multiplayer board-game, but one with which players can interact remotely via their phones. A player enters a public venue to choose a plastic figurine that is placed onto a large-scale physical game-board. An hour or so after leaving the venue, the player will receive their first text message. From now on they interact with their figurine by sending and receiving SMS messages, controlling its movements and actions and seeing events through its eyes. The game involves a high degree of pre-authored content in the form of key events that are scheduled to take place at different destinations. Threaded among these events are a series of missions and dilemmas that confront players in which players use objects to try to help (or hinder) others and to maintain their own health. Day of the Figurines is designed to be a slow game so as to reflect the nature of SMS messaging. The game time takes place over the course of one fictional day. However, this is mapped onto twenty four days of real time, during which the game is played for ten hours each day.

The first iteration of Day of the Figurines was hosted by the Laban Centre in London and played by 85 players over a month spanning August and September 2005. A detailed evaluation of this first iteration was given in the previous iPerG deliverable D12.4, drawing on a combination of player feedback via questionnaires, analysis of system logs and ethnographic observation to unpack key issues concerned with both the player experience and the behind-the-scenes work that was required to deliver the game. The second iteration of Day of the Figurines described in this deliverable represents a major reworking of the game, both in terms of its content and the technology and processes used to deliver it. In particular, this deliverable introduces the following key innovations that have emerged during this second phase:

? Support for scalability through automation of the game rules and message generation and also the use of a simple game grammar. Whereas the first version of Day of the Figurines required a substantial amount of operator work to generate and edit messages, this second version is highly automated and should scale to supporting many hundreds of simultaneous players.

? The addition of greater structure and content to the game including the use of objects, a health system and most importantly, missions which combines events, dilemmas and objects into more extended narrative structures within the game.

? Support for episodic play, combining greater responsiveness, with quickly backing off, with daily keep alive messages. Key to supporting episodic play is a new ‘hub-based’ movement model that enables player’s to move more quickly through the virtual town while also enabling them to be allocated small episodes of play – encounters, events or dilemmas – as they go.

? Techniques to limit message flow so as not to flood players and stay within the cost constraints imposed by SMS (which affect both players and game operators). Flow control mechanisms include the use of silos at destinations to manage the volume of chat and the introduction of message aggregation which involved automatically combining information about several events into a single messages so as to make maximum use of previous SMS bandwidth (a technique that we anticipate to be more widely applicable across SMS-based pervasive games)

? Digital augmentation and physical redesign of the physical gameboard so as to provide a more compelling spectator interface to the game and also to make its operation more efficient and therefore scalable.

? More sophisticated authoring, operation and orchestration interfaces for the game, supported by a back-end database and webserver for managing the game content. A key innovation here is that these interfaces and the supporting database and many java classes required for development are mostly automatically generated from some core Java bean specifications.

This document describes the second version of Day of the Figurines in a uniform IPerG standardized game design structure, explaining the game design, game flow, game content, employed technology, user interfaces and the content of the software distribution. Each IPerG game is described in this common format. Please refer to document D5.3B for a ludological definition of pervasive games and a conceptual framework for understanding and discussing the domain of pervasive gaming.

Purpose of this Document

The purpose of this document is to provide a summarized and structured presentation of the City as Theatre phase two showcase prototype Day of the Figurines (version 2). In addition, this document should provide a pervasive game design example and is intended to foster pervasive gaming design ideas and realizations in Europe.

Target Audience

This document is intended as a public document to all interested parties within the European game designer community and is intended to foster pervasive gaming development within the European Community.

Deliverable Identification Sheet

IST Project No.	***FP6 – 004457***
Acronym	**IPerG**
Full title	Integrated Project on Pervasive Gaming
Project URL	http://www.pervasive-gaming.org/
EU Project Officer	Albert GAUTHIER

Deliverable	**D12.5 Delivery of the second City as Theatre prototype**
Work package	**WP12 City as Theatre**

Date of delivery	**Contractual**	M 24	**Actual**	M24
Status	version 1.0		final ✍	
Nature	Prototype ✍ Report ✍ Dissemination ✍			
Dissemination Level	Public ✍ Consortium ✍			

Authors (Partner)	University of Nottingham, Blast Theory, FIT, Interactive Institute			
Responsible Author	Steve Benford		**Email**	sdb@cs.nott.ac.uk
	Partner	Nottingham	**Phone**	+44 115 9514203

Abstract (for dissemination)	This document describes the game Day of the Figurines (version 2), a large-scale long-term text messaging game realized in the second phase of the IPerG project. In Day of the Figurines, players send and receive SMS messages over the course of a month to remotely control their figurines that move across a large-physical game board. The game follows a day in the life of a virtual town, confronting its players with various events, dilemmas and missions as they try to help one another. This document describes the second iteration of Day of the Figurines in a uniform and standardized IPerG format, introducing changes to the game content and technology that are intended to make it more scalable, more responsive to the characteristics of SMS, and more engaging for both players and spectators.	
Keywords	game design, pervasive gaming, SMS, text messaging	

Version Log			
Issue Date	**Rev No.**	**Author**	**Change**
2006-07-31	1.0	Steve Benford	First Version

2006-10-06	2.0	Martin Flintham	Version for first internal review
2006-10-12	3.0	Anders Ernevi	Final Version

Table of Contents

Table of Figure

1 INTRODUCTION

This document provides a comprehensive overview of the game design of the second City as Theatre prototype "Day of the Figurines". It is based on a common IPerG game design description structure which covers the game design, game flow, game content, employed technologies and user interfaces.

Day of the Figurines is a long-term text messaging game for mobile phones that has been designed by professional artists. The game follows a day in the life of an imaginary small town. Players use their mobile phones to control characters in the town by sending and receiving SMS messages, visiting destinations, meeting other players, witnessing events that take place and dealing with dilemmas that face them. Physically, Day of the Figurines takes the form of a massively multiplayer board-game, but one with which players can interact remotely via their phones. A player enters a public venue to find a large-scale table-top model of an imaginary town. They are asked to select their character from a display of plastic figurines. They give their chosen figurine a name and a description which are entered into a database along with their mobile phone number. Their figurine is then placed on the board at a random position in the town in order to introduce them into the game. As they leave the venue, they are given a small map of the town and a set of rules for the game. An hour or so after leaving the venue, the player will receive their first text message. From now on they interact with their figurine by sending and receiving SMS messages, controlling its movements and actions and seeing events through its eyes. By replying to this first message with the name of a destination in the town their figurine is set on a path towards that destination. Intermittently the player receives text messages to alert them when their figurine arrives at a destination and to nearby figurines with whom they can chat.

The game involves a high degree of pre-authored content in the form of key events that are scheduled take place at these different destinations for example: pubs open, shops close, players discover graffiti and hear news from the pirate radio station, a fete takes place at the park, there is a total eclipse, two Scandinavian death metal bands play at the Locarno nightclub, two lovers are found dead at the cemetery and an army of soldiers enters the town. Threaded among these events are a series of dilemmas that confront players. A scenario is described. The player is asked what they want to do in response and an outcome is determined. Outcomes nearly always result in the deterioration of their figurine's health and in extreme circumstances they get killed and their game is over.

Day of the Figurines is designed to be a slow game so as to reflect the nature of SMS messaging. The game time takes place over the course of one fictional day. However, this is mapped onto twenty four days of real time, during which the game is played for ten hours each day.

The first iteration of Day of the Figurines was hosted by the Laban Centre in London and played by 85 players over a month spanning August and September 2005. A detailed evaluation of this first iteration was given in the previous iPerG deliverable D12.4, drawing on a combination of player feedback via questionnaires, analysis of system logs and ethnographic observation to unpack key issues concerned with both the player experience and the behind-the-scenes work that was required to deliver the game. The second iteration of Day of the Figurines described in this deliverable represents a major reworking of the game, both in terms of its content and the technology and processes used to deliver it. In particular, this deliverable introduces the following key innovations that have emerged during this second phase:

? Support for scalability through automation of the game rules and message generation and also the use of a simple game grammar. Whereas the first version of Day of the Figurines required a substantial amount of operator work to generate and edit messages, this second version is highly automated and should scale to supporting many hundreds of simultaneous players.

? The addition of greater structure and content to the game including the use of objects, a health system and most importantly, missions which combines events, dilemmas and objects into more extended narrative structures within the game.

? Support for episodic play, combining greater responsiveness, with quickly backing off, with daily keep alive messages. Key to supporting episodic play is a new 'hub-based' movement model that enables player's to move more quickly through the virtual town while also enabling them to be allocated small episodes of play – encounters, events or dilemmas – as they go.

? Techniques to limit message flow so as not to flood players and stay within the cost constraints imposed by SMS (which affect both players and game operators). Flow control mechanisms include the use of silos at destinations to manage the volume of chat and the introduction of message aggregation which involved automatically combining information about several events into a single messages so as to make maximum use of precious SMS bandwidth (a technique that we anticipate to be more widely applicable across SMS-based pervasive games)

? Digital augmentation and physical redesign of the physical gameboard so as to provide a more compelling spectator interface to the game and also to make its operation more efficient and therefore scalable.

? More sophisticated authoring, operation and orchestration interfaces for the game, supported by a back-end database and webserver for managing the game content. A key innovation here is that these interfaces and the supporting database and many java classes required for development are mostly automatically generated from some core Java bean specifications.

This document describes the second version of Day of the Figurines in a uniform IPerG standardized game design structure, explaining the game design, game flow, game content, employed technology, user interfaces and the content of the software distribution. Each IPerG game is described in this common format. Please refer to document D5.3B for a ludological definition of pervasive games and a conceptual framework for understanding and discussing the domain of pervasive gaming. The appendices to this deliverable present detailed specifications of key aspects of the second version of Day of the Figurines, covering:

? Game model specification

? Events specification

? Message specifications, linked to game events

? Authored text reference.

? Message elements

? Aggregation and pacing.

2 Game Design Overview

Features	Day of the Figurines Game Design
Research Goals	The overall research goal for this second iteration of Day of the Figurines is to explore the issues involved in designing and deploying an artist-led pervasive game bas ed on SMS text messaging for mobile phones. This includes developing and evaluating new techniques to support the creation of long-term pervasive games based on SMS that can be gracefully interwoven with the patterns of daily life. This overall goal breaks down into the following more specific sub-goals: **Game structure and content** – extending the previous game structure and content with new mechanism to give players a greater sense of overall purpose within the game. **Scaling and automation** – to introduce new structures and processes that will enable the game to scale from of the order of 100 players from the first version to many hundreds of players. **Episodic play** – to support highly episodic patterns of individual play (a tendency that we saw in the first Day of the Figurines). **Message flow** – to manage the flow of messages to players, responding quickly when appropriate but without flooding them. **Spectator Interface** – to provide a compelling spectator interface that introduces players to the game and that situates it within the specific physical venue where it is staged.
Game Setting	Day of the Figurines is placed in an artistic venue, most likely associated with a new media arts festival. This must be indoors where the physical game board will be assembled. Players and the general public are invited to observe and join the game. Operators are available to clear up any doubts, give more information and enter players into the game. After the player fills the form to join the game he/she receives a short manual with the game description. Players are also invited to visit the Day of the Figurines web site.
Games Area	Any place where computers and the table could be placed indoors.
Genre	Day of the Figurines is a long-term pervasive game for mobile phones that is intended to be interwoven with patterns of daily life. Players only need visit the venue just once to join the game. The game is subsequently played on their mobile phones through text messages.
Target Group	Day of the Figurines is aimed at the broad audience for new media art works, who are generally anticipated to be a reasonably 'tech savvy' and also to have a well developed cultural and critical perspective on interactive art and games.
Story Line and Game Play	Day of the Figurines is based on sending and receiving SMS messages where players interact with the game through their mobile phones. The goal of the game is to help other players. To play, visitors enter to a public space where they find a large scale model of an imaginary town. The visitors are invited to choose a figurine. They give to the figurine a name and answer basic questions about him or her. This information is input in a data base and the figurine starts its journey at the edge of the town. Players also have a web interface: www.dayofthefigurines.co.uk. Once a player has been registered in the game and has chosen their figurine, they receive their first message from the game, asking them where they would like to go. Through keyword commands, players can indicate new directions for the figurines, chat with other players, use objects or perhaps be involved in dilemmas, events and missions. The complete specification of the game and its associated game mechanics is given in the appendix, and the implementation of this can be viewed in the accompanying source code. The remainder of this section will briefly highlight notable changes from the first prototype, which has been described and evaluated in D12.2 and D12.3, namely the introduction of automated message parsing, message pacing and aggregation, the Hub movement model,

Silos, Things and Missions.

Keywords and Automated Parsing

The first prototype of Day of the Figurines, presented at the Laban Centre in 2005, involved operators manually reviewing and parsing incoming messages from players in order to determine what the player wished to do. Although many players reported enjoying this improvisational aspect of the prototype, this placed considerable constraints on the possibilities for automation, and how easily the experience could be scaled to larger numbers of players.

For this reason, this second prototype has introduced a palette of keywords which reflect a set of actions that a player can perform within the game in order to interact with the various destinations, objects and other players. All interaction with the game is now parsed automatically using these keywords, and players must send messages using them in order to play the game.

Each message must begin with a keyword, and optionally followed by an argument appropriate to the action a player wishes to perform. The keywords available are as follows:

? Say <text to be said to nearby players>

? Go <name of a destination to travel to>

? Pickup <name of an object at the current destination>

? Drop <currently held object>

? Use <name of an object either held or at the current destination>

? Find <player name at current destination>

? Help (returns a list of keywords and also notifies operator)

? Update (returns player's current status, destination, health etc)

? Leave town (quits the game)

? A,B,C (responds to a dilemma)

Message parsing is described in more detail in the next section, game realisation, and the specific function of each action is described in detail in the appendix.

Message Pacing and Aggregation

In the first prototype, operators performed manual filtering of outgoing messages, to avoid swamping a player with potentially duplicated, mundane messages. Similarly, players who were deemed to be inactive were more likely to only receive messages that were deemed to be particularly interesting to them.

This second prototype continues this theme in an automated fashion. Message aggregation attempts to digest recent events in order to reduce the number of messages sent, by making a distinction between events that warrant an immediate response, such as an event that directly affects a player, and events that provide more background information, such as other players arriving and leaving at a destination. For each of the former that generates a message, the message aggregation system attempts to fill the remainder of the available message space with text regarding the latter.

Secondly, the message aggregation system now maintains a record of which events a player has been told about, and to what extent. For example, if a player arrives at a destination that they have already visited, then they receive a short description that creates room for listing nearby players and objects, on the assumption that they will have received a long description the first time they arrived.

Finally, the first prototype had a certain weighting towards transactional content in messages (e.g. Player A arrives, Player B leaves) rather than more interesting descriptions. For the new version, some of these kinds of messages have been removed altogether and in other cases, space at the end of messages is used to add more descriptive content.

Message pacing and aggregation is discussed in more detail in the next section, and a list of game events and whether they create instant or aggregated message content can be found in the appendix.

The Hub

The first prototype was based around a Cartesian space, with players moving a number of squares across the board each hour to move between destinations. This meant that there were long periods of time while a player was travelling where, even if they wished to play, there was little content and few other players with which to interact.

On the assumption that a player who had just sent a movement message to the game wanted to play, this movement model was discarded in order to better offer engaging content in response. This second prototype has therefore moved to a room model, similar to older text adventure games, where each destination is only ever two moves away from any other. This model operates by introducing the Hub – an 'in-between' place that represents a player being in transit between destinations, and allocates interesting content to players who have just asked to go to another part of the town. Each time that a movement command is received, the game engine moves the player to the Hub, and after a short time onto their desired destination.

Players entering the Hub are placed in a dynamically allocated Silo (Silos are discussed in more detail below). A Silo can be thought of as an instance of a particular Destination, and a player in a Silo is not aware of other players in different Silos, even though they may be at the same Destination. Upon entering the Hub, the system looks for possible content that can be allocated to the player. There are four different types of Hub that a player may be allocated, based on weighted random chance:

? Local Event – the player receives a one-off descriptive event

? Dilemma – the player is given a dilemma which will expire when they leave the Hub

? Mission – the player is allocated a mission that they will retain having left the Hub

? Chat – the player is paired up with another player already in the Hub who has also been allocated Chat, to foster interaction and conversation. When both players chat, the timer that will move the player on is extended to prolong the conversation. If one or both players stop chatting, they are moved on to their target destinations.

The allocation weighting of the different types may be manipulated while the system is running, allowing the game operators to foster more interaction between players by increasing the chance of players being allocated a Chat Hub experience.

Finally, the Hub model also reduces the workload of the operators moving figurines on the game board, as players no longer need to be moved many times to move between two Destinations.

Silos

A second significant change to the first prototype's Destination model is the introduction of Silos, as described above. Rather than Destinations having an absolute capacity, this second prototype introduces the concept of Silos to enable the notion of 'quiet' and 'busy' Destinations.

Each Destination now has an unlimited capacity, and contains a number of Silos, with the capacity of each Silo determined by the Destination. When a player arrives at a Destination, they are either placed in an existing Silo if there is one with space, or a new Silo is created for this player. Silos are destroyed when they are empty. Players may move between Silos overriding the authored capacity to find other specific players that they may be looking for.

This new model means that, for example, the Canal, having a Silo capacity of one, will always be a quiet place that players may go to if they do not wish to chat to other players, and yet many hundreds of players may simultaneously be at the Canal. Similarly, the Locarno is deemed to be a busy destination and has a Silo capacity of ten, meaning that many more players may chat at the Locarno and have the feeling that they are in a busy place, without being overwhelmed by receiving many messages regarding all of the players in the parent Destination.

Things

This second prototype introduces objects that the player can interact with within the game, known as Things. Things can effect a players health, be used on other players, or act as devices to move one or more players inside or outside a Destination that is closed.

A Thing may be carried by a Player or a Destination, although a player may only carry one Thing at a time. Things spawn at specified Destinations, and on use by a player may either be destroyed, re-spawn or remain in the player's possession.

There are five types of Thing:

? Food – affects the using player's health

? Clothing – is automatically worn by the player on pick up

? Medicine – is used on an incapacitated player in the same Silo

? Other – is used on a randomly selected player in the same Silo

? BigBang – affects all players in the current Destination

Things have a variety of usage effects depending on their type, which determine what happens to bystander players and the protagonist player, and what messages they receive as a result.

Things have been introduced to provide a dedicated mechanism for players to help or to interact with other players, for example raising a player's health through the use of medicine, or lowering it through the use of a weapon. They also provide a mechanism upon which Missions can be built in order to add more structure to game play.

Missions

This second prototype introduces Missions, in response to feedback from the first prototype that indicated that some players found a lack of direction and structured player in the game, and they did not know what to do in order to progress.

Missions may last for hours or days, and have an explicit and concrete goal within the town, for example helping a specific player, or performing a specific task. Missions may draw on a list of incapacitated players who require assistance, or specific Destinations and Things when allocated.

Each Mission begins with a message sent to the player setting the scene of the Mission and instructing them what to do. One or more Mission Criteria are then authored, each of which has a specific trigger and optional requirement context, and also states whether fulfilling this criteria completes or terminates the Mission. For example, a Mission may involve finding some clothing in order to keep warm. In this case, a Criteria is authored that is triggered by the player picking up a Thing of type Clothing. This completes the Mission, and the player is sent a message informing them of this. There is also a Criteria that is triggered if the player runs out of time on this particular Mission.

An example of a more complex Mission is as follows, 'the gig at the Locarno':

? On Mission allocation, the player receives the text "a rat faced man in a waistcoat rushes up: 'The drummer's been arrested. Find a drum kit and get to the Locarno by 11pm to take his place.'" The player now has two game hours to complete the mission

? If the player arrives at the Locarno, while carrying the Thing 'drumkit' within the allotted time, they receive the text "you made it! Troll skal atter herske. Back stage the vocalist glowers at you: "From now you Juergen" He pokes your head with the sign of the goat. Go!" The player is moved 'outside' the Locarno, which is described as being a backstage VIP area at this time.

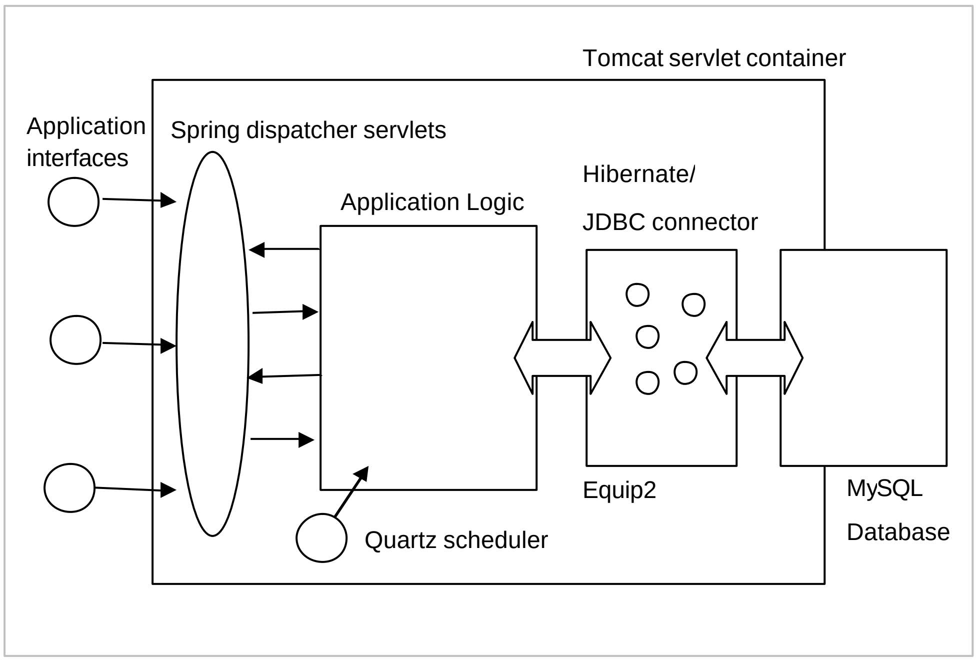
? When the Mission expires, if the player is 'outside' the Locarno, with the drumkit, they must have completed the above Criteria, and receive the following message which completes the Mission: "you and your new long haired chums sweep victoriously onto the stage. You all thrash that kit like nordic lords while androgynous goths gaze in awe."

? However, if the Mission expires and the player is not waiting 'backstage' at the Locarno, they receive the following: "horned, cloven disaster! You're too late.

Satan's Bubbly Fjord have already found a drummer. There's still time to enjoy the gig though.” The Mission has been failed.

Missions and Mission Criteria are specified in detail in the appendix.

3 Game Realisation

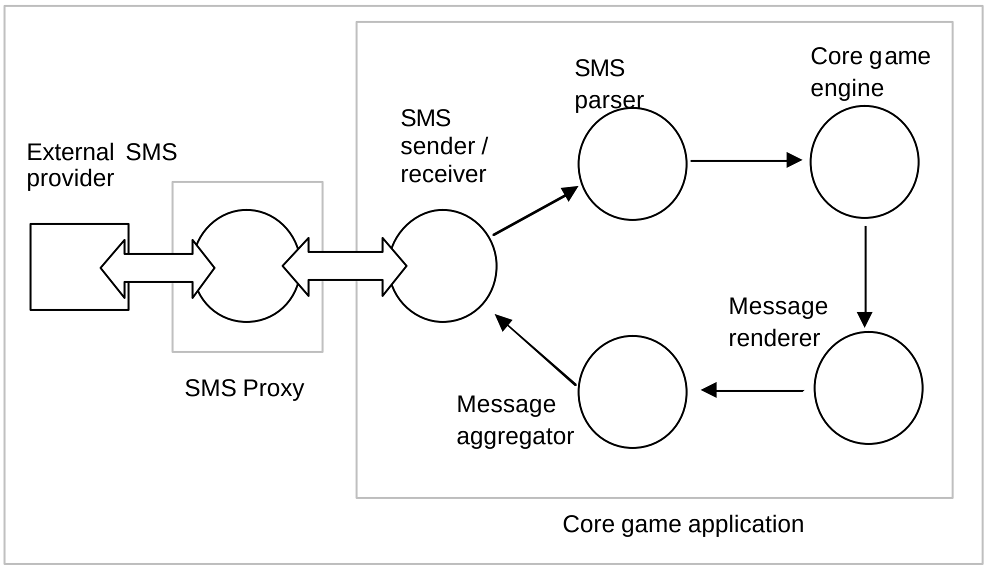
Features	Day of the Figurines Game Design
Introduction	This section describes how the game design specification of Day of the Figurines, as reviewed in the previous section, has been realised and implemented. A formal description of the game design, and accompanying game state, can be found in the appendix. This section begins by giving an overview of the infrastructure that supports the game, which was built as a J2EE web application constantly running on a web server. Next, it introduces the architecture of the game application itself, showing how this was split into a number of logically discrete components. Finally, each of these components is described in turn.
Supporting Infrastructure	Day of the Figurines is implemented as a Java web application that runs as a servlet within Tomcat[1]. The Spring framework[2] allows the application to be split into a number of Java beans, controllers and views, and also supports communication between the beans. Spring controls access to the application by mapping incoming http requests from web interfaces to specific controllers, which in turn render data in the application as further web pages. In addition to this, the Quartz[3] scheduler provides support for regular, time-based events within the application. Figure 4 shows an overview of these supporting infrastructure components.  Tomcat servlet container Application interfaces Spring dispatcher servlets Application Logic Hibernate/ JDBC connector Equip2 MySQL Database Quartz scheduler **Figure 1 Supporting Infrastructure** The application makes extensive use of the Equip2[4] data sharing middleware platform. Equip2 provides a shared dataspace that distributes normal Java objects, by providing synchronous, database-like operations such as adding, matching, updating and removing, and also asynchronous change notifications with optional template based filtering and matching.

[1] http://tomcat.apache.org

[2] http://www.springframework.org

[3] http://www.opensymphony.com/quartz/

[4] For more information about Equip2 please contact: cmg@cs.nott.ac.uk

	Equip2 supports a number of different data space implementations with a common interface. The implementation used for Day of the Figurines is backed by Hibernate[5], an open source java object to relational database mapping system. This allows java objects to be persistently stored by Equip2 in a database such as MySQL, as used in conjunction with Tomcat in Day of the Figurines, while still providing the dataspace access operations described above. Using this system, Java beans are defined in a simple XML format to hold the application's state, and therefore the state of the game. These XML definitions are then parsed using XSLT in order to generate the actual Java beans, Equip2 helper classes, and the Hibernate mapping files that are used to persistently store the beans in the underlying database. These beans can then be used in conjunction with Equip2 operations throughout the application to provide a persistent state. The application and this supporting infrastructure are deployed to an instance of Tomcat with supporting MySQL database. Day of the Figurines runs on a dedicated webserver, where multiple Tomcat instances are accessed through an instance of the Apache webserver software, which maps the relevant url paths to the required Tomcat instance. The webserver is a duel-core machine with 4GB of memory running Redhat Linux, and is hosted at Nottingham.
Application Architecture	Figure 5 shows the architecture of the application as it sits within the infrastructure described above, in terms of the key components of the game application.  External SMS provider SMS Proxy SMS sender / receiver SMS parser Core game engine Message renderer Message aggregator Core game application **Figure 2 Application Architecture** The core of the Day of the Figurines application consists of five key components. Each component communicates with others by adding, updating and deleting Equip objects. Equip objects are also created and accessed by supporting interfaces, such as the Augmented Board and Spectator Client, and the Author, Operator and Orchestration interfaces, described in later sections. The SMS sending and receiving component, which provides the primary interface for interaction with the game to players, is split between two Tomcat instances, which communicate via http requests, enabling the core game application to be started and stopped without losing incoming SMS messages.

[5] http://www.hibernate.org/

	The SMS parsing component listens for new SMS messages received from the first component, and parses them into *Player Action* objects within the context of the game state, for example describing a destination that a player wishes to go to, or a Thing with which they wish to interact. These objects are added to the Equip2 dataspace. The core game engine component listens for *Player Action* objects, which it uses to modify the state of the game for example by triggering a player's movement between destinations, and handling the resulting encounters with other players and other content within the system. Changes in game state are also triggered by time, using the quartz scheduler. All changes in a players state, and in other elements of the overall game state, for example the creation or destruction of silos or the changing of destination descriptions, result in the creation of *Game Event* objects which, analogous with *Player Actions*, contain information about the change in state that has just occurred. These objects are added to the Equip2 dataspace. The Message Renderer component listens for particular *Game Event* objects as they are published in the Equip2 dataspace, and renders each one as a piece of readable text, which will form part of an SMS message sent to a player. The Message Aggregation component decides how, when and which of the rendered text pieces are sent to players, using several pieces to construct an SMS message, attempting to make the best use of the space available, while determining which messages can be delayed and which must be sent immediately. It adds completed SMS messages to the Equip2 dataspace, where they are listen for by the SMS sending and receiving component that pushes them out to players. The rationale behind this architecture is to separate the core game state mechanics and game logic from the sending and receiving of SMS messages and other interfaces, to the extent that these inputs and outputs of the system are operated independently of the core game.
SMS sending and receiving	As described above, the SMS sending and receiving component is split across two Tomcat instances; the instance in which the game application is running, and a dedicated second instance that acts as a proxy for incoming and outgoing messages. The proxy instance communicates with an external, commercial SMS delivery provider via http requests in order to send outgoing and receive incoming messages. It also communicates with a stub interface within the game application Tomcat instance to send and receive messages to and from the game application. The proxy application provides two functions. During normal operation, it forwards messages to and from the core game application. When the game is closed, that is out of normal game hours, then the proxy application replies to incoming messages with a default message stating that the game is closed, and when it is expected to reopen. Secondly, if for some reason the core application is not available, and incoming messages cannot be delivered, then the proxy application will cache the messages in Equip2 and retry until delivery is successful. Similarly, if the external SMS delivery provider is unavailable, the proxy application will cache and attempt to redeliver outgoing messages. This system ensures that if either the core game application or the external SMS provider are unavailable due to maintenance, or due to an unexpected occurrence, messages to and from players are not lost. In addition to this, the external SMS provider gives the ability to query the successful delivery of a message to a player's phone, as opposed to the phone being switched off or the SMS inbox being full. By providing an interface to this function, the SMS proxy application allows the game operators to check whether a message has not only been sent, but has successfully been received by a player. Finally, the proxy application provides monitoring and emulation functions via a web interface for testing and debugging. The interface displays all incoming and outgoing messages with information about the sender or recipient, and the delivery status of the message. It also allows an operator or developer to simulate sending and receiving messages to and from the core game application without having to use the external SMS provider, and therefore a real phone. A simulation may consist of a hand-crafted message, or a bulk upload of a large number of messages that are handled over a period of time.

SMS handling and parsing	The SMS parsing component attempts to map an incoming SMS message from a player onto a known game action, with a relevant in-game context, before this is passed to the core game engine that then acts upon it. To identify the action that a player is attempting to perform, the component uses a combination of direct string matching and edit-distance calculations in order to attempt to guess what a player has specified in the case that it is unrecognised. To interact with and modify the game state, a player specifies a command with, optionally, a number of arguments. Depending on the command, the argument may be a Destination they wish to go to, a Thing they wish to pick up, use or drop, or a Player they wish to find. Each command, and each instance of these game objects, has one or more *Command Alias* objects associated with it, each of which contains a string that can be used to identify a reference to the game object within the SMS message. An object may have more than one *Command Alias* associated with it, and these are authored in such a way as to attempt to pre-empt expected typographical errors when entering a command using a phone keypad. The originating phone number of each incoming SMS message is used to identify the player who sent it. Next, the message is parsed to identify the command that the player wishes to perform, by searching for a *Command Alias* that represents a command verb, such as GO, FIND or USE, within the message. If none are found immediately, then the message is parsed a word at a time, attempting to match the most likely *Command Alias* to the string using an edit-distance function within a certain threshold, so as to overcome possible typos that have not been predicted in advance. Based on the command verb that has been identified, the message is then parsed a second time using a similar mechanism to identify the relevant arguments of the message. However, the current context of the player and the command that they have specified are used to direct the parsing of unknown arguments, for example only matching against Things in the player's current location in the game when trying to pick something up, or only matching known Destinations when trying to go somewhere. This mechanism means that when *Player Action* generated from a parsed message is sent to the core game engine component, no further parsing is required as the information in the object has already been constructed within the context of the game.
Core Game Engine	The core game engine component is driven by incoming *Player Action* events received from the Equip2 data space notification mechanism, and by regular time events received from the Quartz scheduler. It outputs *Game Event* objects in the Equip2 data space where they are monitored by the Message Renderer component, and by other supplementary components, such as display and orchestration interfaces. The game engine maintains an evolving game state within Equip2, which consists of a collection of game objects representing each Player, a history of events that a Player has been involved in or experienced, such as Local Events, Dilemmas and Missions, a number of Silos that contain players and are in turn associated with Destinations, and a number of system events that are scheduled to happen at some time in the future, and trigger further state changes. The game engine is also backed by authored content, which is defined as a number of static objects within Equip2. This content is queried as events within the game engine are triggered, such as a player moving or interacting with a Thing, or time elapsing within the game. On each trigger, the game engine queries the Equip2 data space to determine which game content is relevant for the current time and destination, or in the case of an action triggered by a player, the current context of that player. This, combined with the current game state, informs the game engine of how to modify the game state. Each action, whether player or time instigated, produces one or more *Game Event* objects which inform the Message Renderer component what a player has experienced, and these are grouped into a session for each action. For example, when a player arrives at a Destination, a game event is created for them that states that they have arrived, and what Description is currently relevant for that destination, a game event for each Local Event, Dilemma or Mission that is in scope for that destination, and game events for any players currently in the destination that may have witnessed the arrival, if necessary.

Pacing and Aggregation	The Message Renderer sends SMSs to players according to two paradigms; every message to the system gets a timely response, and actions that directly affect a player will always result in a message to them. In order to create a slow response system, all *Game Events* that require a message to be sent to a player are broken up into two categories; instantaneous and delayed. Instantaneous events are those that directly, and immediately, affect the player's state, such as a player arriving at a destination, being in the subject of a local event, or being hit by another player. These events are all intermittent and generated either by the system or by another player's action. Other *Game Events* result in a delayed response, namely those generated by a player's message to the system, such as a PICKUP or GO command, or when they send a malformed message. When the criteria for the Renderer to send a player a message are met, the Renderer searches for other relevant information it can include, if possible, in the message. This is achieved by searching through the *Game Events* that relate to the player and to their current location that are not worthy of an individual message, for information that can be included within the message. In some cases it is an update of who is currently co-located with player, if new players have arrived since they last interacted with the system, the presence of things dropped by other players, or any changes to player's health. Specific details regarding pacing and aggregation can be found in the appendix.
Message Rendering	Once the Renderer has a *Game Event* that spawns a message, and any other information to try to include in the message, it generates the text. Each piece of a message has an associated *Message Specification*, which includes all the text for the message, or a *Message Element* that is used to fill the message with *Player*, *Destination* or *Thing* relevant content. The *Message Element* is a key word that is mapped to specific content in the system In the simplest case a *Message Element* is a piece of static text, whereas in other cases it could map to the player's nickname or listing nearby players or things. A *Message Element* may result in no characters being inserted in the message if there is nothing to list or not enough space to list even one of the things or players present, or it may be a rich list of the other co-located players, including their description and what they are holding, according to how many characters are still available in the growing SMS. To create the final text for the SMS, some *Message Specifications* can be tiled together, and some *Message Elements* will have others nested inside them. All *Message Specifications* and *Message Elements* can be edited or replaced while the system is running, allowing the authors to change the way in which any message the system sends is rendered. A detailed specification of *Message Specifications* and *Message Elements* can be found in the appendix.

4 Game Content

Features	Day of the Figurines Game Content
Introduction	This section gives an overview of how the content of Day of the Figurines that players experience is authored. A full specification of the content types that are authored can be found in the appendix, and the content authored for the current game can be found on the live application website[6].
Authoring Process	As described in the previous section, authored content in Day of the Figurines is made up of static Equip objects, which the core game engine queries to discover which content is appropriate to the current context of a player as time passes or the player moves through the town. A number of interfaces are provided for creating, editing and reviewing these Equip objects, and are documented in detail in D7.5. Briefly, they consist of a generic, automatically generated set of web-forms that allow modifying all aspects of each content type together with XML serialisation for up and downloading - enabling backup and transferral, a configured view that displays the same web-forms, but with friendly names and hiding underlying system aspects of the types that do not require authoring, and a high-level authoring applet that displays all currently authored content by space and time. These interfaces are integrated within the Day of the Figurines web application. In practice, the web application is deployed twice to the production webserver. One instance runs the game with a static content set, while the second is used by the authors to add, update and review content. When this content has been tested and is deemed to be complete, it is transferred to the live game instance by dumping as XML from the authoring instance and uploading to the live ins tance. The interfaces are backed by a set of generic Spring controllers, which provide access to the Equip dataspace for the creation and manipulation of Equip objects. The authoring applet is backed by a custom controller also provides access to the datas pace. This system has three major advantages to offline authoring. Firstly, it allows multiple authors to work on the same set of content on the authoring server, without affecting the live game, and removing the need to merge content that may have been authored against an older snapshot. A set of sanity-checking scripts may be used to ensure that the content does not contain mistakes. Secondly, it allows a series of XML snapshots of the content to be created and saved, meaning that if at any time content needs to be rolled back to a known good version, it can be re-uploaded. Finally, the generic nature of the web-forms mean that if for some reason the java types need to be changed, the authoring interfaces will automatically reflect these changes.
Content Requirements	Authored content within Day of the Figurines can be split into three logical groups. This content is static, in that it is only queried by the core game engine in order to update the transient state of the game, rather than being modified by the game engine as events occur. The first group consists of static content that must be authored first so as to provide a base upon which other content is authored. At present this consists of Destinations, which are created as Equip objects within the database, with the intention that future content can be tied to a specific Destination and time, although time is obviously a more abstract representation. The second group consists of content that is tied to a destination, or group of destinations, and a particular time or period of time. This information is used by the core game engine to determine when a particular piece of content should be triggered, and this content generally affects the state of the game or players. This content consists of Destination Descriptions, which may change over time, Local Events, Dilemmas and Missions, and

[6] http://holt.mrl.nott.ac.uk/test-dof2/author/index.htm

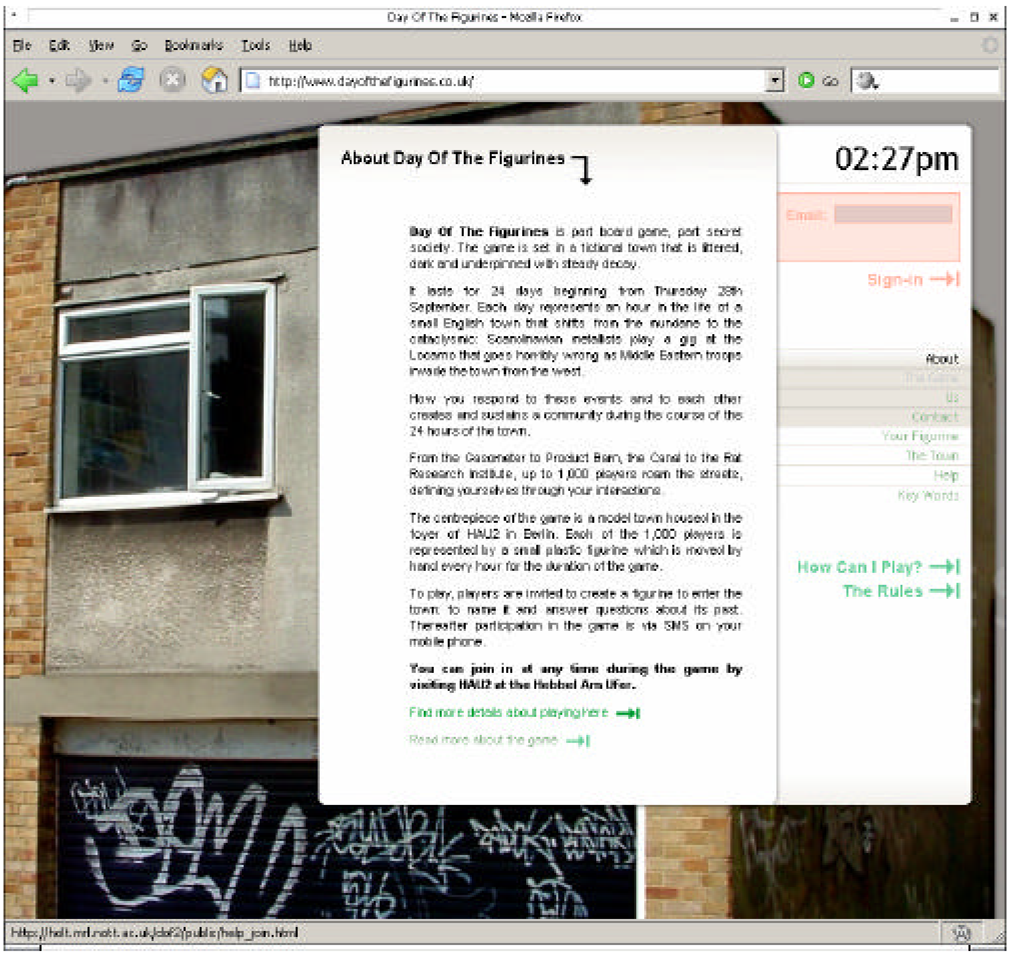
	Things. Each of these pieces of content has an associated Scope, and the Scope defines when and where the parent should be active within the game. A number of Scopes may be allocated to one particular content object, allowing, for example, a Local Event to span multiple time frames or Destinations. Each Scope is associated with one other object in a one-to-one relationship for database query efficiency, as opposed to a one-to-many relationship of the parent object to multiple Scopes. In a similar fashion, multiple Mission_Criteria objects are associated with each Mission. The third group consists of message related content, and includes Command Aliases which, similar to Scopes, provide a mapping from text strings to specified game state objects, and Message Elements and Message Specifications for the construction of outgoing messages. Message Elements and Specifications are static in that each serves a predetermined purpose within the game engine. To facilitate this, a set of template Elements and Specifications are initially uploaded to the dataspace, where they can then be edited by the authors using the interfaces provided.
Content Usage	When the main Day of the Figurines web application is loaded, portions of the authored content is loaded and cached in memory to reduce the number of database queries that occur on each time or action trigger, and thus increase efficiency. This includes as much of the static content as possible, for example Command Alias and Mission Critera objects. The code that handles this function also registers listeners that listen for changes of the content within the Equip dataspace, and modify the memory cache accordingly so that the system does not need to be restarted to modify content. With this in mind, this system allows authors to respond to small errors in the content, for example typos, or to turn off content by removing Scopes, while the game is running in response to any problems that may occur. Finally, at load time the game engine generates more transient game state objects based upon the static authored content in the dataspace. These include the Thing Instance objects that, similar to Scopes, define which Player or Destination is currently holding a Thing. Based on the spawning rules in the static Thing class, the game engine may create new Thing Instances in Destinations if the system is being run for the first time.
Content Helpers	A number of supporting helper functions are provided to aid the author in creating large amounts of content. Each piece of static content may be associated with a Narrative Arc, which again is a unique object within the Equip dataspace. These objects are not used by the core game engine for running Day of the Figurines, but they are used by the authoring applet in order to logically group a number of content objects. This functionality is used to filter these groups within the applet view, giving the author a clearer view of how the group is spread over time and Destinations. A set of automated sanity-checking functions are provided to check whether the authored objects are logically correct within the game logic of Day of the Figurines. This tool checks all content within the system, and displays logical inconsistencies, for example Scopes that are not assigned to a parent object, or conversely content that does not have a Scope, Dilemmas without responses or a default response, or text that is more than 160 characters in length.
Content Breakdown	As stated previously, content that has been authored for the current release of Day of the Figurines can be viewed on the live authoring server website. In summary, the current content consists of: ? 50 Destinations ? 228 Destination Descriptions ? 27 Dilemmas ? 70 Dilemma Responses ? 251 Local Events

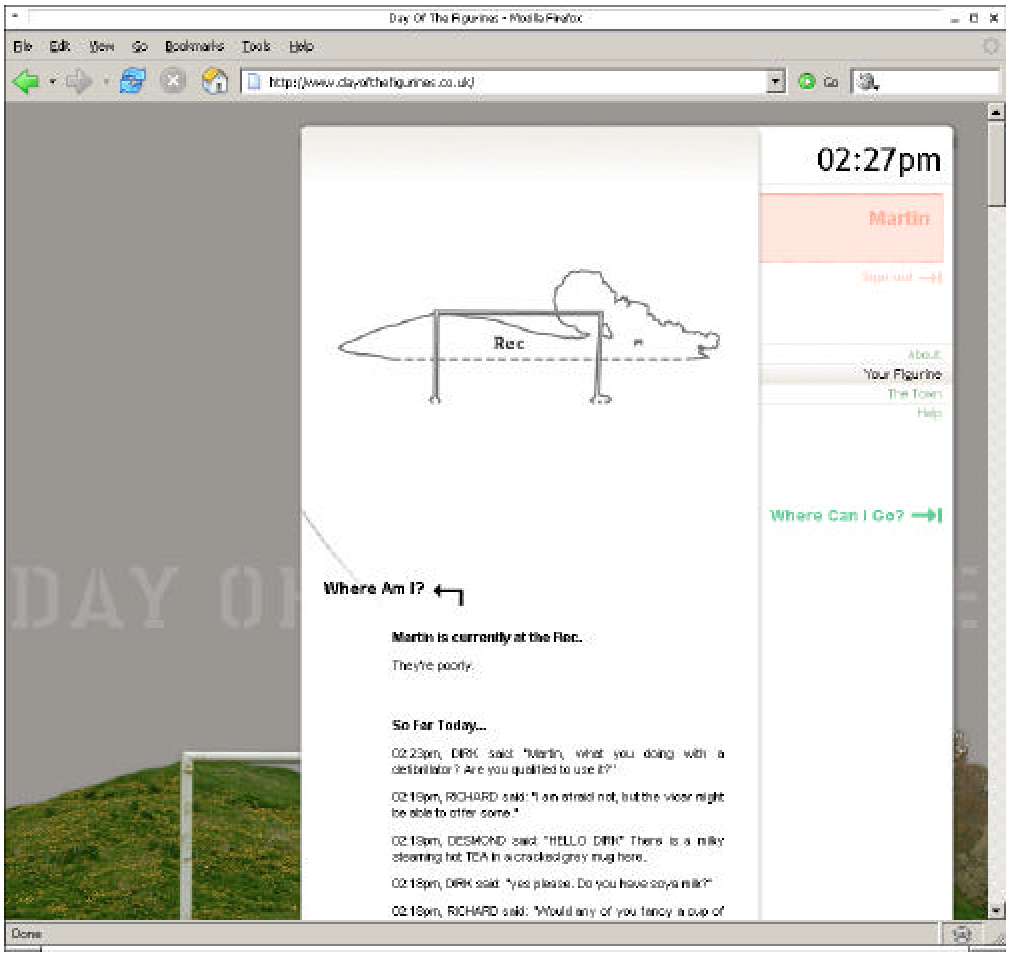
	? 78 Message Elements ? 93 Message Specifications ? 857 Command Aliases ? 17 Narrative Arcs ? 26 Missions ? 77 Mission Criteria ? 33 Things ? 826 Thing Instances ? 1006 Scopes
Online Content	A website located at www.dayofthefigurines.co.uk provides information and instructions on how to play the game. It also allows registered players to login and see where they are in the town, and gives a history of messages that they received.

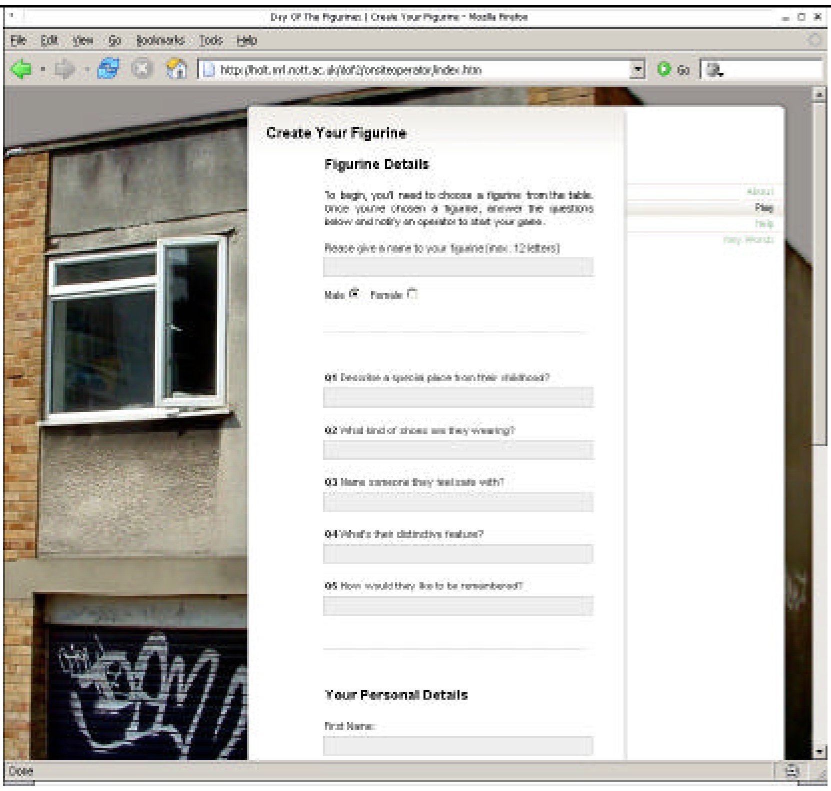
5 Technology & Devices Used

Technology/Device	Amount	Mode of play	Function
PC Workstation	2	Players, Operators	Displays an interface for adding new players to the game, and allowing a player to describe their figurine with operator validation.
PC Workstation with quad-head graphics card	1	Operators	Displays an interface that enables an operator to start and stop the game and send start-of-day messages. Runs the spectator interface for the Augmented Board Runs the Augmented Board software client
Augmented Board	1	Spectators, Operators	Receiving and sending text messages
Spectator Interface	1	Spectators	A display attached to the Augmented Board that displays incoming and outgoing SMS messages.
Mobile Phone	1 per player	Players	Sending and receiving SMS messages to and from the game.
SMS Gateway	1	Players	Commercial external SMS gateway service, forwards messages between the game application and players.
Webserver	1	Players, Operators, Authors	Runs the main game web application containing the core game engine, supports authoring, operator and orchestration interfaces and the public website.

6 User Interfaces

Name	**Purpose**
Player website	The mobile phone and, in particular, SMS, provide a narrow channel for relaying game information between players and the game. The Day of the Figurines player website provides an additional resource for players that is not bandwidth limited as SMS, allowing additional information such as hints and tips, instructions and a historical view of a player's movements to be displayed. The player website is backed by a controller and JSPs that give it access to the main Equip dataspace, allowing it to pull out historical and current player information once a player has logged in.  Day Of The Figurines - Mozilla Firefox http://www.dayofthefigurines.co.uk/ **About Day Of The Figurines** **Day Of The Figurines** is part board game, part secret society. The game is set in a fictional town that is littered, dark and underpinned with steady decay. It lasts for 24 days beginning from Thursday 28th September. Each day represents an hour in the life of a small English town that shifts from the mundane to the cataclysmic: Scandinavian metalists play a gig at the Locarno that goes horribly wrong as Middle Eastern troops invade the town from the west. How you respond to these events and to each other creates and sustains a community during the course of the 24 hours of the town. From the Gasometer to Product Barn, the Canal to the Rat Research Institute, up to 1,000 players roam the streets, defining yourselves through your interactions. The centrepiece of the game is a model town housed in the foyer of HAU2 in Berlin. Each of the 1,000 players is represented by a small plastic figurine which is moved by hand every hour for the duration of the game. To play, players are invited to create a figurine to enter the town, to name it and answer questions about its past. Thereafter participation in the game is via SMS on your mobile phone. **You can join in at any time during the game by visiting HAU2 at the Hebbel Am Ufer.** Find more details about playing here Read more about the game 02:27pm Email: Sign-in About The Game Us Contact Your Figurine The Town Help Key Words How Can I Play? The Rules http://holt.mrl.nott.ac.uk/dof2/public/help_join.html **Figure 3 The Player Website**

	 Day Of The Figurines - Mozilla Firefox File Edit View Go Bookmarks Tools Help http://www.dayofthefigurines.co.uk/ 02:27pm Martin Sign out About Your Figurine The Town Help Where Can I Go? Rec Where Am I? Done **Martin is currently at the Rec.** They're poorly. **So Far Today...** 02:23pm, DIRK said: "Martin, what you doing with a defibrillator? Are you qualified to use it?" 02:19pm, RICHARD said: "I am afraid not, but the vicar might be able to offer some." 02:19pm, DESMOND said: "HELLO DIRK" There is a milky steaming hot TEA in a cracked grey mug here. 02:18pm, DIRK said: "yes please. Do you have soya milk?" 02:18pm, RICHARD said: "Would any of you fancy a cup of **Figure 4 Player History**
Operator Interfaces	The operator interface serves two main functions; adding new players to the game and manipulating the state of the game, for example starting and stopping the game, or sending custom messages to players in response to help requests. The interface is constructed around a set of web pages and JSPs backed by custom controllers that manipulate the internal Equip game state. The game start and stop pages guide the operator through the process of opening or closing the game and the SMS proxy with a set of buttons indicating the correct sequence that must be performed. Further interfaces display help requests from players that require a custom response from the game operators, and show statistics on the overall health of the game using automatically populated tables and graphs. These also provide access to more detailed views on individual players and the delivery status of outgoing messages.


Day Of The Figurines | Create Your Figurine - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://holt.mrl.nott.ac.uk/dof2/onsiteoperator/index.htm
Go
About
Play
Help
Key Words
Done

Create Your Figurine

Figurine Details

To begin, you'll need to choose a figurine from the table. Once you've chosen a figurine, answer the questions below and notify an operator to start your game.

Please give a name to your figurine (max. 12 letters)

Male [✓ ◉] Female ◯

Q1 Describe a special place from their childhood?

Q2 What kind of shoes are they wearing?

Q3 Name someone they feel safe with?

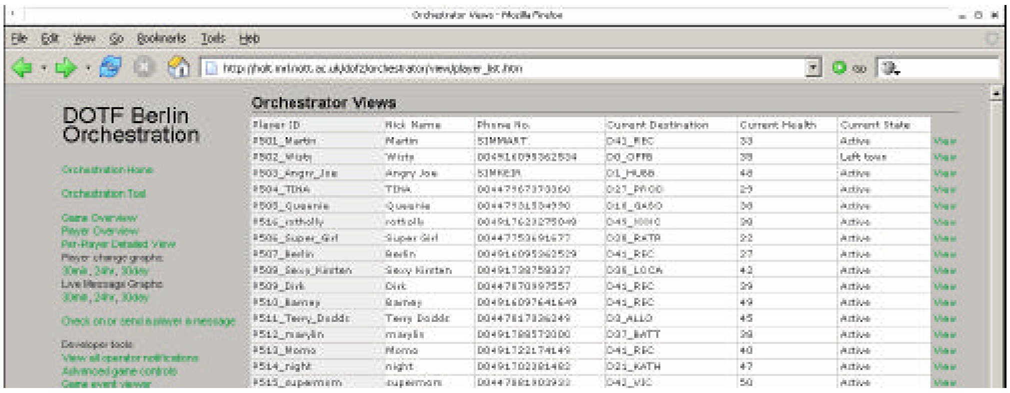
Q4 What's their distinctive feature?

Q5 How would they like to be remembered?

Your Personal Details

First Name:

Figure 5 Adding a Player


Orchestrator Views - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://holt.mrl.nott.ac.uk/dof2/orchestrator/views/player_list.htm
Go

DOTF Berlin Orchestration

Orchestration Home

Orchestration Tool

Game Overview
Player Overview
Per-Player Detailed View
Player change graphs:
30min, 24hr, 30day
Live Message Graphs:
30min, 24hr, 30day

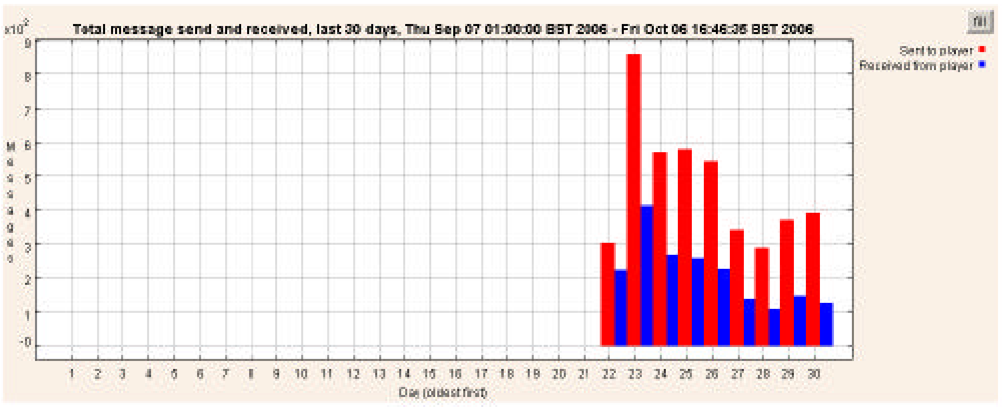
Check on or send a player a message

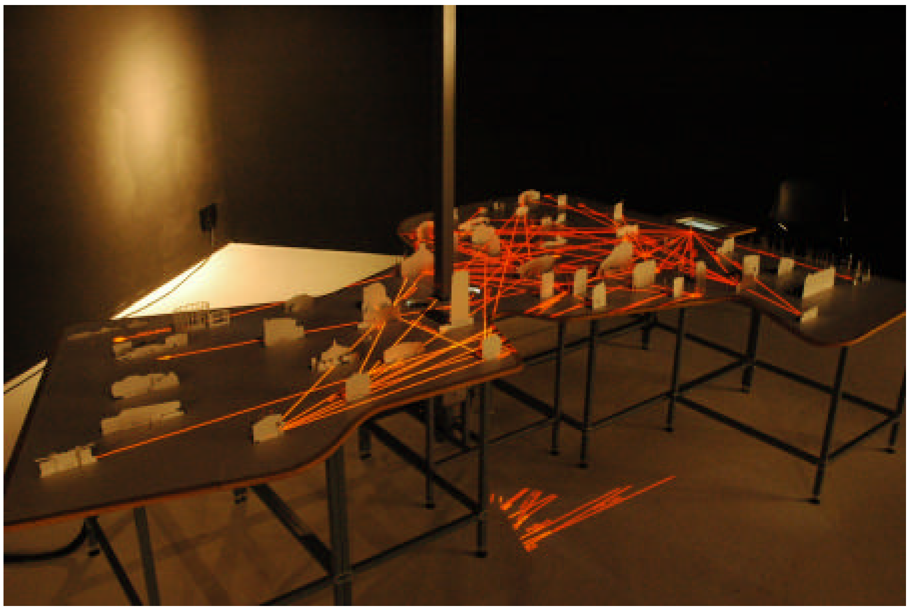
Developer tools:
View all operator notifications
Advanced game controls
Game event viewer

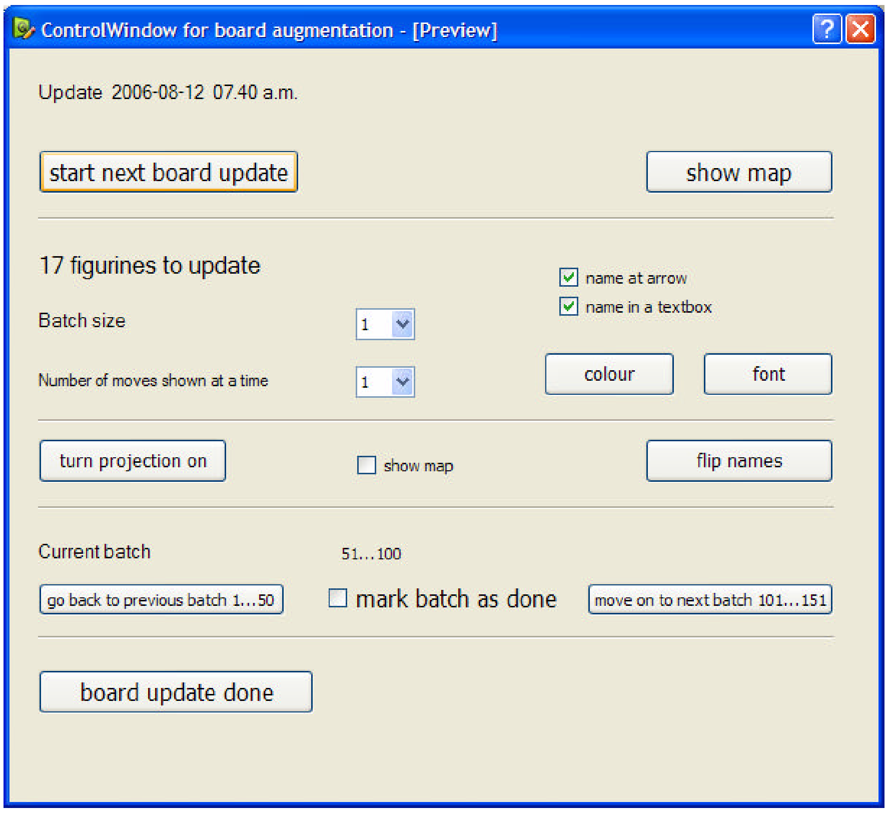
Orchestrator Views

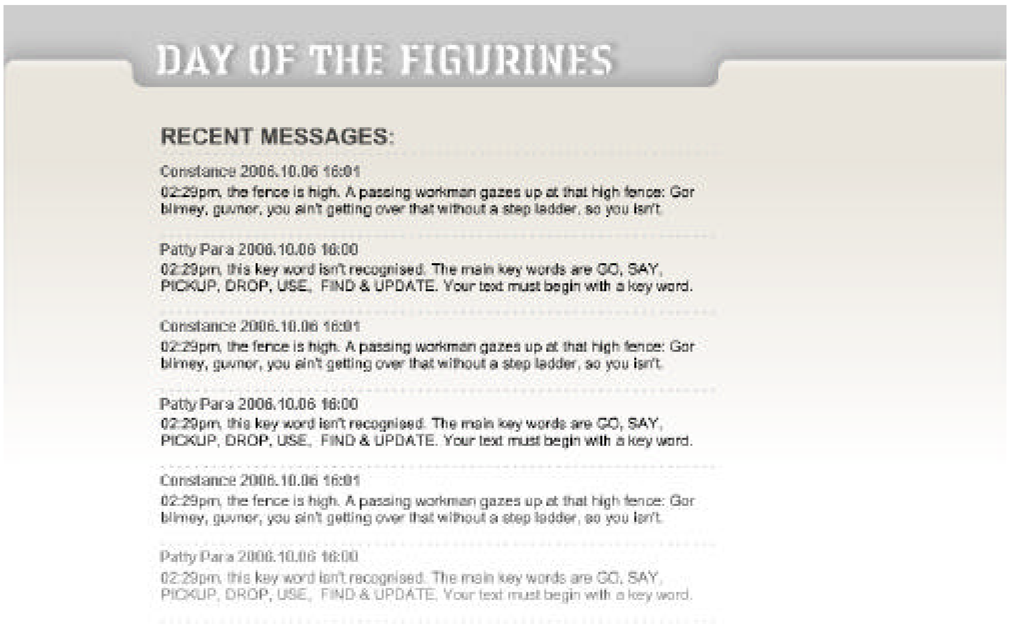
Player ID	Nick Name	Phone No.	Current Destination	Current Health	Current State	
P501_Martin	Martin	SIMMART	D41_REC	33	Active	View
P502_Wisty	Wisty	00491609362534	D0_OFFB	38	Left town	View
P503_Angry_Joe	Angry Joe	SIMKEIR	D1_HUBB	48	Active	View
P504_TINA	TINA	00447567370360	D27_PROD	29	Active	View
P505_Queenie	Queenie	00447931534930	D18_GASO	38	Active	View
P516_rothofly	rothofly	00491762327504	D45_MMIC	39	Active	View
P506_Super_Girl	Super Girl	00447753691677	D26_RATR	22	Active	View
P507_Berlin	Berlin	00491609S362529	D41_REC	27	Active	View
P508_Sexy_Kirsten	Sexy Kirsten	00491738758337	D36_LOCA	42	Active	View
P509_Dirk	Dirk	00447870897557	D41_REC	29	Active	View
P510_Barney	Barney	00491609764164	D41_REC	49	Active	View
P511_Terry_Dodds	Terry Dodds	00447817826249	D3_ALLO	45	Active	View
P512_marylin	marylin	00491789573000	D37_BATT	38	Active	View
P513_Momo	Momo	00491722174149	D41_REC	40	Active	View
P514_night	night	00491702381482	D21_KATH	47	Active	View
P515_supermom	supermom	00447881303932	D41_VIC	50	Active	View

Figure 6 , Player Details; Below, Message Statistics



Author interface	The authoring interface, as described previously, consists of an automatically generated set of web pages that enable the adding, editing and deleting of Equip objects. This is supported by a game specific configuration that only displays necessary fields with friendly names, and an overview applet for viewing authored content by time and destination. A full description of the authoring interface with screenshots and user manual can be found in D7.5.
Game board interface	The game board is a shaped 2x2 meter table that shows the town of Day of the Figurines with its destinations. Stylised cut-outs of the destinations are folded up out of the metal surface. Each player has a small figurine with a magnetic base, which an operator moves on the game board when the player moves within the game. Bellow the game board two projectors are mounted, two holes in the centre of the game board allow the images from the mounted projectors be projected onto a mirror placed above the table the reflection of which is used to display the movements of the figurines. These movements are shown as arrows projected onto the board, and the name of the player is also displayed to aid the operator in identifying the correct figurine.  **Figure 7 Augmented Board** The game board is purely an output interface that is updated by the operators, and also serves as a spectator interface for players in the venue. To update the game board the operators start the update process via a control window and move the figurines according to the projected arrows. An arrow goes from the current position of the figurine on the board to the new position of the figurine on the board. Operators can update the board any time, although typically they will update the game board once an hour. The figure below shows the control window for the board augmentation. When the button "start next board update" is pressed the number of figurines that need to be updated is queried from the game server and displayed. The operator can decide how many figurines he/she would like to update at a time. The augmentation is started by pressing the button "turn projection on". The operator uses a wireless Bluetooth mouse to control the display of the arrows when he/she is at the game board. Via the control window the operator can configure how the augmentation is displayed, including the number of arrows displayed at a time as well as the layout and appearance of the arrows (e.g. change font, font size, colour, etc.).

	 ControlWindow for board augmentation - [Preview] start next board update show map colour font turn projection on flip names go back to previous batch 1...50 move on to next batch 101...151 board update done Update 2006-08-12 07.40 a.m. 17 figurines to update Batch size 1 Number of moves shown at a time 1 ✓ name at arrow ✓ name in a textbox ☐ show map Current batch 51...100 ☐ mark batch as done **Figure 8 Augmentation Control** The game board software connects to the main game web application using the IPerG positioning service. Whenever a player moves within the game, the core game engine updates the instance of the positioning service with information about where they moved from and to, which can then be queried by the board software when the board is updated.
Spectator Interface	The spectator interface is a screen embedded into the side of the game board. It runs a Flash movie that displays the most recent messages that have been sent and received by players. The Flash movie polls for new messages by making an HTTP request to a custom controller in the main game web application.



RECENT MESSAGES:

Constance 2006.10.06 16:01
02:29pm, the fence is high. A passing workman gazes up at that high fence: Gor blimey, guvnor, you ain't getting over that without a step ladder, so you isn't.

Patty Para 2006.10.06 16:00
02:29pm, this key word isn't recognised. The main key words are GO, SAY, PICKUP, DROP, USE, FIND & UPDATE. Your text must begin with a key word.

Constance 2006.10.06 16:01
02:29pm, the fence is high. A passing workman gazes up at that high fence: Gor blimey, guvnor, you ain't getting over that without a step ladder, so you isn't.

Patty Para 2006.10.06 16:00
02:29pm, this key word isn't recognised. The main key words are GO, SAY, PICKUP, DROP, USE, FIND & UPDATE. Your text must begin with a key word.

Constance 2006.10.06 16:01
02:29pm, the fence is high. A passing workman gazes up at that high fence: Gor blimey, guvnor, you ain't getting over that without a step ladder, so you isn't.

Patty Para 2006.10.06 16:00
02:29pm, this key word isn't recognised. The main key words are GO, SAY, PICKUP, DROP, USE, FIND & UPDATE. Your text must begin with a key word.

Figure 9 Spectator Interface

7 Appendix

7.1 Game Model Specification

7.1.1 Destinations & Silos

Destinations

There are 49 'normal' destinations, plus:

? Hub

? 'offboard'

Each destination is subdivided into any number of Silos. Every player in a destination is in exactly one silo of that destination. Each silo has a nominal maximum capacity, but this may be exceeded (e.g. find or carry(?)).

Each destination may be divided into an inside and an outside, but some destinations will only have an 'outside' (even if it is inside :-). So...

? each silo is either inside or outside but not both

? a thing may be inside or outside but not both

access to the 'inside' may be restricted, e.g.:

? missions (see later)

? open/closed (time)

? [not destination capacity - this has been removed]

Silos

Silos are dynamically generated for destinations if required. Each destination has a preferred silo size. [Currently each destination has a silo template - is this just an implementation detail?]

Game state

Silo:

? Destination

? inside/outside

? capacity

? current number of players present

? description (ID, short, not long) - Hub silo only

Destinations descriptions and open/close

Each destination each a set of destination descriptions which allow the system to determine at any moment of game time the inside (if accessible) and outside descriptions. Each description comprises:

? text* of the outside (optional if open) and inside (optional if closed) description

? earliest use time, latest use time

? open/closed flag (causes destination to open/close as appropriate)

[* to be clarified wrt messaging, e.g. short and long forms]

Offboard

Offboard has silos of size 1 only. Otherwise it is a normal destination except that you cannot go to offboard (no alias). E.g. 'find' would work, but people presumably won't use it?!

The Hub

Players visit the hub only when going to some other specified destination. The hub contains unbounded number of Silos. The hub never 'opens'. The hub can have many different descriptions at any moment of game time. Therefore each hub silo (only) has its own current description chosen from those currently valid for the hub when the first player enters it.

See Player Movement (below).

7.1.2 Things

Game state

A (particular) thing (instance):

- is either:
  - being carried by a particular player or
  - in a non-hub destination, ins ide or outside (choose one only)
  - in a hub silo
- identifies its Thing Class (Factory/whatever)

A Thing class (factory/whatever):

- has description(s) - [under discussion because of plurals]
- has type:
  - clothing (always [and only] affects yourself)
  - food (always [and only] affects yourself)
  - medicine (affects incapacitated other if present)
  - other (affects other if present)
- specifies Thing life-cycle:
  - spawn point: destination, inside/outside flag
  - target instance/spawn count (at start of game)
  - actual spawned count
  - destroyed outcome: disappear from game or return to spawn point (if used or if abandoned in a hub silo)(??or potentially after a long time-out if "abandoned")
- specifies Thing use:
  - [whether it affects someone else, determined from type]
  - effect of use on using player using if no-one else active is present or doesn't affect someone else ('effect') (see below)
  - effect of use on using player using if no-one else active and incapacitated is present or doesn't affect someone else ('effect') (see below)
  - effect of use on using player using if someone else active present and affects someone else ('effect') (see below)
  - effect on (active) subject if it affects someone else ('effect') (see below)
  - 'effect' on same-silo (active) bystanders (by definition someone has to be present) (probably just messages)
  - (optional) 'effect' on same-destination (in/out) (active) bystanders [is this a mission thing]

    - Note: special cases (Note: destination-specific behaviours are handled by missions)
    - effect on thing:
        - no effect or
        - destroyed (see destroyed outcome, above)

An 'effect', e.g. of using is defined by:

- change of health:
    - +/- amount
    - Maximum
    - minimum
- change of description:
    - kind: no change, replace, append
    - text
- description (text) for message(s)
- plus, generalising for local events, dilemmas, missions, to include:
    - move outside in current destination
    - move inside in current destination

7.1.3 Local event

Game state

Each local event is characterised by:

- effect
- list of event scopes defining time(s) and destination(s) when/where dilemma can be allocated

An event scope is characterised by:

- a begin time
- an end time (which may be the same as the begin time)
- a single destination

A player receives each local event once; having received a local event is recorded by the associated game event (player receives local event). Local events are only delivered to players in state 'active'.

An event scope which becomes valid (reaches begin time) causes the associated local event to be delivered to all players in that destination (if they have not already had it).

An event scope with end time = begin time is only delivered to players in that(those) destinations at that moment. Otherwise a player arriving after begin time (inclusive) and before end time (inclusive) is delivered the local event (if they have not had it already).

7.1.4 Dilemma

Game state

Each authored dilemma is characterised by:

- dilemma allocation effect
- number of response options
    - each of which is an effect (see Thing use, above):

- ? identity of default response option
- ? [dilemma timeout is global]
- ? list of event scopes defining time(s) and destination(s) when/where dilemma can be allocated

Each dilemma as allocated to a single player is characterised by:

- ? player
- ? authored dilemma
- ? responded flag
- ? identity of response (if responded)
- ? dilemma timeout (if not yet responded)

Dilemma allocation

Caused by entering or (if not on a mission) leaving the hub - see Player movement.

Also caused as a custom result of some action(s) in a mission (see Mission).

7.1.5 Missions

Game state

An authored mission is characterised by:

- ? name
- ? time/destination scope - list of event scopes
- ? min. previous mission count
- ? requires player in need flag
- ? allocation effects:
  - o to player
  - o to player in need (if present)
  - o to bystander(s) (optional)
- ? max timeout for whole mission => failure?!
- ? mission description text
- ? set is mission-specific user action over-rides, each characterised by:
  - o trigger - player action/event (mask)
    - ✍ player
      - ? picking up a specific thing (class)
      - ? using a specific thing (class)
      - ? arriving at a specific destination
    - ✍ or player in need action
      - ? [picking up a specific thing (class)]
      - ? [using a specific thing (class)]
      - ? arriving at a specific destination
      - ? becomes active (??!)
      - ? dying
      - ? having health > some value

      - leaving town
    - or mission timeout
  - required context (criteria), all of:
    - player carrying thing (class) (optional)
    - player current destination (optional)
    - player current silo contains (active) player in need (optional)
    - player current silo contains at least N (active) players in addition to player (may include player in need) (optional)
  - rank/priority (for conflict resolution)
  - ends mission flag
  - if satisfied...
  - custom effects (if arrival, pick up, player in need dying, player in need recovering, player in need leaving town):
    - on player
    - on player in need
    - on bystander
    - on bystanders in same destination (in/out) but different silo
  - custom thing use effects (if thing use):
    - effect on thing:
      - no effect or
      - destroyed (see destroyed outcome, above)

An instance of a mission as performed by a single player is characterised by:

- player identity
- mission identity
- status: current, completed, timeout
- expires game time

Mission allocation

Caused by entering or leaving the hub - see Player movement. Note additional constraints on allocation to a particular player:

- not had this mission before
- minimum previous mission count
- existence/identification of a 'player in need' (if required by mission)

7.1.6 Player

Game State

A player (figurine):

- is in one of the following major states:
  - new - in process of registration
  - playing
    - resting (or non-playing time)
    - active

    - left town
    - dead
- a playing player is always in exactly one silo of one destination (dead and "left town" players are moved to their own silo in the destination "off board")
- has a description
    - player's provided description
    - current description, combining
        - original description which may be replaced during the game by missions, dilemmas and local events (may be constructed from history of events ('replace'))
        - list of additional appended description fragments (which may be added to during the game due to missions, dilemmas and local events ('append'))
- has a health level (0 if dead)
    - dead – 0
    - incapacitated - 1-20 (cannot move, find, carry people or things other than clothes or use; can talk)
    - mobile - 21-100
- may be carrying at most one thing or player [do we have to have carrying players? can it be handled as a special case in missions? ask Blast Theory how sad they will be if this is not present]
- has a gender (used in message stuff only?!)
- has a mobile phone number (used for sending/receiving messages)
- a list of 'friends' (other players), quantified by number of meetings and number of messages said to and heard from (used in Silo allocation)
- zero or one current dilemma which they have yet to respond to
- a list of dilemmas previously experienced and the player's response to each (used to ensure not given duplicate dilemmas (and for operator and player web pages))
- zero or one current mission (cannot be on more than one mission at the same time because they would interfere/conflict)
- a list of missions previously performed and some(?) information about what happened (see missions)
- a list of game events affecting this player (e.g. description modifications)
- a list of local events that this player has experienced (e.g. troop envading local event)
- a list of action requests made by the player (i.e. their parsed messages)
- preferred play times
- resting state (true/false), and (if resting) time to become active
- if active in non-preferred play time then wall clock time to rest again (1 hour Wall clock play timeout)
- award dilemma on arrival at destination flag
- award mission on arrival at destination flag
- (hub timeout - probably in a system event or similar)

Non-game State

A player has other information not used in game play:

- answers to the questionnaire questions (not used in game play, other than by operators reading them and constructing custom messages/saying though operator-controlled characters?)

? associated Person details (not used in game play)

? activity level (not used in game play)

? destinations visited (maybe this is just game events now??)

? says sent/received (maybe this is just game events now??)

? SMS messages sent/received

Resting

Being in non-preferred play time is automatic resting. (What happens if you send a message less than an hour before the end of your preferred play time? is resting deferred for an hour or not?)

A resting player is:

? In playing sub-state 'resting' (not 'active')

? Not visible to other players

? Still occupies a place in their silo

? Still carries the same object

? Cannot be found - if otherwise successful (same destination, considering closed) finder receives resting notification

? Cannot be affected by use of things

? Are not considered to be an observer for local events, say, use of things (no effects are applied to you)

If a player has a current dilemma, their dilemma timeout is in active state game time. Therefore it is frozen while they are resting.

If a player has a hub timeout and is in a multi-player hub:

? and is on their own: they are moved to a single-player silo and their timeout is retrospectively reset to be a single player timeout and frozen (since it is measured in active game time)

? and is with someone else: they are moved to a single-player silo as above, and the player they are with is moved on to their final destination as if the resting player had moved on (see Player movement, go when in a hub silo)

Note: this skews hub experiences away from chat, but this should be OK if preferred play slots are large compared to chat timeouts (please:)

A player starts resting:

? when they reach the end of one of their preferred play periods

? if they send a REST [N] message (default N is 2 (hours))

? if they are in a non-preferred play period but are active (see below) and their activity timer expires

When a player starts resting:

? their state is changed to 'resting' ('player starts resting' event)

? bystanders receive a 'player starts resting' event and observer events

? timers are frozen as appropriate (see above)

A player stops resting:

? when they reach the start of one of their preferred play periods (if after any explicit REST timeout)

? when any explicit REST timeout expires if during a preferred play period

- ? if they send a message (other than REST) while resting

When a player stops resting:

- ? their state is changed to 'active'
- ? generate a 'player stops resting' event (causes recap/wakeup message to player) and observer events
- ? any hub and dilemma timers are reinstated
- ? may trigger custom mission effects if player is payer-in-need in any mission(s)

User changes to player's "resting times" made on the player's public website do not take effect until the following day.

Actions

Can only performed if state is playing. If an action (other than Rest) is attempted while resting then Player is deemed to be active for the next 1 hour (see Resting, above).

[update game activity?!]

Commands

Command	**parameters**	**defaults**
GO	destination	none
FIND	player	none
PICK-UP	thing	none
USE	thing	thing currently carried
PICK-UP-AND-USE	thing	none
DROP	thing	thing currently carried
SAY	text	none
UPDATE	-	-
REST	hours	2 hours
HELP	text	""
A/B/C/...	-	-
LEAVE-TOWN	-	-

Movement (Go, Find)

Cannot move if incapacitated (except if a player is incapacitated while in the hub they will still be moved on their final destination when their hub timer expires). If attempted results in failure to move game event :-)

Go to another destination when in a non-hub destination

If waiting for a dilemma response then apply default response.

Move immediately to the hub (generate leave event). Experience the fun allocation process... (all numbers adjustable)

- ? 20% single player...

    - o 20% - nothing happens, i.e. new/empty capacity 1 silo with (new) random hub description in current scope, short timeout, arrive event
    - o 80%...
        - ✍ 50% - allocated a dilemma, which is in scope for current time, for the hub, and which has not been given to this player before. ??could split here for
            - ? hub arrival dilemma, i.e. new/empty capacity 1 silo with (new) random hub description..., long (default dilemma response) timeout (see below)
            - ? if player is NOT on a mission, then hub depart/destination arrival dilemma, i.e. hub experience is like 'nothing happens', but arrival at destination triggers dilemma (Note: dilemma cannot be chosen yet because player may use go in the hub to go somewhere else)
        - ✍ 50% - if not already on a mission(!), allocated a mission at random, which is in scope for current time, for the hub, and which has not been given to this player before. ?? could split here for
            - ? hub arrival mission, like 'nothing happens' but are allocated mission
            - ? hub depart/destination arrival mission, like 'nothing happens' but will be allocated mission on arrival at final destination (Note: mission cannot be chosen yet because player may use go in the hub to go somewhere else)
        - ✍ if no relevant dilemma/mission available for this player then falls back to 'nothing happens'
- ? 80% - multiplayer meeting, i.e. first look for a non-full non-empty two player silo (don't worry about friends, there cannot be more one at a time), and
    - o if found place player in it and set chat timeouts for both players
    - o else choose a new/empty two player silo, fix description, allocate player to it, set arrive/wait for meeting timeout

Hub timer set according to experience type. When expired, player progresses to final destination.

Timer changed if:

- ? in case of saying in multiplayer meeting, extended (see notes)
- ? after responding to dilemma reset to short wait

Special cases:

- ? dilemma expires without responding, imposes default response and resets hub exit timer as per player response to dilemma (short wait then go to final destination)
- ? player in two player hub whose partner leaves (due to go or their timer expiring) more on to final destination immediately irrespective of own timer.

Go to the same destination when in a non-hub destination

If player is outside and destination is open then do normal arrival (inside) process.

Else (if player is already inside or destination is closed) then already there non-event.

Go when in the hub

If currently awaiting dilemma response then first process default dilemma outcome.

Move immediately to destination (see below).

Note: if in multiplayer chat then other player also moved on (see above).

Arriving at a non-hub destination

Caused by hub timeout or go when in the hub...

If player is on a mission with a player in need and player in need is in this destination (outside, or inside if open) than place player in that Silo. Otherwise, if destination is open then will be placed inside, else outside: do silo allocation for appropriate silos only (inside or outside):

- allocate to the most fun non-full silo :-), i.e.
- find all non-full silos...
    - if there are none then create/find a new/empty silo and allocate player
    - if one then allocate player to it
    - if more than one then select using fun test, i.e. (we propose) weighted sum of present friend's meetings and messages [more complicated things could go here - why don't Blast Theory write some]

Generates arrival game event(s). If player is on a mission then check for a custom arrival message/event/thing and use if present. Complete mission if appropriate.

Check for local event and process if in scope.

If flagged to allocate mission/dilemma on arrival then... choose at random one that is in time/destination scope and has not been done by player already and allocate.

Find when in a non-hub destination

"Find PLAYERNAME"... if:

- named player does not exist - fail, unknown player
- named player has left town or died - find failure event
- named player is resting - generate find failed (resting) event
- named player does exist...
    - in same destination, different silo
        - finding player outside, named player inside and destination closed - fails as if the named player was not in the same destination
        - otherwise - finding player is moved to the named player's silo, even if it is nominally full, generating a find event and arrival and observations of arrival as if the finding player had just arrived at the destination and been allocated to that silo
    - in same silo - player is notified as if they had found the player in a different silo, their silo is not changed, bystanders do not observe the event
    - not in the same destination - find failure event

Find when in the hub

Does not work.

Saying (Say)

Text of message is broadcast to all other active players in the same silo. Act of saying is recorded. Player friendship information updated accordingly.

Things (Pick, Drop, Use)

If incapacitated, a player cannot pick up, drop or use.

If player becomes incapacitated while carrying a non-clothing thing then the thing is dropped.

If player is carrying/wearing a thing and picks up another then the first is dropped.

If a player asks to use a particular thing that is in their destination but they are not carrying then they first pick it up.(??)

Pick up must always specify thing.

If player is on a mission then check for custom effects.

Drop does not specify thing (only one held). Any additional text would have been ignored (i.e. "drop tissues" is just treated as "drop", even if player is not holding "tissues").

Use with no thing implies use the thing currently carried; use with a specific thing implies (a) if that thing is held then use that (b) else try to pick up named thing and then use it (see above). They will be left holding the thing used, or nothing if it was destroyed.

If a thing is used....

- if the player is on a mission then check for custom effects, otherwise...
- if the thing affects someone else... (type 'other') and
    - no-one else active is present in current silo, then effect on user is 'effect of use on using player using if no-one else present or doesn't affect someone else'
    - at least one other active person is present in current silo, one other active player is chosen at random as the subject, and
        - effect of use on using player when someone else present applies to user
        - effect of use of subject applies to subject
        - effect of use on active bystander applies to any other players present in same silo
        - (optional??) effect of use on non-silo but same destination active bystanders applied if present??
- if the thing affects user
    - effect of use on using player using if no-one else present or doesn't affect someone else applies to user
    - effect of use on active bystander applies to any other players present in same silo
    - (optional??) effect of use on non-silo but same destination active bystanders applied if present??
- if destroyed on use, then destroyed action performed (remove from game or respawn)

Update (Update)

Like other actions, update ends a rest. (if an update causes a player to become active do they get both the wake up and the update messages?)

No effect on game state. Request recorded, to trigger update message to player, including (some of):

- destination
- player health description
- carrying
- same silo players
- same destination (inside/outside) things
- current dilemma
- current mission

Leave (Leave town)

Player never receives any message after this other than confirmation of having left. NB may require enforcing in message gateway to avoid race conditions, e.g. if leave message causes player to become active.

Player is removed from current silo, and state becomes 'left town'. Other players in silo receive bystander of left town event. If player is player-in-need of any mission(s) then check for mission actions.

Rest (Rest)

See Resting (above). Optional parameter is number of hours to rest; default is 2 hours.

Help (Help)

Logs player help event. Reported to operator, who can respond on an individual basis. No other automatic effect on game state.

Dilemma response (A, B, C)

If player:

? has no current dilemma, 'no dilemma' event/response

? has current dilemma

    o dilemma does not have corresponding option, apply default response

    o dilemma has corresponding option, apply the effect of that response to the player (no visible effect to bystanders)

Other events

Things (used on)

See Player use (effect of use of subject applies to subject).

Becoming active (end of rest)

See Resting (above).

Description changes

Caused by 'effects', i.e.

? player's use of thing

? thing used on player

? local event

? dilemma

? mission

Health changes

Most changes to health are results of effects; other possibilities are:

? hospital special case

? operator intervention

Health of resting players cannot be affected by effects or hospital.

There is a potential mortality threshold of (?) 10:

? if the player's health is greater than this before an effect then their final health cannot be less than this.

There is an incapacity threshold of (?) 20 and below inclusive:

- players cannot move, find, pick, drop or use when incapacitated; they can only carry clothing.

There is an upper limit of 100. A health increase cannot make a player less half than as close to 100 as they were before. E.g. health 50, cannot increase by more than 25; health 80 cannot increase by more than 10 ((100-80)/2); health 90 cannot increase by more than 5 ((100-90)/2).

If a player's health becomes zero (or less - capped to zero), they die:

- their state becomes dead (not playing)
- player removed from silo & destination (ends up in "off the board")
- if player is player-in-need on any mission(s) then check for and process custom mission actions/effects
- player death event (Operator notified to create a single custom death message); bystander observe death event(s)
- if carrying (clothes), the thing's destroyed action is performed (remove or respawn)
- any outstanding timers are cancelled (hub, dilemma)

If a player is player-in-need on any mission(s) and their health increases then check for custom actions on those missions (player recovery). NB be careful of race condition with player on mission actually having helped them by using the right thing on them.

Dilemmas - allocation of, time-out or other default trigger

See Dilemmas (above)

Local events - occurrence

See Local events (above).

Missions -

See missions (above)

7.1.7 Game Lifecycle

Game time

Game time is essentially independent from physical world time. While the game is active game time progressive at a define rate with respect to physical world time, typically 10 hours physical world = 1 hour game time.

Game time must not be moved backwards.

Game activity

While the game is not active no game actions or events are processed and no game timers make progress.

Relation to Incoming Message Handling

Some message handling can occur when the game is not active:

- initial processing of leave town
- response to first incoming message from a player after game closes with game closed message
- messages when the game is closed are logged but not passed to the game engine for processing

? Note: if the game engine crashes (i.e. message delivery to the game engine fails) then the gateway attempts to reliably deliver these messages at the first opportunity while attempting to attract the attention of an operator)

The incoming gateway should normally be opened after the game is made active and closed before the game is made inactive.

Relation to Outgoing Message Gateway

The outgoing gateway should normally be enabled before the game is made active and disabled after the game is made inactive (to allow events in progress at game close to the processed and corresponding messages generated and despatched).

Opening/closing game

The game will be opened initial when the event starts. It will close finally when the event ends. It will be closed temporarily:

? each night

? at other times if major maintenance work is urgently required, e.g. whole system crash/stall

Opening the game

At the beginning of the event will any players be pre-registered? Maybe; hopefully it won't matter (see next point).

But in general players may be registered while the game is not active. In this case the player's state on registration remains 'new' until the game is activated. A player who is added when the game is active may be immediately made active/resting as appropriate.

The game will never be activated until the start of the event.

When the game is opened:

? resting/active states are updated for current physical world time

? a message is sent to each active player. (game opens event)

? 'new' players are made active/resting as appropriate (get welcome to dof message rather than game opened message)

? event processing resumes

? the game clock is started

Closing the game

When the game is closed:

? the game clock is stopped

? event processing stops - flush (process) events that are now released/in flight

? a message is sent to each active player (game closed event)

7.2 Game Events Specification

7.2.1 Introduction

Each thing that happens within the core game is represented by one (or more) game events, which may be used:

? by the game engine itself, e.g. to avoid duplicate local event allocation

? by the message generation system

- ? by the orchestration interface, to get a detailed view of player activity
- ? by the player web interface, to construct a history of player activity in the game

Major groups of game events are:

- ? things that players do
- ? things that players attempt which are impossible/etc.
- ? other things that happen to players
- ? bystander observation of an game event
- ? other system -related events??

7.2.2 Things to do with players

are (* = observable):

- ? issues a command (go, say, ...)
- ? things to do with movement
  - o leaves a destination/silo*
  - o arrives at a destination/silo*
    - ✍ normal/special? hub/destination? first/return??
  - o find another player
  - o is moved outside/inside a destination (by an effect)
  - o attempts to go to a non-existent destination
  - o attempts to go to the same destination
  - o attempts to find another player but cannot (not here/dead/left or resting)
  - o attempts to find another player who does not exist
  - o attempts to perform an action when incapacitated
- ? things to do with things
  - o picks up a thing*
  - o uses a thing*
  - o has a thing used on them (see uses a thing)
  - o drops a thing*
  - o attempts to pick up an unknown thing
  - o attempts to pick up a thing not present
  - o attempts to use when not holding/nothing there
  - o attempts to drop something when not holding anything
  - o attempts to perform an action when incapacitated
- ? things to do with saying
  - o says something*
- ? things to do with info/update/status
  - o requests update
- ? things to do with local events
  - o is given a local event
- ? things to do with dilemmas
  - o is given a dilemma

  - responds to a dilemma
  - defaults (times out) on a dilemma
  - defaults (does something else) on a dilemma
  - attempts to respond to a dilemma when no current dilemma
- things to do with missions
  - is given a mission*
  - satisfies a mission 'stage'*
  - completes a mission*
  - defaults (times out) on a mission*
  - is assigned to a mission as player-in-need
- health-related player things
  - (see experiences effect)
  - becomes/ceases to be incapacitated*
  - dies*
  - (attempts to perform an action when incapacitated)
- other player-state things
  - experiences effect
  - leaves town*
  - starts resting*
  - becomes active/stops resting
  - starts the game
  - game becomes active while playing (e.g. start of day) (from other system events??)
  - game becomes inactive while playing (e.g. end of day) (from other system events??)
- observations
  - of any of the * above
- operator-related ?? (outside core game?!)
  - player receives custom message ??

7.2.3 Other system-related events

are:

- destination description changes
- destination opens/closes
- local event begins
- local event ends
- game opens
- game closes
- game starts
- game ends
- thing destroyed
- thing spawned
- silo created

7.2.4 Authoring-related events

are:

- ? unauthorised

7.2.5 Parameters

Event	**Parameters**	**Related events**
<u>things to do with players</u>	player, current destination, current silo, game time, system time	
issues a command (go, say, ...)	text, reference to parsed command request	
<u>things to do with movement</u>		
leaves a destination/silo*	destination, silo, cause event	
arrives at a destination/silo*	destination, silo, cause event, cause type: arrive hub/arrive destination/find/effect	
find another player	other player	leave, arrive
is moved outside/inside a destination (by an effect)	inside/outside, cause event	
attempts to go to a non-existant destination	destination name	
attempts to go to the same destination		
attempts to find another player but cannot (not here/dead/left or resting)	other player, reason (not here, dead, left, resting)	
attempts to find another player who does not exist	other player name	
attempts to perform an action when incapacitated	command event	
<u>things to do with things</u>		
picks up a thing*	thing	
uses a thing*	thing, other player (optional)	
has a thing used on them (see uses a thing)	thing, other player, uses thing event	
drops a thing*	thing, inside/outside	
attempts to pick up an unknown thing	thing name, pick-up-and-use flag	
attempts to pick up a thing not present	thing	
attempts to use when not holding/nothing there	thing (optional) [not generated for failed pick-up-and-use]	

attempts to drop something when not holding anything	thing (optional)	
attempts to perform an action when incapacitated	command event	
<u>things to do with saying</u>		
says something*	text, other active players count (i.e. number of observation of this)	
<u>things to do with info/update/status</u>		
requests update		
<u>things to do with local events</u>		
is given a local event	local event, present at start/on arrival	
<u>things to do with dilemmas</u>		
is given a dilemma	dilemma, type: hub arrive/non-hub arrive/mission, from mission (if mission type)	
responds to a dilemma	dilemma, response	
defaults (times out) on a dilemma	dilemma, default outcome	
defaults (does something else) on a dilemma	dilemma, default outcome	
attempts to respond to a dilemma when no current dilemma	response code	
<u>things to do with missions</u>		
is given a mission*	mission, type: hub arrive/non-hub arrive, player-in-need (if required)	
satisfies a mission 'stage'*	mission, stage	
completes a mission*	mission	
defaults (times out) on a mission*	mission	
is assigned to a mission as player-in-need	mission, other player	
<u>health-related player things</u>	(see also experiences effect)	
becomes/ceases to be incapacitated*	incapacitated flag, changes health event	
dies*	changes health event	
(attempts to perform an	command event	

action when incapacitated)		
<u>other player-state things</u>		
experiences effect	effect, affects health flag, final health, crosses boundary flag, affects description flag, effect type: local event, dilemma allocation, dilemma response, mission allocation, mission stage player/player-in-need/bystander/distant bystander, thing use player/player-in-need/bystander/distant bystander, cause event	incapacitated, dies
leaves town*		
starts resting*	type: end of preferred play/rest message, rest time (if rest message)	
becomes active/stops resting*	type: start of preferred play/end of rest message	
starts the game		
game becomes active while playing (e.g. start of day) (from other system events??)		
game becomes inactive while playing (e.g. end of day) (from other system events??)		
<u>observations</u>		
of any of the * above	observed event, different silo flag (copy of observed event fields for efficiency: event type, player, other player, thing, text)	
<u>other system-related events</u>	game time, system time (no player, destination, silo)	
destination opens/closes	destination, open/closed	
local event begins	local event, destination	
local event ends	local event, destination	
game opens	reason text, message players flag, message text	
game closes	type: routine/exceptional, reason text, message players flag, message text	
game starts		
game ends		
thing destroyed	thing, type: hub silo recycling/used, destination, silo, cause event	
thing spawned	reason: start of game/thing destroyed, thing, destination, silo, cause event (thing destroyed)	
silo created	destination, silo, reason: hub no experience/hub dilemma/hub mission/hub chat/destination	

7.3 DOTF-Berlin Message Specifications, linked to Game Events

7.3.1 Introduction

All Messages sent to players are built from Message Specifications. For Barcelona message specifications see DOTF-Berlin_Message_Specification.doc.

Messages are keyed to Game Events.

Game Event	**Role**	**Condition**	*Message specification: Short Name*	*Description*	*Indicative Text (from Barcelona with minor edits)*	**Comments**
things to do with players						
issues a command (go, say, ...)		normally :-)	-			
	player	verb not recognised	NO_SUCH_COMMAND	command not recognised	This key word is not recognised. The main key words are SAY, SAY TO, GO, PICK UP, USE, DROP, INFO and FIND. Go to www.dayofthefigurines.co.uk to learn more.	
things to do with movement						
leaves a destination/silo*	observer	not hub (hub leaves shouldn't be observed anyway)	W_LEAVE_FROM_DESTINATION	player witnesses partner leaving	$PARTNERNICKNAME has just left.	
arrives at a destination/silo*	observer	non-hub destination	W_ARRIVE_AT_DESTINATION	player witnesses partner arriving	$PARTNERFULLNAME has just arrived.	
	player	arrival at non-hub destination during destination description, not find or effect	ARRIVE_DESTINATION	player arriving at a (non-hub) destination	you've arrived at $DESTSHORTNAME, $DESTLONGDESC. $LIST_PLAYERS $LIST_THINGS	Inadequate: e.g. need to know about mission player-in-need and/or key things
	player	arrive in hub, only player in silo, not hub arrival dilemma or mission (i.e. no experience, first person for chat, hub departure mission or	ARRIVE_HUB_SOLO	player arrives in the hub for a single player (no action) experience, or in the case that someone arrives in the	you're on your way to $DESTLONGDESC. $HUBDESCRIPTION	

		dilemma)		hub and is waiting for a multiplayer meeting for some time		
	observer	in hub, have received ARRIVE_HUB_SOLO	HUB_MEETING	player's meet in hub	you meet $PARTNERNICKNAME, $PARTNERDESCRIPTION, $PARTNERHESHE seems to be $PARTNERHEALTH.	
	player	arrive in hub, someone else present (i.e. chat)	ARRIVE_HUB_MEETING	player is joined in the hub by another player, in this case they have not been given the description	on the way you meet $PARTNERNICKNAME, $PARTNERDESCRIPTION. $HUBDESCRIPTION	
find another player	player		FOUND_PLAYER	target of find found in player's location	after a quick search around $DESTSHORTNAME, you find $PARTNERNICKNAME and join them.	
is moved outside/inside a destination (by an effect)			-			message generated by effect event
attempts to go to a non-existent destination	player		GO_NOSUCHDEST	the player didn't specify a destination that we know	This destination is not recognised. Try again using a name of a destination listed on your map or at www.dayofthefigurines.co.uk	
attempts to go to the same destination	player		GO_SAMEDEST	player already at destination	you're already at "$DESTLONGNAME", feeling $PLAYERHEALTH.	
attempts to find another player but cannot (not here/dead /left or	player	other player is not resting in the same destination	FIND_PLAYER_NOT_PRESENT	target of find not currently in the same location as play	$PARTNERNICKNAME can't be found anywhere in $DESTSHORTNAME	

resting)						
attempts to find another player who does not exist	player		FIND_NO_SUCH_PLAYER	target of find not a player the system recognises	the player you are looking for can't be found. Check that the name you used is right and try again.	
attempts to perform an action when incapacitated	player	attempts go/find	PLAYER_CANT_MOVE	message sent player when they try and move but are incapacitated	you are $PLAYERHEALTH, and lie helpless	
<u>things to do with things</u>						
picks up a thing*	player	not clothes	PICKUP_TPOS	Default pickup success message sent to actioning player	you pick up $THINGLONGNAME	
	observer	not clothes	PICKUP_TBOS	Default pickup success message sent to bystander	$ACTIONINGPLAYER picks up $THINGLONGNAME	
	player	clothes	PICKUP_TPOSC	Default pickup clothes success message sent to actioning player	you put on $THINGSHORTNAME	
	observer	clothes	PICKUP_TBOSC	Default pickup clothes success message sent to bystander	$ACTIONINGPLAYER puts on $THINGLONGNAME	
uses a thing*			-			messages from thing use effects
has a thing used on them (see uses a thing)			-			messages from thing use effects
drops a thing*	player	not clothes	DROP_TPOS	Default drop success message sent to actioning player	you put down the $THINGSHORTNAME.	
	observer	not clothes	DROP_TBOS	Default drop success message sent	$ACTIONINGPLAYER puts down $THINGLONGNAM	

				to bystander	E.	
	player	clothes	DROP_TPOSC	Default drop clothes success message sent to actioning player	you take off the $THINGSHORTNAME and place it on the ground.	
	observer	clothes	DROP_TBOSC	Default drop clothes success message sent to bystander	$ACTIONINGPLAYER takes off $THINGLONGNAME and places it on the ground.	
attempts to pick up an unknown thing	player	at least one thing in destination inside/outside	PICKUP_THINGNOTTHERE	ERROR Player tried to pickup an absent 'Thing'	there is no "$MESSAGETEXT" nearby to pickup. You're at $DESTSHORTNAME. $LIST_THINGS	
	player	no thing at all in destination inside/outside	PICKUP_NOTHINGTHERE	ERROR Player tried to pick something up when no things were present	there is nothing near you to pick up. You're at $DESTSHORTNAME. $LIST_THINGS	
attempts to pick up a thing not present	player	at least one thing in destination inside/outside	PICKUP_THINGNOTTHERE	ERROR Player tried to pickup an absent 'Thing'	there is no "$MESSAGETEXT" nearby to pickup. You're at $DESTSHORTNAME. $LIST_THINGS	
	player	no thing at all in destination inside/outside	PICKUP_NOTHINGTHERE	ERROR Player tried to pick something up when no things were present	there is nothing near you to pick up. You're at $DESTSHORTNAME. $LIST_THINGS	
attempts to use when not holding/nothing there	player	provided not due to failed pick-up-and-use	USE_NOTHINGHELD	ERROR Player is trying to use something when they have nothing to use	you don't have anything to use. $LIST_THINGS	
attempts to drop something when not holding anything	player		DROP_NOTHINGHELD	ERROR Player is trying to drop something when they have nothing to drop	you don't have anything you can drop. $LIST_THINGS	

attempts to perform an action when incapacitated	player	attempts pick up	PICKUP_TPOFH	Default pickup failure message due to poor health sent to actioning player	you do not have the strength to pick up $THINGSHORTNAME	
	player	attempts to drop (clothes)	DROP_TPOFC	incap clothes drop failure message due to poor health sent to actioning player	you do not have the strength to move let alone undress.	
<u>things to do with saying</u>						
says something *	observer		PLAYER_HEARD_SAY	player hears a say message	$PARTNERNICKNAME said: "$MESSAGETEXT" .	
<u>things to do with info/update/status</u>						
requests update	player		UPDATE_REQUEST	The message sent to a player who has requested an update	Message built from available/applicable information according to the following priority, incorporating player knowledge: $DESTINATION_INFO $PLAYERHEALTH $LIST_THING_HELD $MISSION_INFO $LIST_PLAYERS $LIST_THINGS	
<u>things to do with local events</u>						
is given a local event			-			message from local event effect
<u>things to do with dilemmas</u>						
is given a dilemma	player		-			message from dilemma allocation effect

responds to a dilemma			-			message from dilemma response effect
defaults (times out) on a dilemma			-			message from dilemma default response effect
defaults (does something else) on a dilemma			-			message from dilemma default response effect
attempts to respond to a dilemma when no current dilemma	player		use UPDATE_REQUEST			As per update request
<u>things to do with missions</u>						
is given a mission*	player & observer		-			message from mission allocation effects
satisfies a mission 'stage'*	player, player in need & observer		-			message from mission stage effects
completes a mission*			-			(message generated by stage event)
defaults (times out) on a mission*			-			(message generated by default stage event)
is assigned to a mission as player-in-need	player		-			message from player in need allocation effect
<u>health-related player things</u>						
becomes/ ceases to	observer		INCAP_TB	message sent to bystander	$PARTNERNICKNAME is now	

be incapacitated*				on nearby player incapacitation	$PARTNERHEALTH.	
	player		INCAP_TP	message sent to player when they become incapacitated by someone or something else	you are lying helpless on the ground. You are $PLAYERHEALTH	
dies*	player		PLAYER_DEAD	message sent to player when they die	you have been killed to death. Thank you for playing Day Of The Figurines.	
	observer		W_PLAYER_DEAD	message sent to partner when they die	$PARTNERNICKNAME has died. Within moments two men in green overalls come along, load $PARTNERNICKNAME's body into their battered white van and drive off	
(attempts to perform an action when incapacitated)						
<u>other player-state things</u>						
experiences effect	player	health boundary change, not becoming incapacitated or dying	HEALTHCH_TP	message sent to player on health boundary change	you are now $PLAYERHEALTH.	
leaves town*	player		LEAVE_TOWN	player leaves the town	$PLAYERNICKNAME catches a lift on a passing truck and leaves town. Thank you for playing Day Of The Figurines. Please reply to this msg with any feedback.	
	observer		W_LEAVE_TOWN	partner leaves town	$PARTNERNICKNAME catches a lift on a passing truck and leaves town. You won't be seeing them again.	

starts resting*	player	player request rest	-			no message
	player	player begin rest and end day	PLAYER_REST_END_DAY		BT write me.	
	observer		W_REST		$PARTNERNICKNAME is resting.	Not a great message
becomes active/stops resting*	player	player wake from requested rest	PLAYER_WAKE		You wake up and slowly realise where you are in .	More description?! Include something like an up
	player	player wake and begin day	PLAYER_WAKE_START_DAY		BT write me.	
	observer		W_PLAYER_WAKE		$PARTNERNICKNAME snorts loudly and opens their eyes with a starts.	
starts the game	player		WELCOME	welcome message to player to start the game	welcome to Day Of The Figurines. $PLAYERNICKNAME has been dropped by a truck at the edge of town. You are $PLAYERHEALTH. Where do you want to go?	
<u>observations</u>						
of any of the * above	player		-			see 'observer' cases for observed event
<u>other system-related events</u>						
destination opens/closes			-			author a local event if a message is required
local event begins			-			see player is given a local event event
local event ends			-			
game opens	every active	if message players flag	-			Text for mangling from event.

	player					Requires specific operator construction!
gam e closes	every active player	if message players flag	-			Text for mangling for event. Requires specific operator construction!
game starts			-			
thing destroyed			-			
thing spawned			-			
silo created			-			
game ends day 1	every playing player		END_GAME_DAY_1	Day game end (sent at end of day)	the hour is over for $PLAYERNICKNAME. Some things a stuff.	
game ends day 2	every playing player		END_GAME_DAY_2	Day game end (sent at end of day)	the hour is over for $PLAYERNICKNAME. Some things a stuff.	
game ends day N	every playing player		END_GAME_DAY_N	Day game end (sent at end of day)	the hour is over for $PLAYERNICKNAME. Some things a stuff.	
game ends day 24	every playing player		END_GAME_DAY_24	Day game end (sent at end of day)	the hour is over for $PLAYERNICKNAME. Some things a stuff.	
starts game day 2	every playing player		START_GAME_DAY_2	Day game end (sent at end of day)	the hour is begun for $PLAYERNICKNAME. Some things a stuff.	
game starts day 3	every playing player		START_GAME_DAY_3	Day game end (sent at end of day)	the hour is begun for $PLAYERNICKNAME. Some things a stuff.	
game starts day N	every playing player		START_GAME_DAY_N	Day game end (sent at end of day)	the hour is begun for $PLAYERNICKNAME. Some things a stuff.	

game starts day 24	every playing player		START_GAME_ DAY_24	Day game end (sent at end of day)	the hour is begun for $PLAYERNICKNA ME. Some things a stuff.	

7.3.2 Notes

Multiple shortening length message specifications have been replaced by a "smart" aggregation system. Where potential elements are prioritised by the authors and subsequently included where appropriate (according to mission and history) and the space available.

7.4 Authored Text Reference

7.4.1 Introduction

This document attempts to pull together and index all possible sources of text that appear in messages sent by the DOTF system, which in some sense are therefore authored. This includes:

- properties of database classes included via message elements
- message specifications, from
  - database MessageSpecifications
  - effects, in thing (use), local events, dilemmas and missions
- other built-in text
  - health descriptions (in source code)
  - custom -coded message element text fragments (in source code)

7.4.2 Authored class properties

Player

- nickName - used in $PLAYERNICKNAME, $PARTNERNICKNAME, etc.
- descriptionText - used in $PARTNERDESCRIPTION, etc., which is intended to be used in context "You are X"/"He/she is X".
- (answers to player's questions - not used)

Destination

- shortName - used in $DESTSHORTNAME, etc. Useful e.g. in "You are at X", "Going to X"
- longName - used in $DESTLONGNAME, etc.

Non-hub destinations will have a (single) current destination description (current inside/outside/closed descriptions held by a single destination description). From this the description is determined and might be cached in the Destination. See Destination_Description.

Silo

Hub Silos have a single description, determined from a randomly selected Hub Destination_Description when the Silo was entered. This may be cached as descriptionText, or looked up using des criptionID.

Destination_Description

All Destination (and Hub Silo) descriptions are specified by Destination_Descriptions, which additionally specify the time over which they apply and whether the (non-hub) destination is open. Text is:

- descriptionInsideText (if it has a distinct inside) - e.g. used in $DESTLONGDESC if inside
- descriptionOutsideText - e.g. used in $DESTLONGDESC if outside

- shortDescriptionInsideText (optional, and if it has a distinct inside) - e.g. used in $DESTSHORTDESC if inside
- shortDescriptionOutsideText - e.g. used in $DESTSHORTDESC if outside
- A description is a complete sentence EXCLUDING final full stop.

Thing

revised singular/plural proposal

- shortNameSingular - if thing is single
- shortNamePlural - for all things
- longNameSingular (was description) - if thing is single
- longNamePlural - for all things
- cardinality type: single/some/many

Note: names should NOT include "a", "the", "some", etc. Used in $LIST_THINGS, $LIST_THING_HELD

Effect

Effects are used in: thing use, local events, dilemma, mission. They include description change option, in particular:

- description change type: no change, replace, append
- text

Replace implies text is used in place of player's own description, which is intended to be used in context "You are X", "He/she is X", "Bob, who is X, ...".

Mission

In addition to mission allocation and stage effects:

- descriptionText, which is a self-contained sentance INCLUDING full-stop, e.g. "The policeman is still waiting for you." (says Keir), used (maybe) in update and/or web site to remind player of their current mission (if any)

7.4.3 Message Specifications

A message specification (DOTF-Berlin_Message_Specification_3.doc) is a text template of a complete message for processing by the manglator. Message specifications come from:

- The text property of the MessageSpecification instances in the database used when generating messages from game events
- The messageText property (optional) of an Effect from a Local event, dilemma, mission or thing use

7.4.4 Other built-in text

Health descriptions

Currently:

- "cock of the walk"
- "banging"
- "hot to trot"
- "very fine indeed"
- "averagely fine"

- "feeling peaky"
- "out of sorts"
- "poorly"
- "pretty ill"
- "very unwell"
- "maximum sick"
- "about to pass out"
- "unable to move"
- "fading fast"
- "very fucked"
- "dead"

For use in phrases "who is XX", "you are XX", "he/she is XX".

Text to be stored in MessageElements, where the title is built from form "HEALTH_" + bottom of health range percent + "_TO_" + top of health range percent, eg. HEALTH_20_TO_30

Special case message elements

See DOTF-Berlin_Message_Elements_2.doc. These include text compiled into the message generation code. Currently (those which do not simply select one of the above pieces of text):

- $LIST_THINGS, List of things in current_destination (inside/outside). e.g. "There is|are " ... " here."
- $LIST_PLAYERS, List of other (active) players in current_silo, as is most relevant to player and fits available space. eg "", "[Player.nickName], [Player.description],", " is|are here"
- $LIST_THING_HELD, thing held/worn, e.g. "" or "You are wearing|carrying [Thing.longName]." of if there is room "You are wearing|carrying [Thing.description]."
- $PARTNERHESHE, "he" or "she" according to gender
- $PARTNERHISHER, "his" or "her" according to gender

7.5 Message Elements

7.5.1 Introduction

Each Message is constructed from a Message Specification (see Message Specification); each message text may combine text with variable elements, i.e. Message Elements.

Messages are (currently) constructed in a game context comprising:

- String data - text to be "mangled"
- dof2.db.Player player - player receiving the message (?)
- dof2.db.Player partner - another significant player, e.g. player being observed, player met
- dof2.db.Player thing_actioner - player using the thing (use of thing, only)
- dof2.db.Player thing_recipient - player that thing is used on (if applicable to thing, use of thing only)
- dof2.db.Destination currrent_destination - current destination of player
- +dof2.db.Destionation current_silo - current silo of player, (e.g. used in hub, inside/outside text)
- dof2.db.Destination future_destination - final destination of player (if in the hub)
- dof2.db.Thing thing - thing (if pick, drop or use)

? String passText - uninterpreted text to include, e.g. body of say or name actually given for (presumed) destination or thing

Barcelona Message Elements:

Element text,	**description,**	**class introspection/special case,**	**details**	
$PLAYERNICKNAME	a player's nick name	class	(Player) player.nickName	
$DESTSHORTNAME	a player's current destination, short name	class	(Destination) current_destination.shortName	
$DESTLONGNAME	a player's current destination, long name	class	(Destination) current_destination.longName	
$DESTSHORTDESC	short description of a player's current destination	special case	(Destination) current_destination.shortDesriptionInsideText or current_destination.shortDescriptionOutsideText according to player inside/outside (If undefined then long description) [could look up from Destination_Description using current_destination.insideDescriptionID/outsideDescriptionID]	
$DESTLONGDESC	long description of a player's current destination	special case	(Destination) current_destination.desriptionInsideText or current_destination.descriptionOutsideText according to player inside/outside (If undefined then long description) [could look up from Destination_Description using current_destination.insideDescriptionID/outsideDescriptionID]	
$HUBDESCRIPTION	description of a hub, will be short, fits in a hub meeting message	class	(Silo) current_silo.descriptionText [or look up from current_silo.descriptionID]	
$THINGDESCRIPTION	description of a thing	class	(Thing) thing.description	
$THINGLONGNAME	long name of a thing	class	(Thing) thing.longName	
$PARTNERDESCRIPTION	description of a player that is being met	class	(Person) partner.descriptionText	
$PARTNERNICKNAME	short name of a player that is being	class	(Person) partner.nickName	

	met			
$ACTIONINGPLAYER	The nickname of the actioning player in a use event	class	(Player) thing_actioner.nickName	
$RECIPIENTPLAYER	The nickname of the recipient player in a use event	class	(Player) thing_recipient.nickName	
$MESSAGETEXT	message text being passed in	special case	= passText parameter	
$LIST_THINGS	player mission and history appropriate listing of the things in destination (and their descriptions)	special case	List of things in current_destination (inside/outside). If no things, then "$NO_THINGS_PRESENT". Else if recently recieved description "There is|are "..." here."; things are grouped by kind, "a [Thing.longName]", "a couple of ...", "a few ...", "many"; comma separated with "and" for last clause. "There is|are "..." here."; things are grouped by kind, "a [Thing.description]", "a couple of ...", "a few ...", "many"; comma separated with "and" for last clause.	Length constrained to maximise relevant information in full message
$NO_THINGS_PRESENT	no things present in destination	special case	"" | "There is nothing here. "	new. Currenlty "" is always used, this allows for custom message, if desired
$LIST_PLAYERS	show mission and history appropriate information about co-located players	special case	List of other (active) players in current_silo. If no other players then $NO_PLAYERS_PRESENT. Else ..." is|are here"; comma-separated and "and" for last clause; "[Player.nickName], [Player.description], "; unless a player is incapacitated, in which case "who is $PLAYERHEALTH"(player) instead of Player.description. In the case that only one player is listed, cf above, except text is ..." is nearest." (not " is here.").	Preferentially select: (1) mission player-in-need (2) incapacitated friends (3) other incapacitated players (4) a (best?) friend (5) random
$NO_PLAYERS_PRESENT	no players present in silo	special case	"" | "There is no-one here. "	new. Currenlty "" is always used, this allows for custom message, if desired

$LIST_THINGS_HELD	thing held/worn (possibly with description, in the context of player message history)	special case	If holding nothing then "". Else if recently recieved description "You are wearing|carrying [Thing.longName]." Else "You are wearing|carrying [Thing.description]." according to Thing type (clothes/not) as message length permits	
$PARTNERHEALTH	partner health	special case	Selection of health description according to partner.health (see below)	
$PLAYERHEALTH	player's health	special case	Selection of health description according to player.health (see below)	
$PARTNERHESHE	PARTNER HE or SHE	special case	"he" or "she" according to partner gender	
$PARTNERHISHER	partner his or hers	special case	"his" | "her" according to partner gender	
$ANSWER1	the answer to question 1	class	(Question) question1.questionText	new
$ANSWER2	the answer to question 2	class	(Question) question2.questionText	new
$ANSWER3	the answer to question 3	class	(Question) question3.questionText	new
$ANSWER4	the answer to question 4	class	(Question) question4.questionText	new

No longer used:

- $PLAYERFULLNAME
- $PARTNERFULLNAME
- $THINGSHORTNAME
- $PLAYERHESHE
- $PLAYERHISHERS

7.5.2 Notes

message length variants

The variable length versions are likely to disappear as message elements included in message specifications; a single "smart" message element may be included to hint at including that "kind" of information e.g.

- things - the following have been replaced with "smart" LIST_THINGS:
  - LISTTHINGSWITHDESCRIPTION
  - LISTTHINGS
  - LISTONETHINGWITHDESCRIPTION
  - LISTONETHING
- players present - the following have been replaced with "smart" LIST_PLAYERS:

  - LISTPLAYERSWITHDESCRIPTION
  - LISTPLAYERS
  - LISTONEPLAYERWITHDESCRIPTION
  - LISTONEPLAYER
- things held - the following have been replaced with "smart": LIST_THING_HELD:
  - THINGHELD
  - THINGHELDWITHDESC

Things

Talking about things

- A (single game) thing has:
  - "shortName" -> "systemName"
  - "longName" -> "shortName" (no definite or indfinite article, i.e. "a", "some", "...")
    - singular (if thing is single)
    - plural
  - "desription" -> "longName"
    - singular (if thing is single)
    - plural
  - cardinality of (single game) thing: singular / "some" / "many"

So...

- A (single game) thing
  - indefinite article
    - cardinality singular: "[is] a $THINGSHORTNAMESINGULAR" or "[is] a $THINGLONGNAMESINGULAR"
    - some: "[are] some $THINGSHORTNAMEPLURAL" or "[are] some $THINGLONGNAMEPLURAL"
    - many: "[are] many $THINGSHORTNAMEPLURAL" or "[are] many $THINGLONGNAMEPLURAL"
  - definite article
    - cardinality singular: "the $THINGSHORTNAMESINGULAR" or "the $THINGLONGNAMESINGULAR"
    - some: "the $THINGSHORTNAMEPLURAL" or "the $THINGLONGNAMEPLURAL"
    - many: "the $THINGSHORTNAMEPLURAL" or "the $THINGLONGNAMEPLURAL"
- N (a small number, "some" or "many") of a game thing (class):
  - indefinite article
    - cardinality singular: "[are] N|some|many $THINGSHORTNAMEPLURAL" or "[are] N|some|many $THINGLONGNAMEPLURAL"
    - some: "[are] some|many $THINGSHORTNAMEPLURAL" or "[are] some|many $THINGLONGNAMEPLURAL"
    - many: "[are] many $THINGSHORTNAMEPLURAL" or "[are] many $THINGLONGNAMEPLURAL"
  - definite article - never used!!!

Health descriptions

Currently:

- "cock of the walk"
- "banging"
- "hot to trot"
- "very fine indeed"
- "averagely fine"
- "feeling peaky"
- "out of sorts"
- "poorly"
- "pretty ill"
- "very unwell"
- "maximum sick"
- "about to pass out"
- "unable to move"
- "fading fast"
- "very fucked"
- "dead"

For use in phrases "who is XX", "you are XX", "he/she is XX".

Text to be stored in MessageElements, where the title is built from form "HEALTH_" + bottom of health range percent + "_TO_" + top of health range percent, e.g. HEALTH_20_TO_30.

7.6 DOTF-Berlin Message Aggregation and Pacing

7.6.1 Sending a Message

The system will send a message to a player;

(*'d entries are those that will include aggregated content)

- if the player is active...
  - in response to a player message...
    - that is correctly formed and successful (say, go, update, pick up, use, drop, find, leave town, help) after a generic time out (around 5 to 7 minute) *
      - e.g."go locarno", "update", "say How did you guys get here?"
    - that is correctly formed but unsuccessful, or is incorrectly formed, after an 'error' time out (around 3 to 5 minute) *
      - e.g. "jkdfn" which causes NO_SUCH_COMMAND or a "go to hell" which causes a GO_NOSUCHDEST
  - at the beginning of their game
  - at the end of their game
  - at the beginning of the day
  - at the beginning of the day (delayed due to resting)
  - at the end of the day

  - at the end of the day (early due to resting)
  - after "waking" from requested "resting"
  - if a player in the same silo as the player has sent a SAY message *
  - if they become incapacitated *
  - if they die
  - if they leave town
  - when they arrive at a destination (including the HUB)
  - when they meet another player in the HUB
  - when some one dies in the same silo as the player *
  - as a part result of an 'effect' (e.g. during a dilemma, mission, local event, ??)
  - if some one uses a thing (assuming there is a bystander, destination-wide bystander, or recipent effect)*

7.6.2 Building (Aggregation) a Message

Elements that can be used in an aggregated message (or message part) are...

- Due to Game Events
  - when the player witness (same silo)
    - another player leaving
    - another player arriving
    - another player drops a thing
    - another player picks up a thing
- Destination specific information
  - what Thing(s) are present
- Silo specific information
  - which player(s) are present (and active)
- Player specific information
  - that the player has gone through a health boundary
  - the Thing the player is holding (if applicable) (low priority)
  - the mission the player is on (if applicable) (low priority)

7.6.3 Aggregation priority in response to player action

For 'update' (and for waking from rest)

- where the player is
- health
- mission they are on
- incapacitated player(s) present
- what they are holding
- player(s) present description
- thing(s) present description
- player(s) present
- thing(s) present

For 'say' (and heard 'say')

? player(s) present description
? thing(s) present description
? player(s) present
? thing(s) present
? health/health boundary change
? mission they are on
? what they are holding
? health
? where the player is

For 'use', 'pick up', 'drop' (and having a thing used on them?)

? response the action
? health/health boundary change
? player(s) present description
? thing(s) present description
? player(s) present
? thing(s) present
? health
? where the player is

For 'find'

? response the action
? player(s) present description
? thing(s) present des cription
? player(s) present health
? thing(s) present
? health/health boundary change
? where the player is
? health
? what they are holding

For 'go'

? response to the action
? health/health boundary change
? health
? what they are holding

When becoming incapacitated

? that you've dropped your thing (if applicable)
? player(s) present description
? player(s) present