FORUM NOKIA

# Designing Single-Player Mobile Games

Version 1.01; September 9, 2003

Mobile Games

NOKIA

### 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

## Change History

| | | |
|---|---|---|
| August 27, 2003 | V1.0 | Initial document release |
| September 9, 2003 | V1.01 | Minor technical correction |

# 1 Introduction

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 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.

While mobile devices are networked and can support multiplayer games, such games are more difficult and costly to develop than single-player titles. In most cases, a multiplayer game requires the game provider to maintain a server and assume bandwidth costs. Because the main business model for mobile games at present is one-time application fees — and no ongoing revenue to set against ongoing support costs — there’s little financial incentive for mobile game developers to move to multiplayer games.

Hence single-player mobile games dominate today’s market and are likely to account for a large portion of the market for the foreseeable future.

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

# 2 What Does the Player Do?

Game design is not about story, nor is it about character. This is not to say that story and character can't be part of an engaging game design — but they are not fundamentally necessary, either. Chess has no story; Go has no characters — and they are both superb games.

Game design is about *action.* Not necessarily fast action: Complicated strategy games can move quite slowly, for example. But fundamentally, when playing a game, a player *does things* to cause changes in the game state; the game interacts with those actions; and the player views the changed game state and decides what to do next.

When you begin conceiving of a game, the important question is not "What is the story?" or "What is the setting?" or "Who is the player's character?" — though these questions may become important later, and in some cases (such as with a licensed game), you may be handed the setting or character or story before you begin game design.

The important question is "What does the player do?"

The media assets you create in the course of development — the characters, obstacles, monsters, and other things the player will encounter — are the "nouns" of the game. But what the software enables are the "verbs" of the game — the set of allowable actions for players. And what the UI does is allow the player to trigger the verbs.

It is worth writing down all of the actions you expect the user to be able to perform in your game before you even begin writing the design specification. Each action will need to be coded, and each will need to be mapped to a UI feature. With mobile games, you want each important action to be attached to a single key during play whenever this is feasible. So, if your list is 30 verbs long, you have a problem. It is possible to nest less-important actions under menus — but not if a player will need to take these actions quickly during the game. (You can get around this problem by pausing the game when, say, the player brings up the inventory menu to change weapons.)

While some PC and console games have a huge number of potential actions — in NetHack, for example, virtually every key on a QWERTY keyboard has its own meaning — it is important to recognize that some very good games work with a relative handful of verbs. For example, Doom has only seven: turn left; turn right; move ahead; move back; shoot; switch weapons; and pick up (and the last is accomplished simply by moving through an object, requiring no UI feature).

When you have your list of verbs, think about them. Can you see how this list of actions could provide an enjoyable game? If not, would adding one or two more actions improve things? Or is the list too long, and, if so, what can you prune without materially damaging the game?

For each action, will a single key press be sufficient? In some cases, it won't. In a golf game, for instance, a player typically must specify the direction of a shot, and also the amount of power to be used. That requires at least two key presses — and most current mobile phones don't allow the detection of simultaneous key presses. In a case like this, you need to decide how to surmount the limitation to implement the range of actions you want. In the case of golf (or similar games in which direction and power are important, such as snooker or bowling), the usual solution is to have the player first use directional keys to determine the direction, then press and hold another key while a "power" bar gradually lengthens on the screen, reflecting how powerful the stroke will be if you release the key at this instant — with the player releasing the key at the desired moment to trigger the shot.

You want to map each action to one key, and each key to one action, whenever feasible; players will find it confusing if the 5 key is sometimes used to switch weapons and sometimes used to jump, or if both 5 and 7 mean "jump." If you must map multiple actions to a single key, ensure that the key's meaning is always clear to the player in context. If you find that your allowable actions are hard to map to the device's controls, think about how you can modify your game to work better with a mobile phone.

To look at it another way, game design has two main components: specifying the fundamental gameplay algorithms, and specifying the UI. The two — gameplay and UI — must dovetail neatly, and since the controls available on a mobile phone are limited, you need to think about how to implement a clean interface for your game from inception.

For a discussion of UI design and usability issues for mobile phones, see the document *Nokia Series 40 J2ME™ Game Usability Guidelines*, which can be found at http://www.forum.nokia.com/games. (You will find it useful even if Series 40 phones are not your target platform.)

# 3 Struggle and Challenge

Games are struggle. That is, a game that is too simple is trivial and boring; contrariwise, a game that is too difficult is frustrating. In both cases, the player is likely to set the game aside quickly.

People enjoy games that challenge them — that, at each step, offer a problem they have to deal with, a problem that is within their abilities to overcome, but one difficult enough that overcoming it is challenging and results in a sense of satisfaction.

In multiplayer games, the opponents provide challenges for one another, and winning means overcoming the other players. But in solo-play games, it is up to the game system to supply all the challenges. In these games, there are typically three types of challenges:

- Physical challenges. A game is a physical challenge if the player must manipulate the interface, at just the right moments, to achieve his or her desired objective. Racing games, side-scrollers, and first-person shooters are all examples of games that rely on physical challenge.
- Puzzles, or mental challenges. A player overcomes a puzzle by problem-solving. Graphic adventures, puzzles such as Bejeweled and Tetris, and games such as Chess all rely on mental challenges.
- AI (artificial intelligence) opponents. Solo-play games can provide challenge and opposition for players by having computer-controlled "opponents" who take the place of the sorts of live opponents you have in multiplayer games. Real-time strategy games, first-person shooters played in solitaire mode, and god games such as Civilization are examples of games that rely on AI.

It should be noted that few games rely solely on one type of challenge; for instance, a solo-play racing game may depend primarily on physical challenge, but generally, the player is racing against AI-controlled vehicles — two types of challenge. Similarly, many console games depend primarily on physical challenge but also provide puzzles that players must solve using their mental skills. A real-time strategy game may depend primarily on AI, but speed and mastery of the UI are important as well.

## 3.1 Physical Challenges

Any physical challenge relies ultimately on user mastery of the interface. Consequently, UI design becomes highly important for games that rely on physical challenge, and the particular characteristics of mobile phone handsets need to be considered carefully. As an example, most phones do have directional keypads, but the keypads are typically designed for menu navigation and are not always as responsive or easy to use as, say, a PlayStation 2 joystick. Consequently, if your game depends on allowing a user to specify direction (as almost all do), you generally should allow players to use either the directional keypad or the numerical keys to specify direction (2 = up, 4 = left, and so on).

## 3.2 Puzzles

Designing mental challenges is usually more difficult than designing physical ones. Typical approaches involve the following:

Resource trade-offs. Example: If I use a power-up now, it is not available for later use.

Tricky physical placement of game objects. Example: I must jump in a particular pattern to make my way across the lava by landing on rocks in the lava stream.

Interacting systems whose behavior is hard to solve. The classic “sliding square” puzzle is perhaps the simplest example.

Combination of game objects to produce effects. Example: the inventory puzzles of text adventures.

Even if your game is not puzzle-based, it is worth thinking about how you can construct puzzles using the features you build into the game for other reasons. For example, if you have a system that allows players to run and jump, it is not difficult to construct a puzzle that relies on these abilities to get to a “secret level,” which will add another bit of depth to your game.

### 3.3 AI

In some conventional games, AI can be very complicated. The developers of real-time strategy games, for instance, devote much of their programming effort to devising intelligent, sneaky AI opponents for people to play against. Given the limitations of current mobile devices, in terms of processing power and application memory space, mobile games are not going to offer AI opponents of that caliber.

However, such complicated AI is not always necessary. As an example, consider Space Invaders, where the only AI for the invading aliens is a simple rule that says, “Move left until you can’t anymore, then move down one row, then move right; if in bottom row, drop a bomb at irregular intervals.”

In many cases, a few simple rules of thumb suffice to give your AIs the appearance of something more than arbitrary, random behavior, and make them more challenging to the player.

### 3.4 Using “Verbs” to Define Challenges

Often, the actions you allow players to perform will suggest the sorts of challenges you can offer them. If players can run and jump, then some physical challenges — and puzzles built around physical challenges — are obvious. If players can push game objects around, then puzzles involving object manipulation to open up new game areas suggest themselves.

In general, the next step after defining your verbs is to define your challenges — the sorts of problems players will need to work to overcome.

# 4 Categories of Pleasure

People enjoy different games for different reasons. In one game, different players will enjoy different aspects — and even a single player will enjoy different aspects at different times. Consider what sorts of pleasures people can and will draw from your game; how to make these aspects more interesting; and how to make the game enjoyable in as many different ways, and to as many people, as possible.

To examine this in more depth, it is helpful to borrow Marc LeBlanc's taxonomy. (LeBlanc is an eminent game designer and was involved in the development of System Shock and Thief, among other titles.) According to LeBlanc, games offer eight categories of pleasure.

## 4.1 Sensation

The first category is "sensation" — sensory pleasure. Games can offer appealing graphics, sounds, and tactile pleasure — the feeling of "rightness" of a well-designed interface. Think about the graphical look of your game, what graphical tricks you can do with the animation to make it more interesting, and so on. Think about how well the look and feel of the game dovetails with its theme; if your game is about flesh-eating zombies, you probably don't want it to look "cute" (though, come to think of it, a game about cute flesh-eating zombies might be interesting). On the other hand, if your game is about an Italian plumber rescuing his girlfriend from a giant ape, cuteness may be just the thing.

## 4.2 Fantasy

By "fantasy," LeBlanc means something analogous to the concept of "suspension of disbelief" in fiction. You enjoy immersing yourself in the world of the game, accepting its underlying fiction as you play — even if the fantasy is as silly as "You are an Italian plumber trying to rescue his girlfriend from a giant ape."

Every aspect of your game — the verbs, the look, and the problems the player faces — should serve to reinforce the game's underlying fantasy. Providing a "backstory" in the help system is not sufficient — few people will read it anyway, and if your game *requires* a player to read it to understand what the game is about, you have failed. The game itself should impart the fantasy.

## 4.3 Narrative

Although "story" is not fundamental to gameplay, every session of play is an unfolding sequence of events, experienced over time by the player. Ideally, that sequence of events should be satisfying; the connection among the events should make sense; and the player should experience a sense of progress and (if he or she wins) satisfaction at the ultimate outcome. Consider the classic idea of narrative structure; ideally, a game, like a story, creates a sense of increasing tension, leading to a satisfying climax. If, in testing, you find that the endgame is dull, for example, you have a problem.

## 4.4 Challenge

As said previously, challenge is at the heart of any game. Ideally, a game keeps a player always at the edge of his or her capabilities — but still within them. Different players, of course, have different abilities, which is why many games allow the player to choose a difficulty setting, or adjust the difficulty automatically by monitoring how well the player is doing. If either approach is feasible for your basic game concept (and device constraints), it is well worth implementing.

### 4.5 Fellowship

Shared, intense experiences produce a sense of fellowship. This is particularly true in multiplayer games, but true of solo-play experiences as well; gamers often talk about the games they've played, and their experiences in the game, with other gamers — just as people talk about movies they've seen or books they've read.

Think about how you can foster a sense of fellowship. One approach is to provide puzzles; players can ask each other how to get past a particular obstacle. Cheats and Easter eggs (see glossary) also give them something to talk about. Allowing players to upload high scores to a shared leader board gives them something to brag about.

### 4.6 Discovery

When a player embarks on a new game, it is unknown; encountering new things in the game can create a sense of delight. If it's feasible, it is useful to have the player always encountering something new — some new challenge, some new monster, some new puzzle. Games that are limited in variety become stale quickly.

### 4.7 Expression

People enjoy ways of distinguishing themselves from others, of presenting their personalities in a particular light. In this sense, "expression" is most important to multiplayer games, in which players can express their characters in the avatars they choose, the equipment they wear, and the things they say. Even in a solo-play game, however, players appreciate ways of distinguishing their gameplay; even something as trivial as a choice of avatar, or the ability to input your character's name, can help. If the game allows a player to "win" in several different ways — by slaughtering all enemies or by sneaking around them, for instance — so much the better.

### 4.8 Masochism

This is an odd word choice on LeBlanc's part, but what he means is that there is a pleasure gained by submitting yourself to the structure of the game. Playing a game is rather odd, when you think about it; you agree to limit your behavior, to operate with a constrained set of rules that can sometimes be frustrating to struggle with, to achieve a wholly artificial objective that does not gain you love, make you money, or bring you fame. Yet precisely because it is a constrained, artificial experience, one that does not cause real pain if you fail, you find games enjoyable.

When someone plays your game, he or she is entering into your world and abiding by the rules you have established. It is your obligation to make that transaction worthwhile — to ensure that the rules' structure provides an interesting experience in itself, that this world is worth entering.

### 4.9 Thinking About These Categories

There are many ways of looking at games; LeBlanc's is only one, and his "categories of pleasure" are not intended to be exhaustive. Still, these categories can be a useful tool; when designing a game, it helps to think about each of them, which types of pleasure your game imparts, and how to strengthen each aspect of the design.

# 5 Designing Games in the Real World

So far, this document has discussed game design on a largely theoretical level. But every game is developed in the real world, and every game is subject to constraints. Constraints are sometimes viewed as annoying; no designer likes to be told that a feature is not possible because budget or the device's capabilities do not allow it. On the other hand, constraints can be viewed as a spur to creativity; the sonnet is a highly constraining form of verse, but some of the world's finest poems are sonnets.

## 5.1 Working with Technical Constraints

By comparison to consoles or PCs, mobile devices are quite constrained. They also vary a great deal in terms of screen size, keypad layout, and software implementation. Different manufacturers have different implementations of the Mobile Information Device Profile (MIDP), for instance — and even devices by the same manufacturer differ from one another. Nokia, at least, divides its devices into "series" with shared characteristics; thus, a game developed using the Developer Platform for Series 40 will work, with minimal changes, on all Series 40 Nokia devices. (For specifics about Nokia devices, see http://www.forum.nokia.com/devices.)

In general, it is good practice to develop a game for the most limited device the game must run on, then modify it to take advantage of features offered by more-capable devices. This is generally easier than trying to prune an application designed for a more fully featured device to make it work with a less-capable one.

Following are some of the technical constraints designers of mobile games should consider:

### 5.1.1 Application Size

There is always an upper limit on how large an application can be and still run. In addition, mobile games are generally delivered to devices through an operator's WAP gateway, and the operator generally has an upper limit on the file size it will allow through the gateway.

Nokia phones implementing JSR-185 (Java Technology for the Wireless Industry) — which includes Series 40 devices — permit Java Archive (JAR) files up to 64 kB in size and Java Application Descriptor (JAD) files up to 5 kB in size. This is also known as a *standard-sized application*. Since it is a standard, most operator gateways will have no problems with files of this size. Series 60 Nokia phones allow larger MIDlets — in principle, as large as the available space on the phone, which can be a megabyte or more, but in practice, the space is usually shared with other applications, and, in any event, operator constraints (and the speed of transmission of the Over the Air [OTA] network) make so large an application unfeasible. A file size of 100 kB to 200 kB is generally reasonable for Series 60 games, though developers should make sure that operators with which they partner will allow transmission of files of the chosen size, should it either fall under the aforementioned range or not.

Another concern is the amount of memory in the record management store (RMS) used by the MIDlet. The MIDlet suite should check the available RMS storage memory to ensure that there is enough for the suite's use. With JSR-185 compliant devices (including Series 40 devices), the maximum amount a suite can use is 30 kB of RMS. With more-capable devices, such as Series 60 devices, this will be less of an issue.

Rich games for the Nokia N-Gage™ mobile game deck are in another category; since they are delivered on memory cards, they can be multiple megabytes in size.

Program code, graphics, and sounds all take up space and contribute to the application's compiled size. A game's designer must ensure that the game can be implemented in the available memory space.

For a detailed discussion of this issue, see the document *Developing Java™ Games for Platform Portability*, which can be found at http://www.forum.nokia.com/games/ (click on the "Enter Game Programmers page" link at the bottom of the Game Programmers box).

### 5.1.2 Application Memory Space

When an application runs, it consumes more memory than the application file itself. Memory is used to store graphic buffers, objects created at run time, and so on. Programmers call the memory used "the heap." Every device has a limit on the amount of heap memory usable by applications. Consequently, the designer must also ensure that the game he or she specifies can run in the available heap space.

Series 40 Nokia phones have approximately 200 kB available to applications. Series 60 Nokia phones typically have more than 1 MB available, but this space is shared with other applications, and the Series 60 Platform supports multitasking, so other applications may be consuming part of the heap; however, heap space is rarely as much of an issue with Series 60 phones. The Nokia N-Gage game deck is a Series 60 device and behaves like other Series 60 devices in terms of heap memory.

Note that JSR-185 requires at least 128 kB heap memory, but Series 40 phones have a little more. If you are concerned about porting a game for Series 40 Nokia phones to other devices, however, you may wish to try to keep heap usage to 128 kB.

### 5.1.3 Screen Size and Format

Mobile devices have screens that are much smaller than those on PCs or consoles. The game obviously must be designed to fit in the device's screen. Because of this, some game styles are difficult or impossible to implement for mobile devices. For example, a real-time strategy game that requires the player to quickly jump from one area of the world to another, and monitor the overall game map via a radar view, would be hard to achieve on a mobile phone.

Typically, developers want their games to run on a variety of devices, and screen sizes vary from device to device. Most (but not all) Series 40 Nokia devices have screens that are 128 x 128 pixels, in 12-bit color (4,096 colors). As of this writing, all Series 60 Nokia phones have screens that are 176 x 208 pixels, also in 12-bit color. However, the Series 60 specification does not require any particular screen size, and future Nokia devices (as well as Series 60 devices from other manufacturers) may have screens of different sizes.

It is generally considered best practice to modify the graphics for different screen sizes, to ensure an optimal user experience. In some cases — if all game action occurs near the center of the screen, for example — it may be possible to make porting easier by simply centering the image, so that small screens lose the edge of the screen image, which is not critical to gameplay, and large screens have a blank area around the screen image.

Another concern, however, is that character sprites should not appear so large that they dominate the screen, nor so small that they are lost. As a rule of thumb, the width and height of a major character should be between 10 and 15 percent of the width and height of the screen.

A final concern is screen format; console and PC screens are wider than they are high (landscape format), while mobile phone screens are often either square or higher than they are wide (portrait format). This is a particular issue for side-scrolling games and 3-D action games, because a character at screen center is "closer" to the screen edge, and has less time to respond to obstacles or opponents that appear at the

edge of the screen, than in a console or PC game. Particularly when porting games to mobile devices from other platforms, designers need to be sensitive to this issue, and may need to modify the game's timing to give the player enough time to react.

### 5.1.4 Processing Power

Historically, games have always been among the most processor-intensive applications available, and game developers are always hungry for more speed. Typically, most cycles are spent "pushing pixels": performing the calculations necessary to modify the game view and update the display. And typically, each new generation of games provides better graphics, requiring more processing power — and often leaving owners of older machines behind.

Mobile device manufacturers rarely state the processing power of the chips their devices contain on their developer support Web sites. However, for small Java™ 2 Platform, Micro Edition (J2ME™) games, processing power is rarely a constraint. MIDlet size limits alone prevent such games from containing graphics that require too much processing power to display. Thus, when developing games for Series 40 Nokia devices, developers are unlikely to have problems with the processor speed. (There is, of course, no substitute for testing in the actual device.)

As of this writing, all Series 60 Nokia devices have 104-MHz ARM processors. The Series 60 specification does not require the use of any particular chipset, however, and devices from other manufacturers, as well as future Nokia devices, may have different processors. Given Moore's Law, power is likely to increase rather than decrease, of course. The 104-MHz speed is sufficient to support real-time rendered 3-D graphics, though not with the number of polygons or special graphics effects typical of high-end console and PC games.

Here, the smaller screen size works in developers' favor; fewer pixels on the screen mean fewer pixels to push with each buffer flip, so all things being equal, a lower-powered processor in a device with a small screen can perform as well as a higher-powered processor in a device with a large screen.

### 5.1.5 Mobile Device UI

Mobile phones are designed first and foremost for voice telephony, as are the controls on most of these devices. The controls are not optimized for use in games (except in the case of the Nokia N-Gage game deck).

Generally, developers can rely on the existence of a standard ITU-T keypad, two soft keys, and buttons for starting and ending phone calls; usually, the latter two cannot be addressed by games. In addition, all Series 40 and Series 60 Nokia phones have directional keypads.

Except for some PDAs, mobile devices do not have pointing devices; thus, games that rely on a pointing device will not work on mobile phones. Also, the MIDP specification does not require mobile phones to detect multiple simultaneous key presses (and no Series 40 Nokia phone does). Thus, the kind of "chording" typical of many console games is not feasible on mobile devices, and designers must avoid implying to players that it is.

Keypad layouts vary; in particular, the Nokia 3650 imaging phone has an unusual keypad layout. As a result, slightly different UI designs may be needed for different phones. For example, using 2 for "Up," 4 for "Left," 6 for "Right," and 8 for "Down" — as many mobile games do — will make the UI unintuitive when the game is played on a Nokia 3650 phone.

In J2ME implementations, this can be worked around by using MIDP's high-level "game actions" (Left, Right, Fire, and so on), which each device will map correctly to appropriate keys, rather than specifying particular keys in the code. MIDP's high-level actions are limited, however, so this approach is not feasible if your game includes "verbs" that involve actions other than specifying direction and "firing."

## 5.2 Working Within the Social Space of Mobile Games

PC or console games typically engage players for extended periods. Mobile games are quite different; they can be played anywhere, at any time — and they can be interrupted by phone calls at any moment.

### 5.2.1 Handling Interruptions Gracefully

Expect players to interrupt the game at any time. The game should be designed to be paused easily. Further, if a player receives a call and hits the "talk" button, the game should automatically save. Otherwise, the player will be quite irritated to find that he must start the game over.

In addition, give players a way to save a game in progress quickly and easily, so if their bus comes, or something of the sort, they can put the game away with confidence that they have not lost anything.

### 5.2.2 Go Easy on the Sound

A mobile game can be played anywhere — including places where others may become annoyed if the phone is continually bleeping and blooping. While audio can contribute materially to a game experience, it is important to give players a way to turn sound and music off. It is also usually advisable not to use background music, at least with Series 40 and similar devices. Sound consisting of simple beeps becomes annoying after a while; and proper volume control functionality may be missing, resulting in excessively loud music.

### 5.2.3 Keep the Backlight On

Generally, a mobile phone's backlight comes on whenever a key is pressed — but if a few seconds pass without any key presses, it goes off. Thus, if a player looks up from a game for a few seconds, he may return to find that he can't see the game. He can press a button to turn the light on again — but that button may be mapped to a game function, and the game meaning of the button may not be appropriate to the situation. In the worst case, a player may wind up losing the game just because he tried to see what was going on.

Design and implement the game so that the backlight remains on as long as the game is running.

# 6 The Design Specification

In a game development project, the designer's main task is to write the design specification.

Design specifications vary greatly from project to project and from company to company. As a basic rule, the more people involved in a project, and the less experienced they are at working together, the more detailed and precise a design specification needs to be. If you are designing, programming, and creating the art assets for a game all by yourself, you may need nothing more than a few scribbled notes to remind yourself of what you planned. Contrariwise, if you are working with a team of 12 on a multi-megabyte rich game for the Nokia N-Gage game deck, a highly detailed design specification is necessary to ensure that everyone understands what the game is supposed to be like, and what tasks need to be accomplished to complete it.

## 6.1 UI Specification

The design spec must specify the game's user interface. In the most extreme case, it should specify all screens in the game (title screen, initial menu, end-of-game screens, and so on), all possible menus, all possible button presses and their effects in context, and so on.

When specifying UI, keep in mind that new players need help. Include help screens, ideally ones that include an image of the keypad, labeled to indicate what each control does. Consider having text that appears over the gameplay area, particularly in early levels, identifying new features encountered by the player. Remember that no one reads the manual (or even help screens, if he or she can avoid it), so in-context help is very useful in getting people up to speed on how to play your game.

## 6.2 Gameplay Algorithm Specification

The design spec must also describe basic gameplay algorithms, in enough detail that the programmers will understand how to implement them. In general, a gameplay algorithm is any set of rules or calculations that affect response to player input or the behavior of objects or characters in the game.

Examples of algorithms you might need to specify include:

- Combat resolution. Is it simply a matter of facing in the right direction and shooting, with all targets dying after one hit? Or is it a complicated role-playing game system, with armor, weapon type, and character skill taken into account? What formulas are used?
- Movement of characters and objects. How fast do they move? Does terrain affect movement speed, and, if so, how? Do characters "walk," or is this a space game that implements basic Newtonian physics? If the latter, are there gravitational effects?
- AI routines. What rules are used to control the behavior of non-player characters and opposing "monsters"?
- Power-ups. What effect does each power-up have? How is it expressed, numerically or in other ways?

## 6.3 Level Design

Typically, an initial design spec does not specify every challenge the player faces, every level in the game, in detail. Often, level design proceeds in tandem with implementation of the game engine. Sometimes, written level designs are dispensed with, in favor of working directly with level design tools. Ultimately,

though, each level or challenge needs to be specified — unless levels and challenges are algorithmically generated, in which case the algorithms that generate them need to be specified.

At a minimum, the design spec should determine the number of levels — if only as a reality check, since application memory space on most mobile devices is highly constrained, and it is necessary to determine whether the desired number of levels can be implemented, given the other demands of the application.

### 6.4 Media Asset Specification

During development, you will need to draw up a document specifying exactly what media assets must be created for the game. This includes each and every frame of animation; background images; title screen and other graphics; each "event" sound and music clip; and so on.

In some cases, this is also part of the design specification; in others, it is a separate document. The document's creator must work closely with the technical lead to determine that the media assets are created in the format required by the program, and that a consistent naming scheme is used so that the programmers can easily import them.

### 6.5 High-Level Object Model

To ensure smooth and rapid development, the game's technical lead programmer will need to specify the high-level object model to be used in coding the game. If the designer has done his or her job, in terms of defining the allowable actions, gameplay algorithms, and possible items in the world, defining the high-level object model should be straightforward. If the game designer is technically proficient, he or she may work closely with the technical lead in creating this object model, and it may be part of the design specification. Often, however, the object model is considered part of another document, the technical specification, which is written by the technical lead rather than the game designer. The technical specification also defines target platforms and development tools.

### 6.6 Schedule, Budget, and Test Plan

Typically, the schedule, budget, and test plan are not part of the design specification, though sometimes, particularly in small teams, the game designer is responsible for some or all of these.

The schedule or project plan tracks the contributions of the people involved in the project, as well as the number of days each is expected to spend on the project, and at what times, to bring it to conclusion. The budget specifies the overall expenditure on the project, in terms of equipment and outside vendors as well as internal staff time and overhead.

The test plan is particularly important for mobile games, because games are typically expected to run on a range of mobile handsets, rather than on a single phone model; and since keypad layout, screen size, and specifics of software implementation can vary widely, it is generally necessary to test the game separately in each targeted handset.

It should be noted, however, that game development differs from other software development efforts in one important way: With non-game software, testers mainly ensure that the application performs as described in the design specification, and as long as it does, their job is done. A game, however, can perform exactly as it was originally specified — and still be dull or even actively annoying to play. Until a game is running, it can be hard to get a feel for how much (or little) fun it is. Thus, testing for games includes not only quality assurance, but also "play-testing." Play-testing means playing the game repeatedly, identifying unsatisfying aspects, specifying changes to improve them, working with the programmers to implement those changes, and testing again. The object is to create a fun game through

this iterative process. The designer should be intimately involved in the play-testing process — and the team should budget adequate time to ensure that play-testing is done.

# 7 Conclusion

Solo-play 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?

As with all solo-play games, the game system itself, rather than opposing players, must provide challenges for users. Typically, solo-play games do this with physical challenges, puzzles, and/or AI opponents.

Some special considerations apply to solo-play mobile games, however. In particular, the designer must be careful to design within the constraints of the target device, and pay particular care to ensuring a clean UI. As well, it is important to consider the context in which phones are used, to ensure a good experience for both the player and those around him or her.

# 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. |
| Avatar | The physical representation of a player's character in the game. Used only for games in which the player takes the role of a single character. |
| Cheat | A special, undocumented feature programmed into the game that, when triggered, gives players special powers (e.g., invulnerability). In a solo-play game, a player "hurts" only himself when he cheats, and players often enjoy discovering and playing with cheats. |
| Easter egg | An undocumented feature in a game that players can discover during play. Easter eggs are generally intended to be amusing, not to help or thwart the player in any way. |
| First-person shooter | 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. |
| God game | A type of game in which the player controls an entire civilization, nation, tribe, business, etc., rather than an individual character. Civilization is an example. |
| Graphic adventure | A type of game in which a player unlocks new areas of the game and advances the story by solving puzzles. Myst is an example. |
| Real-time strategy game | A type of game in which a player collects resources, builds military structures, raises an army, and engages in combat (in real time, rather than turn by turn) with opponents. Warcraft is an example. |
| Side-scroller | A type of game in which the player controls a single character who moves through a 2-D world, with the world scrolling behind the character. Sonic the Hedgehog is an example. |
| "Sliding square" puzzle | A classic puzzle consisting of 15 small squares arranged in a 4 x 4 frame, with one square left open. Squares adjacent to the open square may be slid into that square; the object of the puzzle is ultimately to move the squares into the right positions to display an image. |

| | |
|---|---|
| Text adventure | The predecessor to the graphic adventures. Text adventures also involve puzzle solving to unlock game areas, but the game is carried in text alone, without images. Zork is an example. |

# 9 References

Crawford, Chris. *Chris Crawford on Game Design*. Indianapolis, IN: New Riders Press, 2003.

Rollings, Andrew, and Dave Morris. *Game Architecture and Design*. Indianapolis, IN: New Riders Press, 2004.

Marc LeBlanc's game design site: http://world.std.com/~mahk/algorithmancy/

Costikyan, Greg. *I Have No Words & I Must Design*: http://www.costik.com/nowords.pdf

*Nokia Series 40 J2ME™ Game Usability Guidelines*: http://www.forum.nokia.com/games, then click "Enter Game Designers Page" in the Game Designers box.

*Developing Java™ Games for Platform Portability*:http://www.forum.nokia.com/games, then click "Enter Game Programmers Page" in the Game Programmers box.

# Build > Test > Sell

**Developing and marketing mobile applications and services with Nokia**

## Get started

Use free resources available through http://www.forum.nokia.com/: articles covering the latest technical and business issues, tools for developing and deploying applications and services, biweekly newsletters, and active discussion boards.

http://www.forum.nokia.com/

## Download tools and SDKs

Download free SDKs, emulators, simulators, and other tools that integrate with industry-leading integrated development environments.

http://www.forum.nokia.com/tools

## Get specifications and documentation

Get comprehensive device and platform specifications, and find detailed technical documentation, tutorials, technical notes, case studies, FAQs, and more.

http://www.forum.nokia.com/devices
http://www.forum.nokia.com/documents

## Get support and testing services

Access Nokia's wide range of technical support services, including case resolutions from our professional support staff and fee-based online support from our experts. Forum Nokia's Developer Hub services help with development and testing by providing both online and onsite access to servers, messaging centers, and the like.

http://www.forum.nokia.com/support

## Take your applications to market

Take your applications and services to market through Nokia Tradepoint and Nokia Software Market. Other opportunities are available through Nokia Mobile Phones and other Series 60 licensees.

http://www.forum.nokia.com/business

## Build your business with Nokia

Nokia works with selected leading developers in the games, branded media and content, and enterprise software industries. Submit your application to Nokia at http://www.forum.nokia.com/business.

http://www.forum.nokia.com/business