

Overview of Multiplayer Mobile Game Design

Version 1.1; December 17, 2003

Mobile Games

NOKIA

Copyright © 2003 Nokia Corporation. All rights reserved.

Nokia and Nokia Connecting People are registered trademarks of Nokia Corporation. Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. Other product and company names mentioned herein may be trademarks or trade names of their respective owners.

Disclaimer

The information in this document is provided “as is,” with no warranties whatsoever, including any warranty of merchantability, fitness for any particular purpose, or any warranty otherwise arising out of any proposal, specification, or sample. Furthermore, information provided in this document is preliminary, and may be changed substantially prior to final release. This document is provided for informational purposes only.

Nokia Corporation disclaims all liability, including liability for infringement of any proprietary rights, relating to implementation of information presented in this document. Nokia Corporation does not warrant or represent that such use will not infringe such rights.

Nokia Corporation retains the right to make changes to this specification at any time, without notice.

License

A license is hereby granted to download and print a copy of this specification for personal use only. No other license to any other intellectual property rights is granted herein.

Contents

1	Introduction	6
2	How Multiplayer Games Differ from Single-Player Games.....	7
2.1	Players Provide the Struggle	7
2.2	Games Must Be Repeatable	7
2.3	Handling Drops Gracefully	7
2.4	Player Matching Issues	8
2.5	Short Play Sessions	9
2.6	Game Balance.....	10
3	Approaches to Game Networking.....	11
3.1	Server-Driven Games.....	11
3.2	Peer-to-Peer Games.....	11
3.3	Cellular Network Games Must Be Server-Driven	12
3.4	Bluetooth Games Are Locally Server-Driven.....	12
4	Dealing with Latency.....	13
4.1	Levels of Latency	13
4.2	Design Approaches	13
4.2.1	Round-robin games	13
4.2.2	Simultaneous-movement games.....	14
4.2.3	“Act whenever” games	14
4.2.4	Slow update games	14
4.2.5	Shared solitaire games	14
4.3	Latency and Network Protocols.....	15
4.3.1	Air network games with MIDP	15
4.3.2	Air network games with Symbian OS C++	15
4.3.3	Bluetooth games	15
5	Designing for Community	16
5.1	Chat	16
5.2	Challenges	16
5.3	Diplomacy.....	17
5.4	Buddy Lists	17
5.5	Clan Messaging	17
5.6	Web Site.....	17
6	The Metagame	18
6.1	Tournaments or Seasons	18

6.2	Trading	18
6.3	Offline Activities.....	18
6.4	Stable Strategies	18
7	Conclusion	19
8	Terms and Abbreviations	20
9	References.....	21

Change History

December 17, 2003	V1.0	Initial document release
January 12, 2004	V1.1	Updated for MIDP 2.0

1 Introduction

This document is intended for two audiences: game designers seeking a better understanding of the particular problems involved in multiplayer mobile game design, and mobile developers seeking a better understanding of game design in general and multiplayer mobile game design in particular.

The document assumes that readers are already familiar with *Overview of Single-Player Mobile Game Design*, which is available on the Forum Nokia Web site (see Chapter 9, “References”). Much of the ground covered in that document will not be repeated here, and while it is about single-player rather than multiplayer mobile games, much of what it discusses applies to multiplayer mobile games as well.

Since the inception of digital games, single-player titles have predominated, largely because of the inherently single-user nature of most electronic devices. While many game consoles allow two-player games with the attachment of a second controller and some PC games have always allowed “hot seat” play, in which two players take turns with one machine, not until the advent of commercial online services and the Internet did multiplayer games become widespread.

Mobile games, on the other hand, are likely to follow a different developmental path from that of games for other devices. Unlike consoles and PCs, mobile phones are networked by nature, providing an environment that is fitting for multiplayer games.

Yet a number of technical and business issues have prevented multiplayer mobile games from becoming widespread. These issues are gradually being addressed, and successful multiplayer mobile games are beginning to appear. It is quite likely that, as such games spread, they will become an increasingly important part of the mobile game market, particularly because mobile phones are, by nature, social devices, used to communicate and engage with others — and games that allow players to communicate and engage with each other fit naturally with the device’s nature and use.

2 How Multiplayer Games Differ from Single-Player Games

2.1 Players Provide the Struggle

Any game is, in a sense, a struggle; players expect to work to achieve their objectives, and if they win too easily, they will not find the game interesting. In a single-player game, the player's struggle is with the game system itself; the player struggles with physical challenges (such as jumping at the right moment in a platformer), mental challenges (such as solving a puzzle), and computer-controlled artificial intelligence (AI) opponents.

In a multiplayer game, the player's struggle is usually with the opposing players, rather than with obstacles posed by the game system. In a way, this is easier for game designers; because the players create their own opposition, your design does not have to supply it.

Consequently, multiplayer games should give players ways to hinder and help each other. In addition, often the sorts of challenges posed by single-player games still exist. For example, in a first-person shooter (FPS), actions such as aiming shots and ducking pose physical challenges; in a strategy game, the need to think through moves poses mental challenges. And AIs can still be helpful, for a number of reasons: to take over a player's position if that player drops out of a game (something that can happen very easily in a mobile game); to let people learn the game by playing against AIs before trying it against live opponents; and to control less important characters or smaller positions in the game.

2.2 Games Must Be Repeatable

Players often talk about "beating a game," by which they mean playing through and succeeding at all of its levels. But that applies only to a single-player game. In a multiplayer game, you don't beat the game; you beat the opposing players.

Many single-player games are designed as, in effect, one-time experiences. While it's certainly possible to play through them again, just as it is possible to re-read a book or re-watch a movie, much of their appeal lies in players' finding out what happens in successive levels — an experience that itself is not repeatable.

The one-time experience is not a useful basis for multiplayer games, because these games are typically played for short sessions and generally are not "savable." In a multiplayer game, players expect to become intimately familiar with the play setting; variety is instead provided by unpredictable player actions.

With multiplayer games, designers must strive for games that are infinitely replayable.

2.3 Handling Drops Gracefully

In a multiplayer game, it often happens that one player needs to leave the game. This is especially true for a mobile game, because a player may need to take an incoming phone call on the device or drive into a tunnel and lose the wireless network connection. The loss of one player should not disrupt the experience of the other players: The game must handle such departures gracefully.

It is not always easy for a game to detect that a player has dropped out, because most mobile data technologies do not keep open a permanent connection between devices or between a device and a server.

It helps to program the game to tell the server (or the Bluetooth master device), “I am leaving,” when a player pauses or closes the game application — but this handles only purposeful drops, not accidental ones.

Other techniques include:

Having each device poll the server periodically, simply to say, “I am still here,” with the server assuming that a device that has not polled within a certain period has dropped or has lost its connection.

Having the server assume that a device that performs no communication for some reasonable period (say, 20 seconds) has dropped. This technique would make perfect sense for, say, a fast action game over Bluetooth, but less sense in a game of chess played over the air network, since in that style of game a player might well ponder his or her move for a long time before sending data.

Having a system that allows players to vote on kicking one player out of the game. Thus, when a player is not responding, the other players can eject him or her. They can also eject people they find annoying. This kind of system needs to be handled carefully, however, because in some kinds of games, players might abuse it to get rid of the player who is closest to winning.

Once a drop is detected, of course, the game must determine how to handle the drop and how to keep going so that the remaining players can still have fun. A number of techniques are feasible:

“Civil Disorder.” The dropped player’s position remains in the game, but does nothing. The phrase was originally used in Allan Calhammer’s board game Diplomacy, in which a dropped player’s armies and fleets remain in place on the board, and defend against attacks, but are motionless.

AI takeover. An AI opponent takes over the player’s position. This approach is commonly used in online multiplayer card games.

Replacement player. With some types of games, it may be feasible to fill the dropped player’s position with a new player. For example, in online multiplayer card games, a new player can, if he or she chooses, take over a position being played by an AI. Designers should avoid having the game annoy new players by dropping them into hopeless positions.

Designers also must avoid unwittingly giving players incentives to drop. Suppose the game has a leader-board system, in which a player’s ranking on the leader board improves with each win and declines with each loss. And suppose that the leader-board position is not affected by games that a player drops. In this case, competitive players will just turn their phones off when they are losing, which will annoy the other players.

In such a case, it would be better to have the leader-board position affected by wins and losses in all games that the player starts. In other words, if the player drops and his or her position is taken over by an AI that completes the game, the player’s leader-board rank is changed as a result, even though part of the game was played by an AI. This gives players a strong incentive to complete games, since most of them will play better than an AI. It may annoy some players who drop inadvertently, but that is preferable to a system that can be easily abused.

2.4 Player Matching Issues

Consider this scenario: A player excitedly installs a new multiplayer game, starts it up, makes a wireless connection to the server, joins a game — and discovers that he is matched with highly experienced and skilled players who destroy him in a matter of minutes.

In all likelihood, the player never returns.

This is not just a hypothetical scenario; it happens all the time. Many multiplayer games are unfriendly to “newbies.” This is also, obviously, far from an ideal situation.

When feasible, it is a good idea to engineer player-ranking and player-matching systems that try to match players of equivalent skill against one another. Such systems need to be designed with care, however, because players will inevitably try to “game” such systems. Pitfalls to avoid include:

- Providing incentives for players to “drop” (as previously discussed).

- Providing incentives for experienced players to enter newbie games and obliterate inexperienced opponents. (One way to avoid this problem is to ensure that a victory over much lower-ranked players has a much smaller effect on a player’s own leader-board ranking than a victory against peers or superior players.)

- Providing incentives for experienced players to establish new identities so they can “start over” and advance quickly by obliterating newbies.

- Allowing “perfect” scores. For example, if the ranking system is based purely on a win/loss ratio, a player who wins one game and never returns has a perfect score and appears at the top of the leader board — which is patently absurd.

In general, there is no perfect player ranking and matching system — like voting systems, each has advantages and disadvantages — but just as it is worthwhile to think through and test game systems, it is important to think through and test ranking and matching systems as well.

2.5 Short Play Sessions

One of the most highly regarded PC games of all time is Civilization, a game in which the player controls a civilization, guiding it from the Stone Age to the Space Age, in competition with AI-controlled civilizations. A priori, one might think this game would make a great multiplayer game, because one need only replace the AIs with live players. That is false. On two occasions, multiplayer versions of Civilization have been published, and both of those games failed, commercially and critically.

The problem is that Civilization is a game that takes 12 or more hours to play. The lengthy playing time is not a problem for a single-player game; the player simply saves when he or she needs to do something else, and returns to the game later. Suppose, however, that you have six players and, after an hour or two of play, you save the game; what’s the likelihood that you will be able to assemble the same six players at a set time in the future to continue the game? Contrariwise, how likely are you to get six players to sit down and play continuously for 12 hours?

In general, mobile games (both single- and multiplayer) should be designed for relatively short sessions, since a game can be interrupted at any time by an incoming call. With single-player mobile games, it is always possible simply to save the game, but with multiplayer games, it is not, since other players may want to continue after one player departs. As a result, multiplayer mobile games should be designed either to be played to completion in a relatively brief period or to be playable in short sessions over time. (The games described in Section 4.2.3, “‘Act whenever’ games,” are an example of the latter approach.)

Experience with multiplayer games online shows that the most popular games are generally those playable in 15 minutes or less — half an hour at most. Massively multiplayer games (MMGs) are the exception to this rule — but even though they go on forever, most people play them in sessions of a few hours at a time.

2.6 Game Balance

A single-player game is said to be “balanced” if it feels neither too hard nor too easy to play. In a multiplayer game, “balance” has a different meaning: The game is balanced if players feel that they all have an equal chance of winning.

Of course, this does not mean that all players must start from identical or equivalent positions. It is possible to “balance” an asymmetric game in at least two ways. One is by having asymmetric win conditions; if player A is powerful and player B is weak, but player A needs many victory points to win and player B only needs a few, the game might still be balanced.

The other way is by permitting diplomacy — that is, providing ways for the players to help and hinder each other in the game, along with a means for them to form alliances. Asymmetric diplomatic games tend to be self-balancing, in the sense that weaker powers combine against stronger ones.

3 Approaches to Game Networking

In the conventional online games business, two approaches typically are taken with multiplayer networked games. Some games are server-driven, while others are peer-to-peer.

3.1 Server-Driven Games

In a server-driven game, each player's machine communicates only with a remote server. The server tracks all players' actions and sends each machine the information it needs to display the game state to its player.

This method has some advantages. First, only the server needs to know the whole game state; thus, it can ensure that the players cannot "hack" their machines to gain access to information they should not have (such as what is on the other side of walls). Second, each player's machine is sending and receiving only the data it needs, so that even when a player is on a dial-up connection, his bandwidth will not become overloaded. Third, with a speedy server (or networked cluster of servers) and a broadband pipe, it is possible to run MMGs this way, with thousands or tens of thousands playing in the game simultaneously.

This method has one big disadvantage: Someone has to pay for the server and the bandwidth it uses. In the case of MMGs such as EverQuest, the costs are very substantial — and such games typically charge players monthly subscription fees to defray that cost.

The method can also be used for smaller-scale multiplayer games (that is, ones with a handful of players, rather than an MMG). As an example, Quake works this way. Yet the developers of Quake do not provide the servers; they simply allow any game owner to set up a server anywhere on the Internet. Players use third-party tools like Gamespy to locate these servers and play on them. Since each server handles a mere handful of players, the costs of running one are modest, and the fan community suffices to support the game.

3.2 Peer-to-Peer Games

In a peer-to-peer game, no single player's machine is considered authoritative; instead, all players' machines talk directly to all other players' machines. Each machine keeps track of the overall game state, with checksums passed to ensure that they remain in sync.

This has one big advantage: The players themselves supply the hardware and bandwidth to play the game, lifting this cost from the shoulders of the developer or publisher.

It does, however, have drawbacks: For one thing, if the game has any hidden information (areas of the map a player is not entitled to see, for example), it is fairly easy to "hack the client" to reveal this hidden information. Second, in a peer-to-peer game, the amount of data that needs to be exchanged increases exponentially with the number of players. As an example, if the number of players increases from 4 to 5, each player must now be sending information out to 4 others, rather than 3 others. So, we have increased network traffic from 12 (4 players each sending to 3 others) to 20 (5 players each sending to 4 others). On a dial-up connection, a player may quickly start to run up against bandwidth limits. Peer-to-peer games used to be limited to 8 or fewer players, although clever data compression schemes have allowed some to support up to 32 simultaneous players.

3.3 Cellular Network Games Must Be Server-Driven

When a multiplayer mobile game is played over the cellular air network (as opposed to Bluetooth), peer-to-peer networking is not an option. All data traffic is routed through the tower and over the operator's equipment; individual handsets do not have IP addresses; and direct, device-to-device communication is simply not supported, at least as of this writing.

Consequently, developers who want to provide multiplayer games over the cellular air network today must understand that the game must be server-driven; and they (or partners) will need to operate servers to provide game play to their customers.

This means that developing a multiplayer mobile game is actually two development projects in one: You must develop the client-side application that runs on the handset and communicates with the server; and you must develop the server-side application that interprets communications from the handset, adjudicates the results, and returns new game states to the players.

From a business perspective, this has two implications: First, multiplayer mobile game development requires a skill set not required for single-player mobile game development (server-side coding, which may be accomplished in Java™ 2 Platform, Enterprise Edition [J2EE™]), and is therefore likely to be more costly.

Second, operating a multiplayer mobile game means that the developer (or mobile game publisher) will have ongoing, recurring support costs, for hardware, hosting, and bandwidth.

In an ideal world, a recurring revenue source would offset these costs. Yet most mobile games (single-player and multiplayer) are sold on the basis of a one-time application fee; and while the network operators derive air-minute or data-transfer revenue for data transferred during game play, it is very rare (as of this writing) for operators to share any portion of that revenue with game developers.

This is a big business issue for multiplayer mobile game developers, and one that they can only hope will be addressed by operators in the future.

3.4 Bluetooth Games Are Locally Server-Driven

Bluetooth is a technology that allows devices to establish a local-area wireless network. Devices must be within about 10 meters of each other to establish connections. Series 60 devices all support Bluetooth, as may some other Nokia devices in the future.

Bluetooth has one big advantage, from a game-playing perspective, over other wireless technologies: It is fast. Over the cellular network, you must anticipate multisecond latency. Over a Bluetooth network, latency is typically on the order of 20 to 50 milliseconds — a lower level of latency than over the wired Internet. This permits multiplayer, fast action game playing locally.

However, local Bluetooth networks (called piconets) conform to a “hub and spoke” design. That is, one device, called the “master,” connects to up to seven other devices, called “slaves.” All data communication is routed through the master; slaves have no ability to exchange data directly with each other.

Thus, a Bluetooth game will also be a client/server application — but locally served, meaning that one player's device acts as server (master) for the others. This does mean, however, that the developer does not have to support a server remotely over the Internet; players bear their own device costs (and bandwidth is free).

For more about Bluetooth game development, see *Games over Bluetooth* (see Chapter 9, “References”).

4 Dealing with Latency

Latency is the amount of time it takes a system to respond. With a stand-alone PC, latency is normally measured in microseconds — that is, the amount of time between a user’s key press and the computer’s response is tiny. Latency over the conventional Internet is usually on the order of 100 to 200 milliseconds, although it can vary widely. Latency over the cellular air network is usually on the order of multiple seconds.

4.1 Levels of Latency

Even Internet levels of latency pose big problems for developers of fast action games. Players on different machines need to be kept in sync, and it would be unacceptable for the action to freeze for a hundred milliseconds at a time. Consequently, most games use predictive algorithms, assuming that running characters keep running in the same direction, for instance, until they receive data that says otherwise. Many games mask the effects of latency by building delays into the system. For example, in Quake, it generally takes a few hundred milliseconds for a bullet to fly from a gun to its target, so that by the time the game needs to display the appropriate animation for the target, data has already been exchanged between the firer’s and target’s machines.

Even with techniques such as these, some very fast game styles are not feasible over the Internet; you will not find multiplayer online streetfighting games, for example. Hand-to-hand combat is so swift that Internet levels of latency make this very difficult to accomplish.

Moving to cellular air networks, with multisecond latency, the situation is even worse. For many game styles, delays of several seconds are simply intolerable. FPSs, multiplayer platformers, and, indeed, almost any real-time game style, will not work.

The problem is ultimately one of network configuration. For voice communication, and for almost all data services other than games, air network latencies are tolerable. Consequently, the operators have no strong business incentive to reduce latencies from current levels. While the move to third-generation (3G) technologies will greatly expand the amount of bandwidth available over the air network, it will not necessarily alter the latency equation. (That is, you may be able to send 100 KB in the time it used to take to send 10 KB, but it will still take a second or more for the data to start arriving.)

It is conceivable that, as the mobile games business grows and operators come to understand the problem, they will work to reduce the level of latency, but there is scant evidence that anything of the kind is happening at the moment. Multiplayer mobile game designers must understand that there is no technological fix on the horizon — and that, consequently, dealing with air network latency is a problem that can be addressed only through design.

4.2 Design Approaches

Several approaches have been successfully used to address latency issues. Several are described in this section. This is not intended as an exhaustive list; with some imagination, designers may well hit on other solutions.

4.2.1 Round-robin games

A round-robin game is one in which each player takes a turn, “around the table,” though of course in a networked game there is no literal table. Since players generally take multiple seconds to act, multisecond

latency is tolerable. One drawback is that if there are too many players, people become bored waiting for their turn; it is advisable to limit round-robin games to four or fewer players.

Examples: Chess; Hearts.

4.2.2 Simultaneous-movement games

A simultaneous-movement game is one in which each player decides what to do during the next turn, and composes his or her orders. Then, all orders are revealed simultaneously, and the turn is adjudicated. In a board game, this might be accomplished by having each player write moves down (this is the approach taken in Diplomacy); in a digital game, it is accomplished by having each player use the software to decide what to do, then send the orders over the network to a server. When the server receives all players' orders (or after a set period of time), it resolves the turn and sends the new game state to each player.

One of the advantages of this system is that players don't have to wait so much; a player who sends in his orders early might be required to wait a little while, but that's not as bad as having to wait through five other players' turns.

Examples: Diplomacy; Laser Squad Nemesis.

4.2.3 "Act whenever" games

A game of this type is ongoing and persistent, and a player may sign in at any time, play for a time, and sign off. In some games, an offline player's position cannot be affected by players in the game at that moment; in others, this is not true.

Examples: MMGs such as EverQuest.

4.2.4 Slow update games

A game of this type is ongoing and persistent, with each player having a real and persistent position in the game. The player's position is given orders by the player, which it slowly and continuously executes until the player signs in to the game and changes his or her orders. Thus, players are motivated to monitor the game in short, frequent sessions, rather than to play it continuously.

Example: The old academic game Empire.

4.2.5 Shared solitaire games

In a shared solitaire game, each player is presented with the same single-player game. Each plays for some period of time, after which the game ends (for all players). The player who, starting from the same position as all others, scored the highest is considered the winner.

This has the advantage of giving players the feeling that they are competing with and participating in the same game as the others — but the only network communication required for the game is at the beginning (when players are matched and the initial positions are sent to all players) and at the end (when scores are reported and the winner declared). In between, play is purely on the handset, where fast action can occur; and a few seconds of latency, before the game starts and after completion before scores are reported, is quite tolerable.

Examples: PoppaZoppa; Qube.

4.3 Latency and Network Protocols

4.3.1 Air network games with MIDP

The Mobile Information Device Profile (MIDP) 1.0 standard requires MIDP-compliant devices to support only one network standard: HTTP. The MIDP 2.0 standard adds support of secure HTTP (HTTPS). Although both protocols allow handset manufacturers to support other protocols optionally, and some handsets do support User Datagram Protocol (UDP) or TCP socket connections, developers cannot rely on the existence of either protocol in a MIDP device. UDP and TCP are supported in Nokia devices that support MIDP 2.0, but not MIDP 1.0 devices.

HTTP is a protocol that requires multiple “handshakes” between a client device and a server before data is exchanged. Since each step in the handshake is subject to air network latency, you must plan for latency as high as five seconds when using HTTP.

UDP and TCP connections do not require the same level of handshaking and, when you can use these protocols, latency is likely to be in the one-to-three-second range.

For more on these issues, see the document *Multiplayer MIDP Game Programming* (see Chapter 9, “References”).

4.3.2 Air network games with Symbian OS C++

Developers coding in C++ for Series 60 devices are in luck; those handsets support UDP and TCP connections in addition to HTTP.

4.3.3 Bluetooth games

Bluetooth uses its own packet protocol. It also provides much lower levels of latency (typically between 20 and 50 milliseconds) than the cellular air network. Thus, fast action games are quite feasible for Bluetooth-enabled devices.

However, all players must be within 10 meters of the “master” device (the player who is acting as server for the others); and there is an upper limit of eight players. Indeed, Nokia recommends that Bluetooth games be limited to four players, for bandwidth contention reasons.

See *Games over Bluetooth* (Chapter 9, “References”) for more on Bluetooth and game development.

5 Designing for Community

It is true that one strong appeal of multiplayer games is that determined human opponents are much more challenging than AIs. But that is only part of the appeal of multiplayer games.

It is also true that if you are developing for highly constrained devices, such as Series 40 phones with their 64 KB limit on MIDlet size, developing any kind of multiplayer game at all is a challenge, and there may not be much opportunity to provide community support features.

However, the best multiplayer games encourage communication among the players, and help to build strong connections among the players. As online game pioneer Gordon Walton says, “They come for the game, but they stay for the community.”

5.1 Chat

With conventional online games, chat among players is one of the most appealing features. This is true both for hard-core games such as MMGs, where chat is necessary for trade and to find others to group with, as well as for lighter games such as card games played online, where “table talk” and the ability to socialize during play are important.

Chat is, of course, much more difficult to implement on mobile devices, because a telephone keypad makes text entry harder than a keyboard and because even though these devices are designed for voice communication, it is difficult or impossible to carry voice and data over a single cellular network connection.

In addition, text chat is difficult to provide for games that require rapid player responses, because players will want to be playing the game rather than entering text.

However, for some game styles you may find it useful to implement chat or messaging. This is particularly true for games that are played out over days or weeks, rather than in single play sessions, and in which players can assist or thwart each other. For games of this type, players will want to communicate in order to form or break alliances. Communication can be implemented either via Short Message Service (SMS) — or via the server, with messages sent via HTTP (or another protocol) from the handset to the server, then stored on the server and sent to the recipient the next time that player signs in to the game. The advantage of the latter scheme is that it is less costly for the players, and also allows you to preserve players’ anonymity, since only the server needs to know the identity of each player’s handset.

For faster-moving games, you may still find it worthwhile to implement a “taunt” system, whereby a player can dispatch one of a few short bits of text to the others with a few key presses, rather than entering a complete message manually.

5.2 Challenges

For two-player games in particular, but perhaps for larger-scale games as well, it may be worthwhile to implement a “challenge” system, whereby one player can challenge others to a game. This is useful from a “viral marketing” standpoint, as friends will spread the word about a new game they enjoy to one another in the form of challenges.

One way to do this is by having the player send the recipient’s phone number to the server, and having the server compose an SMS message to that number, including a link that allows the player to download the game. Ideally, players are registered by phone number, so that if this phone number belongs to a player

who is already registered, the message instead allows the player to sign in to the game and start playing the challenger.

5.3 Diplomacy

Some multiplayer games are like Monopoly: The players do not directly interact during play. However, it is generally more satisfying to play games in which players can hinder and assist each other. If the game is a free-for-all, like Quake in death-match mode, there is little need for communication; but if players can form and break alliances, they will need to communicate to do so, and enlisting the help of others becomes one of the focuses of the game.

Given the difficulty of chat via mobile devices, it may be worthwhile to build a system to allow players to automatically offer, accept, and break alliances, and/or to signal their intentions by in-game actions rather than with words. Bridge bidding systems are an example of the latter.

5.4 Buddy Lists

Many online games allow players to create buddy lists, so that they know when friends are in the game and available to play. Given the limited screen size of mobile devices, doing this is more difficult, but may be worthwhile for certain game types.

5.5 Clan Messaging

In games that involve large numbers of players, players often group into “clans,” and appreciate tools to facilitate communication among clan members. Most MMGs, for instance, have separate chat channels for clans. It is possible to provide similar facilities in a mobile game, either with custom development or by using a third-party group messaging tool.

5.6 Web Site

Particularly for games that are persistent, it may be worthwhile to have a Web site where people can post to a bulletin board to discuss the game, swap hints, join clans, view leader boards, and so on. These kinds of things are difficult to do on a mobile device but are straightforward on the Web, and they rarely need to be accessed during game playing itself.

6 The Metagame

“Metagame” is a term originally coined by Richard Garfield, designer of Magic: The Gathering. A metagame is a “game around the game,” a set of surrounding things that increase interest in the game and promote repeat play.

Magic: The Gathering is itself a good example: The game involves two people playing a fantasy duel with their Magic cards, but players spend at least as much time buying cards, trading, and assembling their decks as they do playing. In fact, most of the strategy of the game is in deck-building, not in face-to-face play.

A sports season is another example: To fans, the shape of the whole season, and whether or not their team ends as champions, is more important than whether they win or lose a particular game.

With any multiplayer game, it is worth thinking about whether it lends itself to metagame, and if so, how you can make the metagame interesting. Some possibilities follow.

6.1 Tournaments or Seasons

Tournaments and “seasons” like those of sports give players an incentive to compete for the honor of winning the tournament or championship. This encourages them to think of the game as an ongoing entity instead of one session of play.

6.2 Trading

As the example of Magic shows, giving people a way to trade game items creates a metagame of its own.

6.3 Offline Activities

The classic example here is Magic deck construction. If players can do things that affect their performance in the game when outside of the game, they will do so, and will spend time and effort on the game and draw enjoyment from it — without hitting your servers.

6.4 Stable Strategies

Chess and Go are the best examples of games with stable strategies; starting positions are always the same. As a result, players can think about and debate strategy in an effort to improve their game — and that debate and study is a metagame in itself.

7 Conclusion

Multiplayer mobile game design is much like the design of any other game. The same basic questions apply: What does the player do? What pleasures does the game provide? What sorts of challenges does the player face?

Multiplayer games simplify some things — the existence of opponents lifts some of the burden of making the game a struggle from the designer — but more often complicate things. Multiplayer games generally must be infinitely replayable, deal with disappearing players gracefully, and be playable in relatively brief sessions. Designers of multiplayer games must also think about how their games can encourage communication and community among their players, as well as whether they create metagames to encourage repeat play.

For mobile multiplayer games, except for those implemented via Bluetooth, latency is a dominant concern, because the cellular air network has much higher latency than most other network environments. Since network latency is unlikely to decline in the foreseeable future, the issue is one that can be addressed only by clever design — which, paradoxically, offers game designers an opportunity to shine by resolving it in creative ways.

8 Terms and Abbreviations

Term or abbreviation	Meaning
AI	Artificial intelligence. In games, AI refers specifically to the logic governing the behavior of computer-controlled opponents.
Asymmetric game	A game in which players are not balanced in terms of power.
First-person shooter (FPS)	A type of game in which the screen displays a 3-D “first-person” view — that is, displays the game world “from the player character’s eyes,” instead of showing the character on-screen — and in which combat occurs through shooting. Doom is an example.
HTTP	Hypertext Transport Protocol. A protocol built on top of TCP that ensures accurate delivery of data, but with somewhat higher latency than protocols such as UDP; it is the protocol used to serve Web pages, and is required for all MIDP-compliant devices.
Metagame	A metagame is a “game around the game,” considerations that affect play and encourage repeat play. Examples of metagames are a sports season, and the trading and deck-building activities of Magic: The Gathering.
MIDP	Mobile Information Device Profile, the version of Java used in mobile phones.
Platformer	A particular type of game in which the player controls a single character at or near screen center, with the world scrolling behind the character right to left (or vice versa, when the character moves backward). Often, characters jump from one horizontal “platform” to another, hence the name of this game style.
Round-robin game	Any game in which play passes in order from one player to the next and only one player is “active” at a given time.
TCP	Transmission Control Protocol. It is built on top of Internet Protocol (IP), provides a detectable connection, and retransmits data if necessary to ensure receipt.
UDP	User Datagram Protocol. It is also built on top of IP, but is connectionless and does not ensure receipt of data — but because no data needs to be retransmitted, it provides somewhat lower latency than TCP. On the Internet, its primary use is in games, DNS requests, and streaming media.

9 References

All of these documents may be found at www.forum.nokia.com/documents.

Overview of Single-Player Mobile Game Design

Multiplayer MIDP Game Programming

Games over Bluetooth: Recommendations for Game Developers