GameMaster Release 1.0

Programming GameMaster Rule Books

or

The Quotations of the Prophet Rhys

Version 1.0b3

Rhys Hollow and Quinn

2 December 1991

Table of Contents

∞

1. An Introduction to GameMaster

GameMaster is a Macintosh application that makes it very easy to create two player, network capable games. The principle behind GameMaster is very simple. GameMaster itself is a standard Macintosh application that handles all of the usual drudgery associated with programming a Macintosh. All that is required of the game author is to create a file that encapsulates the rules and display of the game. The file is known as the *rule book*. These rule book files are placed in the same folder as GameMaster.

When running GameMaster the user sees a menu that allows them to start a game using any of the available rule books. Thus each game is played according to a particular rule book. A window is created in which the game is displayed. The exact operation of this *game window* is determined by the rule book.

1.1. Network Capabilities

Any properly written GameMaster rule book will automatically generate network capable games. The rule books have a very simple model of networking. GameMaster handles all the usual horror associated with network programming.

GameMaster uses both AppleTalk (using ADSP) and MacTCP to communicate with other GameMasters running on other Macintoshes. The MacTCP alternative was provided because it allows for world-wide game play.

1.2. The Purpose of this Manual

This manual was written to help Macintosh programmers learn how to write GameMaster rule books. It is provided bundled with a number of other files as the GameMaster Developer's Kit. The rest of this manual is dedicated to describing exactly what GameMaster expects of its rule books.

1.3. The History of GameMaster

GameMaster was written by Rhys Hollow (gurhs@uniwa.uwa.oz.au) in late 1991. The AppleTalk support was written by Marcus Jager (marcus@cs.uwa.oz.au). The MacTCP support is courtesy of Peter Lewis (nlewispn@cc.curtin.edu.au). This manual was written by Quinn (quinn@cs.uwa.oz.au) and based on Rhys' programming docu-

mentation. The game rules were written by Rhys (Othello, BattleShips, Connect 4), Peter (Simple Talk, File Transfer, Global Thermonuclear War, Derf Ball, Dots and Boxes), and Quinn (Chess and TicTacToe).
GameMaster was written in Think Pascal 4, with help from ResEdit 2.1.1. The program was mostly developed on a Macintosh Classic. The documentation was written in WriteNow 2.2.

1.4. Contacting the Authors

If you wish to contact the author of GameMaster you can Email Rhys at his address above. Please make sure you include the word GameMaster in the subject line.
If you are reporting bugs (Bugs! What bugs!) please quote the system on which you were running (ie hardware, system software, extensions) and the version of GameMaster you were using. Please report bugs in rule books to the author of the rule book.
Please send any comments regarding this manual to Quinn at the address above.
∞

2. An Overview of Rule Books

A GameMaster rule book is a file of type ‘GMRB’ and creator ‘GMST’. Its data fork should be empty. The resource fork contains a number of fixed resources that describe the game, a resource that contains the code that embodies the game rules, and an arbitrary number of rule book resources. A more thorough description of the resource fork’s contents can be found in Chapter 3.

2.1. Game Events

GameMaster works on the following principle. When launched (and subsequently when idle) it creates a list of all the rule books in the folder from which it was launched. The user can then select New Game and choose the rule book that they would like to use. GameMaster opens the rule book’s resource fork, checks it for integrity, and then loads the rule book’s code. From then on the dialog between GameMaster and the rule book is based around GameMaster calling the rule book whenever certain events happen. Unlike a standard Macintosh application, the rule book is not called for trivial events. GameMaster only calls the rule book for events that it can’t handle. A complete description of each event can be found in Chapter 5. In addition rule books can indicate to GameMaster that they are not interested in specific events.

2.2. Game Data Structures

Each time GameMaster calls a rule book it passes a reference to a data structure known as the *game event record*. This record contains information telling the rule book what to do. The rule book can also change the record in order to pass information back to GameMaster. The structure of the game event record is described in Chapter 4.

The first time it calls a rule book GameMaster allows the rule book to store a 4 byte quantity into the game event record’s globals field. Every subsequent time GameMaster calls the rule book it will pass back that value in the globals field of the game event record. This allows the rule book to create a handle or pointer to its global variables and access them on each event.

The rule book is shared between all games played with that particular set of rules. Each instance of the rules (each game) has a *game handle* that stores the state of the

game. GameMaster creates an empty handle and stores it in the game field of the game event record. The rule book is free to grow and shrink this heap block at will. GameMaster expects that this block of memory will contain all of the relevant state information required to start the identical game on a remote machine or to restore the game from a saved file. However, GameMaster sends a ge_Flatten event to warn the rule book immediately before it relies on this fact.

2.3. Game Display

For each active game GameMaster creates a modeless dialog window that the game uses to display the game state to the user. The rule book provides two important data structures used to create this window. The first is a dialog item list. GameMaster bases the game's window on this list. The second is the width and height of the *play area* as determine by the ge_InitRuleBook event. The play area is a rectangle within the game window with its top-left at (0,0) and its width and height determined by the rule book.

GameMaster imposes certain restrictions on the size of the play area brought about by the dialog procedure described in the next section. Firstly the width of the play area must be more than 135 pixels. Secondly the play area must fit on the screen. Unless you want to support calculating the play area at run-time this means that it must fit on the standard 9 inch screen of a Mac Classic (512 by 342). Given that you lose 20 pixels to the menu bar, 20 pixels to the window title bar, and 40 to 50 pixels to GameMaster's controls, this means that the maximum height of the play area is about 260.

All game window updates are done through the dialog item list. The rule book can have control, PICT, ICON, button, checkbox, radio button, static text and edit items text in this list. These are updated automagically by the Dialog Manager. The rule book can also have user items for any dynamic components. The rule book can then set each user item's update procedure on the ge_NewGame and ge_OldGame events.

Rule books can also add and remove menus from the menu bar and be notified when the user selects these menu items.

2.4. Game Dialogs

GameMaster often needs to interact with the user. The most common case of user

interaction is asking the user whether they would like to play a game. As we all know ‘modal bad, modeless good’ so GameMaster was designed to support non-modal interaction. It achieves this by adding a small panel to the bottom of each game window.

Three states of the Dialog Panel

When the game requires user interaction, GameMaster slides down this panel to reveal a message and optionally some buttons. This panel is used for two purposes. Firstly it can display a message informing the user of the status of the game (centre). Secondly it can display a mini-dialog (right). From there the game does not continue until the user dismisses the dialog by clicking on the buttons. In both of these states the game is known as *dialoging*. When a game is dialoging no user input events are sent to its rule book.

This Dialog Panel affects a rule book in three ways. Firstly, the rule book designer must take into account the size of this panel when laying out the rule book’s display. Secondly, the rule book can display its own message and buttons in the panel using ge_Ask events. Lastly, any real-time rule book should note that when the Dialog Panel is down user input to the game is ignored.

2.5. Game Event Environment

Before sending any events to the rule book GameMaster does the following things:–

- Calls SetPort to set the current GrafPort to that of the game’s window, except for events that can be sent to a rule book when there is no current game ie ge_InitRuleBook and ge_FinishRuleBook.
- Calls UseResFile to make the rule book’s resource file the first to be searched.
- Fills out the appropriate fields in the game event record

Note that there may be other resource files (specifically GameMaster’s and other rule books’) in the resource search path. If the rule book modifies its resource fork (definitely not recommended) then it must be certain that it only modifies its own resources.

2.6. Rule Books and Memory

The rule books share their heap with GameMaster. Thus the rule books must be careful

about the amount of memory that they allocate. As part of its required structure a rule book must contain a resource that describes how much memory it uses, both to load the rule book and to start a new game using that rule book. GameMaster prevents the user from playing a game unless there is sufficient memory to fulfil these requirements. However, this does not mean the rule book will never run out of memory because other rule books might overrun their memory allocations.

2.7. Version Numbering

GameMaster rule books can be connected to one another across networks. It would be very bad if GameMaster connected two incompatible version of a rule book. To prevent this the rule book indicates to GameMaster the following version numbers:–

- The rule book version number.
- The oldest version number of the rule book to which this rule book can be safely connected.
- The version number of the GameMaster that this rule book was written for.
- The oldest version of GameMaster with which this rule book can be used.

Each version number is an integer (two bytes). This integer is interpreted as a 4 digit BCD number. The first nibble is ignored and should be zero. The second nibble is the major version number, the third the minor version number and the fourth the bug fix version number.

When GameMaster displays version numbers (such as in the about box) it displays each of these nibbles in decimal separated by a period (‘.’). For example a version number $0123 yields version 1.2.3.

∞

3. The Resources

The resources in a rule book's resource fork can be divided into three classes: required resources, optional resources and rule book resources. Much of the rest of this chapter is dedicated to describing the required resources, in particular the GMRZ resource. The final section details the optional resources. Rule book resources are entirely rule book dependent. However, it is necessary to first describe how required, optional and rule book resources are distinguished.

3.1. Resource ID Allocation

The resource IDs less than 128 are reserved by the system.
The resource IDs between 128 and 999 are reserved for use by GameMaster. This leaves the resource ID range 1000 to 32767 for use by the rule book resources.
Note that all required and optional resources have IDs between 128 and 999. Also note that the rule book may contain certain system resources with IDs outside of the allocated range, for example 'vers' 1 and 2, 'ICN#' -16455.

3.2. Required Resources

GameMaster requires that the rule book contain the following resources:–

'GMRZ' 128
This resource contains important information that GameMaster needs to know about each rule book. Its structure is quite complex and it is described in detail in the next section.

'PROC' 128
This resource is the code that makes up the core of the rule book. The resource must start with a procedure with the following interface:–

```
procedure MyGameRules(var gameevent : gameEventRecord);
```

Naturally the procedure identifier (ie MyGameRules) is unimportant.

‘DITL’ 128

This dialog item list controls the overall layout of the window associated with any game that uses this rule book. The item list can contain any of the standard dialog items. When GameMaster starts a new game it creates a modeless dialog based on this DITL. GameMaster adds its own items after the rule’s items. Thus the rule book should not rely on the length of the item list.

Note that this DITL resource should be purgeable. Also note that any secondary resources (PICTs, ICONs, etc) are rule book resources and must have resource IDs greater than 1000.

3.3. The GMRZ Resource

The GMRZ has a large number of fields of which the rule book designer should be aware. Each of the fields of the GMRZ resource is described in turn below. Note that the type identifiers are taken from the ResEdit 2.x template supplied as part of the GameMaster Developer’s Kit.

PO3F Name of the Game

This string represents the name of the game that the rule book plays. It is used to display the name of the game in the about box.

PO7F Author

Your name. Wow, you’re world famous!

PO7F Copyright String

This string is provided so you can legally retain your claim to fame.

PO3F ID String

This string identifies the rule book to GameMaster. This string is used when GameMaster connects with another GameMaster across the network. It is compared with the corresponding string on the remote machine to make sure that the rule books are the same. This is important because the user can rename rule books from the Finder and connecting two games based on different rule books would be bad!

HWRD Rule Book Version

This word is the version number of the rule book. The format of this (and subsequent) version numbers was described in Section 2.7.

HWRD Required Rule Book Version

This word is the oldest version of this rule book with which this rule book can safely be connected.

HWRD GameMaster Version

The version number of GameMaster under which this rule book was developed.

HWRD Required GameMaster Version

The oldest version of GameMaster with which this rule book can be safely used.

DLNG Rule Book Partition

The number of bytes that must be free for this rule book to be used. If there are no games currently being played with this rule book, GameMaster adds this number to the Game Memory Partition to determine the total amount of space required for a new game.

DLNG Game Memory Partition

The number of bytes taken by each new game using this rule book.

BBITDisallow Save

If this bit is set then the user will not be able to save games played with this rule book. Note that unless the Disallow Connect bit is also set the rule book can still be sent ge_OldGame events as part of the network connect process.

BBITDisallow Connect

If this bit is set then the user will not be able to connect games made with this rule book. You can set this for rule books that do not support networking.

BBITDisallow Restart

If this bit is set then the user will not be able to restart games played with this rule book.

BBITDisallow Swap

If this bit is set then the user will not be able to swap games played with this rule book. Swapping is only possible when two games are connected and it involves changing which player the local machine is playing. Note that unless the Disallow Connect bit is also set the rule book can still be sent ge_Swap events as part of the network connect process.

BBITNo Idle Events

If this bit is set then no ge_Idle events will be sent to this rule book. Idle events are only useful if you wish to animate your display or write a real-time game. Most rule books can safely set this bit.

BBITNo SleepQuery Events

If this bit is set then no ge_SleepQuery events will be sent to this rule book. ge_SleepQuery events are used to time the idle events and change the shape of the cursor. Set this if you never change the shape of the cursor and you do not require idle events.

BBITReserved0

This bit is reserved for future expansion and should be clear.

BBITReserved1

This bit is reserved for future expansion and should be clear.

DBYT Reserved2

This bit is reserved for future expansion and should be clear.

DWRD Reserved3

This bit is reserved for future expansion and should be clear.

3.4. Optional Resources

‘PICT’ 128

If this picture is present GameMaster will put it in the rule book’s about box. The

picture must be wider than 200 pixels and otherwise such that the about box fits on the screen.

∞

4. The Game Event Record

The game event record has the following structure:–

```
gameEventRecord =
        record
            event: gameEventType;
            game : Handle;
            globals : Handle;
            where : Point;
            modifiers : integer;
            message : Str255;
            but1, but2 : buttonStr;
            int1, int2 : integer;
            long1 : longint;
            modified : boolean;
            myturn : boolean;
            version : integer;
            dialoging: boolean;
            dumspace: array[1..10] of longint;
        end;
```

The following sections describe each field in detail.

event : gameEventType; Read/Write

The event field is set appropriately every time GameMaster calls the rule book. The value of the event field determines the semantics of the message passed to the rule book. The definition of gameEventType is as follows:–

```
gameEventType = (ge_InitRuleBook, ge_FinishRuleBook,
        ge_NewGame, ge_OldGame, ge_Swap,ge_Restart, ge_Close,
        ge_Activate, ge_Deactivate, ge_UpdateMenus, ge_Menu,
        ge_MouseDown, ge_KeyDown, ge_Idle, ge_SleepQuery,
        ge_ConnectionMade, ge_MessageReceived, ge_ConnectionLost,
        ge_Ask, ge_Answer, ge_Error, ge_SendMessage,
        ge_Flatten);
```

The semantics of each of the values of the event field is discussed in Chapter 5. The rule book can also modify the event field in order to tell GameMaster to perform certain actions.

If the rule book receives any game event records with the event field set to a value not in this list then it should ignore the event.

game : Handle; Read Only

The game field holds a handle to the a heap block that the rule book uses to store the state information (or game record) for each game. The rule book must cast the handle to the appropriate type of its game storage. The handle is both allocated and deallocated by GameMaster but the rule book is free to resize it at will. If the game field is nil then this game record has not yet been created or has already been destroyed.

globals : Handle; Read/Write

The globals field can be used by the rule book to hold any arbitrary data it requires. Normally this would be either a pointer or a handle to its global data. The rule book must cast the globals handle to the type of the globals storage that it uses. The globals field is the same for every game played with the rule book. The only time that the rule book is called without a valid globals field in the game event record is on ge_InitRuleBook at which time it is expected to set it up.

where : Point; Read Only

The where field is used to communicate the coordinates of the mouse in local coordinates on certain events.

modifiers : integer; Read Only

The modifiers field is used to communicate the current modifiers to the rule book on certain events. The bits in the field have the same meaning as the modifiers field in the Toolbox EventRecord.

message : str255; Read/Write

The message field is used to hold the text of messages displayed to the user and also messages sent between rule books connected via the network. The length of the string in the message field should always be less than 250 characters.

but1, but2 : buttonStr; Read/Write

The but1 and but2 fields are used to hold the labels for buttons that the rule book can get GameMaster to display using the ge_Ask event. The buttonStr type is defined as follows:–

```
buttonStr = string[7]
```

The buttons are of a fixed size and you should make sure that your button strings fit.

int1, int2 : integer; Read/Write

The int1 and int2 fields are used to hold generic integers. The semantics of these fields is event dependent.

long1 : longint; Read/Write

The long1 field is used to hold a generic longint. The semantics of this field is event dependent.

modified : boolean; Read/Write

The modified field is used to keep track of whether the game referenced by the game event record has been modified since it was last saved. Set this field to true if you modify anything important in the game state.

myturn : boolean; Read/Write

The myturn field is used to keep track of whether GameMaster should send you certain events. If the game has not been connected then the rule book should set myturn to true to receive all events. If the game has been connected then the state of myturn determines which game is in control. If myturn is false then GameMaster will not send you the following events:–

- ge_MouseDown
- ge_KeyDown

The reason why myturn exists is to prevent all sorts of horrible race conditions while the connection is being arbitrated.

If the local game is in control then the local myturn field will be true. If the remote game is in control then the local myturn field will be false. If both games want control then both games should have myturn set. It would be extremely unusual for both games to have myturn false.

version : integer; Read Only

The version field holds the version number of the GameMaster that is executing this rule book. The format is the same as that described in Section 2.7 and used in the GMRZ resource.

dialoging : boolean; Read Only

This boolean is true if GameMaster was displaying a dialog at the time it sent the event to the rule book. If you try to display a dialog while GameMaster is already displaying your dialog will not be displayed.

dumspace : array [1..10] of longint; no access

The dumspace field provide space for future expansion of the GameMaster/rule book interface. You should neither change nor rely on the contents of this array.

∞

5. The Game Events

When GameMaster detects certain conditions, or requires specific information, it sends the rule book an event. The meaning of each event is determined by the event field of the game event record. The events can be grouped into 6 classes: Rule Book Open/Close, Game Open/Close, Game Communications, Game Control, Game Input, Game Return.

5.1. Notation Conventions

Each event described has a summary in the following form:–

gameEventRecord

field	fieldType	∫	comment
field	fieldType	fi	comment
field	fieldType	‹	comment
field	fieldType	€	comment

The first column is the field name. The next column is the type of the field. The third column shows who is responsible for setting the field. The last column gives a comment as to what the field contains.

If the field is marked with a congruent (∫) then the contents of that field are set to a particular value. The rule book can rely on that value being present but must not change it.

If the field is marked with an out arrow (fi) then the rule book must set the field to an appropriate value before returning to GameMaster. Unless otherwise noted the value of the field on entry is undefined.

If the field is marked with an in arrow (‹) then the field contains an appropriate value on entry. Unless otherwise noted GameMaster doesn't care what the value is when you return.

If the field is marked with a bidirectional arrow (€) then GameMaster sets the field to a value which the rule book can use. The rule book must update (if necessary) that value before it returns.

If the field is missing then the value of the field is undefined and the rule book must neither rely on nor change it.

5.2. Rule Book Open/Close Events

These Rule Book Open/Close Events are special in that they are the only events that can be sent to the rule book without a valid game. Thus the game field of the game event record is nil and thePort is not set to a game window.

ge_InitRuleBook

GameMaster guarantees that the first event it will ever send to a rule book is the ge_InitRuleBook event. The rule book can store any 4 byte value in the globals field and perform any other initialisation. Normally the rule book would allocate any global data on the heap and store a pointer or handle to it in the globals fields.

gameEventRecord

event	gameEventType	‹	ge_InitRuleBook
version	integer	∫	GameMaster version
globals	Handle	fi	handle to rule globals
int1	integer	fi	play area width
int2	integer	fi	play area height
dialoging	boolean	∫	true if dialog currently displayed

On return from this event GameMaster expects that the width of the game's play area be stored in int1 and the height in int2.

ge_FinishRuleBook

GameMaster guarantees that the last event it will ever send to a rule book (before closing its resource fork) is the ge_FinishRuleBook event. The rule book should use this opportunity to deallocate any global storage it has. Setting the globals pointer to nil is also a good idea.

gameEventRecord

event	gameEventType	‹	ge_FinishRuleBook
version	integer	∫	GameMaster version
globals	Handle	‹	handle to rule globals
		fi	nil
dialoging	boolean	∫	true if dialog currently displayed

5.3. Game Open/Close Events

The Game Open/Close Events provide the basic control necessary to start new and old games, restart games and close games.

ge_NewGame

When the user creates a game using the New Game menu item GameMaster sends this event to the rule book. GameMaster has already created a zero sized handle and stored it in the game field. The rule book should grow this handle and initialise it to the appropriate initial state.

gameEventRecord

event	gameEventType	‹	ge_NewGame
version	integer	∫	GameMaster version
game	Handle	∫	game record handle
globals	Handle	∫	handle to rule globals
modified	boolean	∫	false
myturn	boolean	‹	true
		fi	whether it's the game's turn
dialoging	boolean	∫	true if dialog currently displayed

On return GameMaster expects that the game handle be filled out with an initial game state.

ge_OldGame

When the user resumes a saved game GameMaster sends the ge_OldGame event to the rule book. The game handle has already been filled out with the saved game state. The rule book should overwrite any fields in the game which are inappropriate to load from disk and return.

gameEventRecord

event	gameEventType	‹	ge_OldGame
version	integer	∫	GameMaster version
game	Handle	∫	game record handle
globals	Handle	∫	handle to rule globals
modified	boolean	∫	false
myturn	boolean	‹	true
		fi	whether it's the game's turn
dialoging	boolean	∫	true if dialog currently displayed

The ge_OldGame event is also used as part of the network connect process.

ge_Swap

The ge_Swap event is sent to a rule book when the user chooses Swap from the GameMaster menus. Swapping is only meaningful when two games are connected. The idea behind swap is that it should change what side the each player is playing. For example, if Fred was playing Black and Sheila white, then after a swap Fred would be playing White and Sheila black.

gameEventRecord

event	gameEventType	‹	ge_Swap
version	integer	∫	GameMaster version
game	Handle	∫	game record handle
globals	Handle	∫	handle to rule globals
modified	boolean	€	true if the game has been modified
myturn	boolean	€	whether it's the game's turn
dialoging	boolean	∫	true if dialog currently displayed

The ge_Swap event is also used as part of the network connect process.

ge_Restart

The ge_Restart event is sent to a rule book when the user chooses Restart from the GameMaster menus. If the game is connected then both sides should organise to be playing different players after the restart.

gameEventRecord

event	gameEventType	‹	ge_Restart
version	integer	∫	GameMaster version
game	Handle	∫	game record handle
globals	Handle	∫	handle to rule globals
modified	boolean	€	true if the game has been modified
myturn	boolean	€	whether it's the game's turn
dialoging	boolean	∫	true if dialog currently displayed

The rule should reinitialise the contents of the game handle to that of a new game.

ge_Flatten

GameMaster sends the ge_Flatten event to a rule book when it needs to use the contents of the game handle. Normally this means that the game is about to be saved or sent across the network. The game handle must remain ‘flat’ until another event is sent to the rule book.

gameEventRecord

event	gameEventType	‹	ge_Flatten
version	integer	∫	GameMaster version
game	Handle	∫	game record handle
globals	Handle	∫	handle to rule globals
dialoging	boolean	∫	true if dialog currently displayed

If the game state as recorded in the game handle is out of date then the game should update the game handle. The ge_Flatten event gives the rule book writer the option of storing pointers in the game record for efficiency while still allowing games to be saved and connected across the network.

ge_Close

GameMaster sends the ge_Close event to a rule book when the user closes one of its game windows.

gameEventRecord

event	gameEventType	‹	ge_Close
version	integer	∫	GameMaster version
game	Handle	∫	game record handle
globals	Handle	∫	handle to rule globals
dialoging	boolean	∫	true if dialog currently displayed

This event is simply a chance for the rule book to do any cleaning up not covered by other events. Specifically, if the rule book stores pointers in the contents of the game handle (and only there) it should note that the game handle is about to be disposed.

5.4. Game Communications Events

The Game Communications Events provide the framework by which rule books on separate machines can talk.

ge_ConnectionMade

The ge_ConnectionMade event is sent to a rule book when the game referenced in the game event record is connected to to a remote game. GameMaster uses the game state stored in the game record to ensure that both the local and remote players are playing the same game.

gameEventRecord

event	gameEventType	‹	ge_ConnectionMade
version	integer	∫	GameMaster version
game	Handle	∫	game record handle
globals	Handle	∫	handle to rule globals
modified	boolean	∫	true if the game has been modified
myturn	boolean	‹	true
		fi	whether it's the game's turn
dialoging	boolean	∫	true if dialog currently displayed

The game should update the game handle to allow for the fact that messages may now be sent and received.

ge_ConnectionLost

The ge_ConnectionLost event is sent to a rule book when the connection to another game is lost for some reason, possibly network troubles, the remote machine has crashed, or the other player has just closed his end.

gameEventRecord

event	gameEventType	‹	ge_ConnectionLost
version	integer	∫	GameMaster version
game	Handle	∫	game record handle
globals	Handle	∫	handle to rule globals
modified	boolean	∫	true if the game has been modified
myturn	boolean	‹	whether it's the game's turn
		fi	true
dialoging	boolean	∫	true if dialog currently displayed

ge_MessageReceived

The ge_MessageReceived event can only be sent if the game is connected to another remote game, that is between a ge_ConnectionMade and a ge_ConnectionLost event. When a game receives a message it should perform any action associated with that

message.

gameEventRecord

event	gameEventType	‹	ge_MessageReceived
version	integer	∫	GameMaster version
game	Handle	∫	game record handle
globals	Handle	∫	handle to rule globals
message	Str255	‹	message from the remote game
modified	boolean	€	true if the game has been modified
myturn	boolean	€	whether it's the game's turn
dialoging	boolean	∫	true if dialog currently displayed

The other connection event is ge_SendMessage. This is a Game Return Event and is discussed in Section 5.7.

5.5. Game Control Events

The Game Control Events are designed to allow rule books to use menus. They also give the rule books notification of window (de)activation. The rule books can also use these events for their own purposes.

ge_Activate

The ge_Activate event is sent to a rule book when one of its game windows is activated (ie brought to the front).

gameEventRecord

event	gameEventType	‹	ge_Activate
version	integer	∫	GameMaster version
game	Handle	∫	game record handle
globals	Handle	∫	handle to rule globals
modified	boolean	∫	true if the game has been modified
myturn	boolean	∫	whether it's the game's turn
dialoging	boolean	∫	true if dialog currently displayed

The rule book should perform any action associated with bringing the window to the front. One probable action is to add menus to the menu bar. Note that if the rule book adds menus it should make sure that the menu ID is greater than 1000 and call DrawMenuBar.

ge_Deactivate

The ge_Deactivate event is sent to a rule book when one of its game windows is deactivated (ie another window is brought to the front).

gameEventRecord

event	gameEventType	‹	ge_Deactivate
version	integer	∫	GameMaster version
game	Handle	∫	game record handle
globals	Handle	∫	handle to rule globals
modified	boolean	∫	true if the game has been modified
myturn	boolean	∫	whether it's the game's turn
dialoging	boolean	∫	true if dialog currently displayed

The rule book should perform any action associated with deactivating the window. One probable action is to remove menus added on the ge_Activate. Note that if the rule book removes menus it should redraw the menu bar.

ge_UpdateMenus

GameMaster sends this event to the rule book associated with the active game window when the user clicks in the menu bar or selects a command key. This gives the game an opportunity to update the menus before the user sees them.

gameEventRecord

event	gameEventType	‹	ge_UpdateMenus
version	integer	∫	GameMaster version
game	Handle	∫	game record handle
globals	Handle	∫	handle to rule globals
modifiers	integer	∫	event modifiers
modified	boolean	€	true if the game has been modified
myturn	boolean	€	whether it's the game's turn
dialoging	boolean	∫	true if dialog currently displayed

The rule book should dim, underline, check, etc any menus and/or menu items on menus that it has added to the menu bar.

ge_Menu

When the user selects a menu item, either through the menu bar or its command key

equivalent, from one of the menus that GameMaster is not responsible for it sends the ge_Menu event to the rule book associated with the active game window.

gameEventRecord

event	gameEventType	‹	ge_Menu
version	integer	∫	GameMaster version
game	Handle	∫	game record handle
globals	Handle	∫	handle to rule globals
modifiers	integer	∫	event modifiers
int1	integer	∫	menu id
int2	integer	∫	item number
modified	boolean	€	true if the game has been modified
myturn	boolean	€	whether it’s the game’s turn
dialoging	boolean	∫	true if dialog currently displayed

The rule book should perform any action associated with the menu selection.

5.6. Game Input Events

The Game Input Events are used to send user input to the rule books.

ge_MouseDown

The ge_MouseDown event is sent whenever the user clicks in one of the enabled items in the game’s item list. Note that GameMaster has already called DialogSelect and sets int1 to the item number of the selected item.

gameEventRecord

event	gameEventType	‹	ge_MouseDown
version	integer	∫	GameMaster version
globals	Handle	∫	handle to rule globals
game	Handle	∫	game record handle
where	Point	‹	location of mouse down (local coords)
modifiers	integer	∫	event modifiers
int1	integer	∫	dialog item number
modified	boolean	€	true if the game has been modified
myturn	boolean	€	whether it’s the game’s turn
dialoging	boolean	∫	true if dialog currently displayed

The rule book can perform any action associated with the mouse down.

ge_KeyDown

When the user presses a key a ge_KeyDown event is sent to the rule book associated with the active game window. The modifiers are sent in the modifiers field. The ASCII of the key pressed is in int1 and the key code in int2.

gameEventRecord

event	gameEventType	‹	ge_KeyDown
version	integer	∫	GameMaster version
game	Handle	∫	game record handle
globals	Handle	∫	handle to rule globals
modifiers	integer	∫	event modifiers
int1	integer	∫	char code
int2	integer	∫	key code
modified	boolean	€	true if the game has been modified
myturn	boolean	€	whether it's the game's turn
dialoging	boolean	∫	true if dialog currently displayed

Note that command keys equivalents are intercepted by GameMaster and are sent as ge_Menu events.

ge_Idle

The ge_Idle event is sent to each game periodically. It allows games to perform events in pseudo-real time.

gameEventRecord

event	gameEventType	‹	ge_Idle
version	integer	∫	GameMaster version
game	Handle	∫	game record handle
globals	Handle	∫	handle to rule globals
modifiers	integer	∫	event modifiers
modified	boolean	€	true if the game has been modified
myturn	boolean	€	whether it's the game's turn
dialoging	boolean	∫	true if dialog currently displayed

ge_SleepQuery

The ge_SleepQuery is only sent to the game with the frontmost window and performs two functions. Firstly it allows the rule book to specify how often it would like to receive ge_Idle events. Secondly it gives the rule book an opportunity to force an idle

event if the mouse moves. This is useful when the rule book changes the cursor shape to reflect what it is over.

gameEventRecord

event	gameEventType	‹	ge_SleepQuery
version	integer	∫	GameMaster version
game	Handle	∫	game record handle
globals	Handle	∫	handle to rule globals
int1	integer	fi	the number of ticks before the next idle event
long1	longint	‹	nil
		fi	a region handle describing the current cursor region
modified	boolean	€	true if the game has been modified
myturn	boolean	€	whether it's the game's turn
dialoging	boolean	∫	true if dialog currently displayed

GameMaster will send an idle event to the rule book if either the mouse leaves the cursor region or int1 ticks pass.
GameMaster will dispose of the region returned in long1.

ge_Answer

The ge_Answer event is sent to a rule book when the user presses one of the buttons brought up by the ge_Ask.

gameEventRecord

event	gameEventType	‹	ge_MouseDown
version	integer	∫	GameMaster version
game	Handle	∫	game record handle
globals	Handle	∫	handle to rule globals
int1	integer	‹	which button was pressed
modified	boolean	€	true if the game has been modified
myturn	boolean	€	whether it's the game's turn
dialoging	boolean	∫	true if dialog currently displayed

The value of int1 is 1 if button 1 was pressed and 2 if button 2 was pressed. If int1 is 0 then the last ge_Ask failed to complete. This is most probably because GameMaster is already displaying its own dialog. Do not, under any circumstances, respond to this by sending another ge_Ask. You can check the value of the dialoging boolean to determine if GameMaster is already displaying a dialog.

5.7. Game Return Events

A rule book can send events back to GameMaster to request certain actions. It does this by changing the event field of the game event record. The event record thus returned is known as a Game Return Event. Only the fields specific to each Game Return Event are documented below. If it is not mentioned then a field retains the meaning associated with the original event.

ge_SendMessage

The ge_SendMessage event can be sent back on any event after (and including) the ge_ConnectionMade and before (but not including) the ge_ConnectionLost.

gameEventRecord

event	gameEventType	∫	ge_SendMessage
message	Str255	fi	message to send to remote game

After receiving a ge_SendMessage, GameMaster will send a ge_MessageReceived to the remote game with the message set to the message returned.
Note that the message string is limited to 250 characters even though 255 characters are allocated for it.

ge_Error

The ge_Error event can be sent back on any event. The rule book uses this event to signal that an unrecoverable (bad!) error has occurred.

gameEventRecord

event	gameEventType	∫	ge_SendMessage
message	Str255	fi	error message
int1	integer	fi	error number

Upon receiving a ge_Error, GameMaster will notify the user of the error and shut down the game associated with that message. Rule books should most probably be aware that, at the moment, GameMaster completely ignores ge_Error events! It was provided purely as a mechanism for rule books to be compatible with a future GameMaster with improved error handling.

ge_Ask

The ge_Ask event can be sent back by the rule book to ask the user a simple question.

gameEventRecord

event	gameEventType	∫	ge_SendMessage
message	Str255	fi	question to ask
but1	buttonStr	fi	string for first button
but2	buttonStr	fi	string for second button

Upon receiving a ge_Ask, GameMaster will put up a dialog asking the user a question. The message is displayed. If but1 is not empty then a button is displayed with that text. If but2 is not empty then a button is displayed with that text. GameMaster waits for the user to press one of the buttons and then sends a ge_Answer event. If no button is displayed then the user must simply wait. The rule book can cancel this wait state by sending a ge_Ask event with message, but1 and but2 all empty.

To be compatible with the dialogs built in to GameMaster but1 should be ‘No’ or ‘Cancel’ and but2 ‘OK’ or ‘Yes’. This is obviously subject to the semantics of the rule book.

∞

6. Programming Rules: Hints and Tips

This chapter is a collection of hints and tips for programming GameMaster rule books.

6.1. GrafPorts, Dialogs and Windows

Before calling the rule book GameMaster sets the port to the game window. The rule can use GetPort to find the address of this GrafPort. Because a WindowRecord is an extension of a GrafPort and a DialogRecord is an extension of a WindowRecord the rule can cast the result of the GetPort to a WindowPeek or DialogPeek. Note that Inside Macintosh defines GrafPtr, WindowPtr and DialogPtr to be equivalent types.

6.2. Drawing

One of the hardest parts of writing a rule book is giving the user item update procedures enough information to draw the item. The user item update procedure has the following interface:–

```
procedure UserItemUpdate(w : WindowPtr; itemNo : integer);
```

This gives the procedure access to the WindowRecord (by casting w to a WindowPeek and indirecting) which in turn gives access to the refcon. The rule book can place the game handle into this refcon (on a ge_NewGame/ge_OldGame), thereby giving the procedure access to the game handles contents. The rule book can also store the globals handle in its game record. This gives the procedure access to the rule book's globals.

6.3. Accessing QuickDraw Globals

Theoretically the rule books should have a valid A5 world. This means that it can quite easily access GameMaster's QuickDraw globals. It should not modify them or access the application globals.

At the moment we can't get Think Pascal to do this for us nicely. However, we do supply a unit called QuickDrawRules.p that gives rule books easy access to the QuickDraw globals.

6.4. Debugging

Sorry guys but you can't use the Think Pascal integrated debugger to debug your rule books. At the moment it's MacsBug only. However to help you GameMaster exports a symbol (ShutDownGameMaster). When your rule book (or for that matter GameMaster) crashes you can enter 'g ShutDownGameMaster ' to quit GameMaster gracefully, shutting down MacTCP as you go.

Do not enter 'g ShutDown' for short. It's not much fun!

6.5. Standard State

We have found it convenient to maintain three state variables in our game records. They are:–

- connection_state (local or remote)
- local_player (black or white) If connection_state is remote then local_player determines who we're playing.
- player_to_move (black or white) The player who should move next.

Thus you can set myturn using:–

```
myturn :=
            (connection_state=local) or
                ((connection_state=remote) and
                    (local_player=player_to_move));
```

6.6. Rule Book Miscellanea

A collection of important things to remember:–

- Always make you resource IDs greater than 1000. ResEdit 2.x has the annoying habit of giving new created resources the default ID of 128. This is especially dangerous when you create a PICT item using the nice DITL editor.
- Menu IDs must be greater than 1000. Remember that there is a difference between menu IDs and the ID of their menu resource. Learn how to set the menu ID in ResEdit (select 'Edit Menu & MDEF ID' from the 'MENU' menu).
- Remember to enable a dialog item if you want to receive mouse clicks on it.
- Remember you do not need to track dialog controls. However if you simulate a button using a user item you should track it properly (ie simulate the behaviour of toolbox buttons).
- Remember that the game handle you are passed on a ge_NewGame is a valid empty handle. Call SetHandleSize to increase its size.

- Check error codes when calling the ToolBox (Hey, I sound like DTS!). Return a ge_Error event if you encounter a situation you cannot handle sensibly.
- Pascal is inherently safer than C.

6.7. How GameMaster Connects Games

If you follow the guidelines outlined above (and the interface defined in the previous chapters) GameMaster will organise to connect your game through the network automatically. How it does this is documented here for your interest only.

Imagine GameMaster running on two machines, one called Eriodon and the other Rocky. The user of Eriodon starts a game and connects to Rocky. The following happens:–

- The Eriodon GameMaster sends a ge_Flatten to its local rule book to make sure that the game handle is the most up to date available.
- The Eriodon GameMaster sends the contents of the game handle to the Rocky GameMaster.
- The Rocky GameMaster starts a game. If the rule book did not already have a running game it sends a ge_InitRuleBook to it. Regardless it sends a ge_OldGame to the rule book using the state supplied by the Eriodon GameMaster.
- The Rocky GameMaster then sends a ge_Swap its local rule book. This ensures that the two players are playing on different sides.
- Both GameMasters then send ge_ConnectionMade to the games.

From here on in the rule books are expected to keep their game records ‘in synch’ by exchanging messages.

∞

7. The GameMaster Distribution

7.1. The GameMaster User's Kit

The basic GameMaster is distributed as *GameMaster User's Kit* which contains:–

- *GameMaster* The main application.
- *The GameMaster User's Manual* A simple manual to help you use GameMaster.
- *Connect 4* Rhys' masterpiece.
- *Othello* Cute and fun.
- *BattleShips* A classic two player game (but only for networked machines).
- *Dots and Boxes* Mind bending.
- *Global Thermonuclear War* Realistic.
- *DerfBall* A devil of a game.
- *Simple Talk* Useful but not really a game.
- *File Transfer* A demonstration of just how much we expect others to abuse the concept of GameMaster.
- *Chess* A classic.
- *TicTacToe* Simple stuff.
- *Empire* If Jager ever finishes.

7.2. The GameMaster Developer's Kit

The *GameMaster Developer's Kit* contains the following files:–

- *Programming GameMaster Rule Books* This manual.
- *GameTypes.unit* The Pascal interfaces needed to write GameMaster rules.
- *GMRZTemplate.rsrc* A ResEdit template to add to your ResEdit Preferences file to help you edit GMRZ resources.
- The source code to all of the games distributed in the GameMaster User's Kit.
- A number of Pascal units useful for developing rule books.

∞