Tema 5: MODELOS DE VALIDACIÓN

Ingeniería de Protocolos y Servicios

5.1. Introducción

•Para especificar y verificar el conjunto de reglas de un protocolo se utiliza un modelo de validación.

•Un modelo de validación será una visión parcial del protocolo, una visión que se centrará en obtener un juego de reglas consistente y completo.

•Se pretende modelar un protocolo lo más simple posible para estudiar su estructura y verificar que sea consistente y completo.

DTIC
Área Ingeniería Telemática
francesc.burrull@upct.es

Tema 5: MODELOS DE VALIDACIÓN

Ingeniería de Protocolos y Servicios

5.1. Introducción

•Para describir un modelo de validación será necesario un lenguaje.

•Existen multitud de lenguajes para validación de protocolos. Entre ellos hay 3 que son estándar internacional:

- Estelle
- Lotos
- SDL

DTIC
Área Ingeniería Telemática
francesc.burrull@upct.es

Tema 5: MODELOS DE VALIDACIÓN

Ingeniería de Protocolos y Servicios

5.1. Introducción

SDL (*Specification and Description Language*)

Es un lenguaje que ofrece descripción y especificación sin ambigüedades sobre el comportamiento de sistemas de telecomunicaciones.

Dichas especificaciones son formales en el sentido que permiten análisis e interpretación sin ambigüedades.

- Especificación: descripción del comportamiento requerido del sistema.
- Descripción: descripción del comportamiento del sistema (comportamiento de la implementación).

DTIC
Área Ingeniería Telemática
francesc.burrull@upct.es

Tema 5: MODELOS DE VALIDACIÓN

Ingeniería de Protocolos y Servicios

5.1. Introducción

SDL (*Specification and Description Language*)

•Fácil de aprender, usar e interpretar

•Extensible a futuros desarrollos

•Soporta múltiples metodologías de especificación y diseño de sistemas.

•Puede usarse para test de descripciones de sistemas, en combinación con MSC (*Message Sequence Chart*) ITU-T Rec Z.120 y TTCN (*Tree and Tabular Combined Notation*)

DTIC
Área Ingeniería Telemática

francesc.burrull@upct.es

5.2. Definiciones (ITU-T Rec. Z.100 11/99)

• Agente: Denota un sistema, un bloque o un proceso que contiene una o más máquinas de estado finito.

• Bloque: Agente que contiene uno o más bloques concurrentes o procesos y quizás alguna máquina de estado finito que utilice datos internos al bloque.

• Canal: Vía de comunicación entre agentes.

5.2. Definiciones (ITU-T Rec. Z.100 11/99)

• Procedimiento: Encapsulado de parte del comportamiento de un agente. Será llamado desde dentro del agente.

• Proceso: Un agente que contiene una máquina de estado finito y puede contener otros procesos.

• Estado: Una máquina de estado finito de un agente se encuentra en un estado si está esperando un estímulo.

• Estímulo: Evento que causa que un agente en un estado entre en transición.

5.3 Tipos de Sintaxis (ITU-T Rec. Z.100 11/99)

- Representación Gráfica (SDL/GR -*graphic representation*-)
- Representación Textual (SDL/PR -*phrase representation*-)

Son equivalentes.

Mayoritariamente se usa la Representación Gráfica

5.4 Ejemplo: Demongame

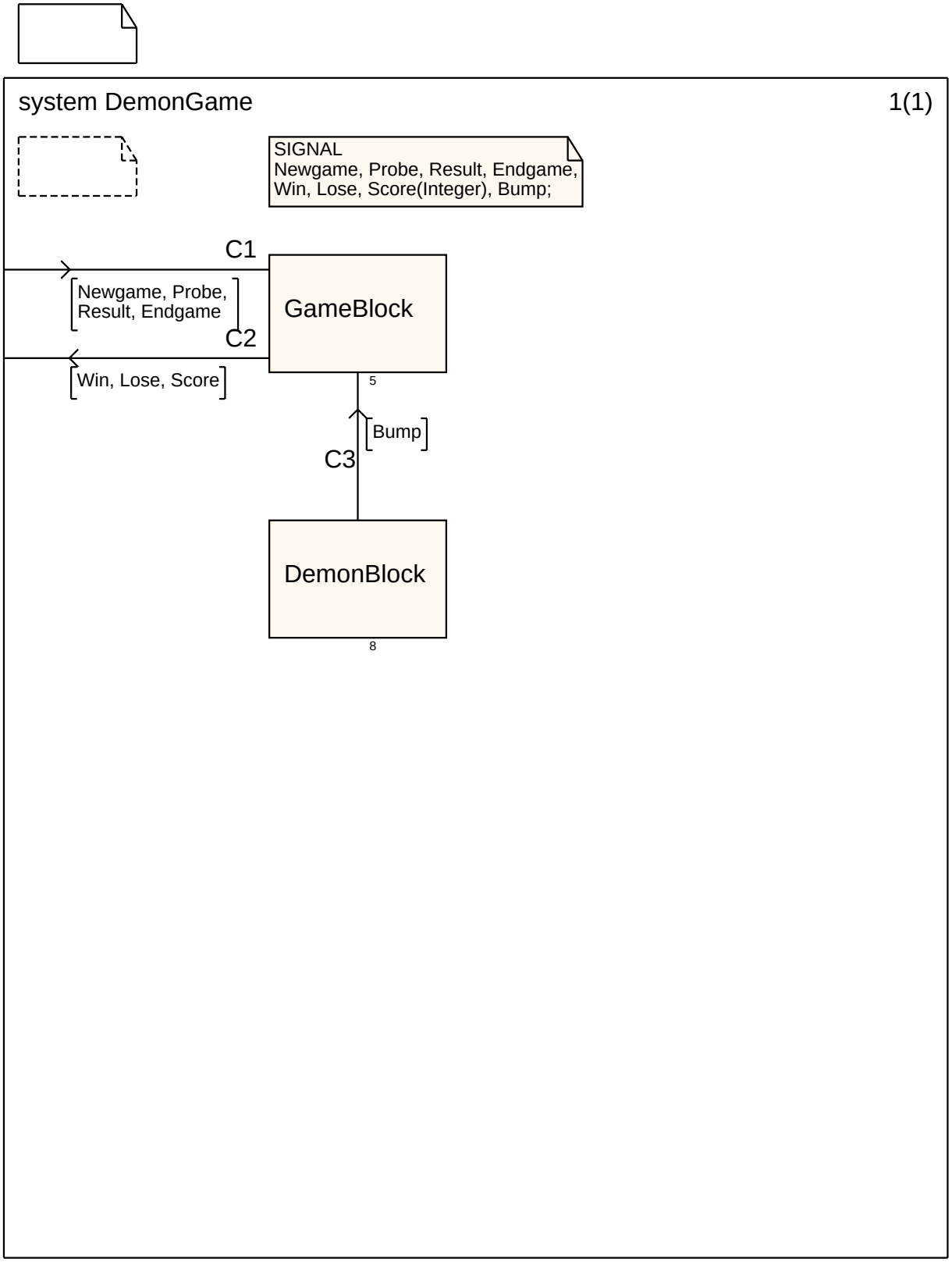
Este ejemplo se usa en los ejemplos de la recomendación Z100

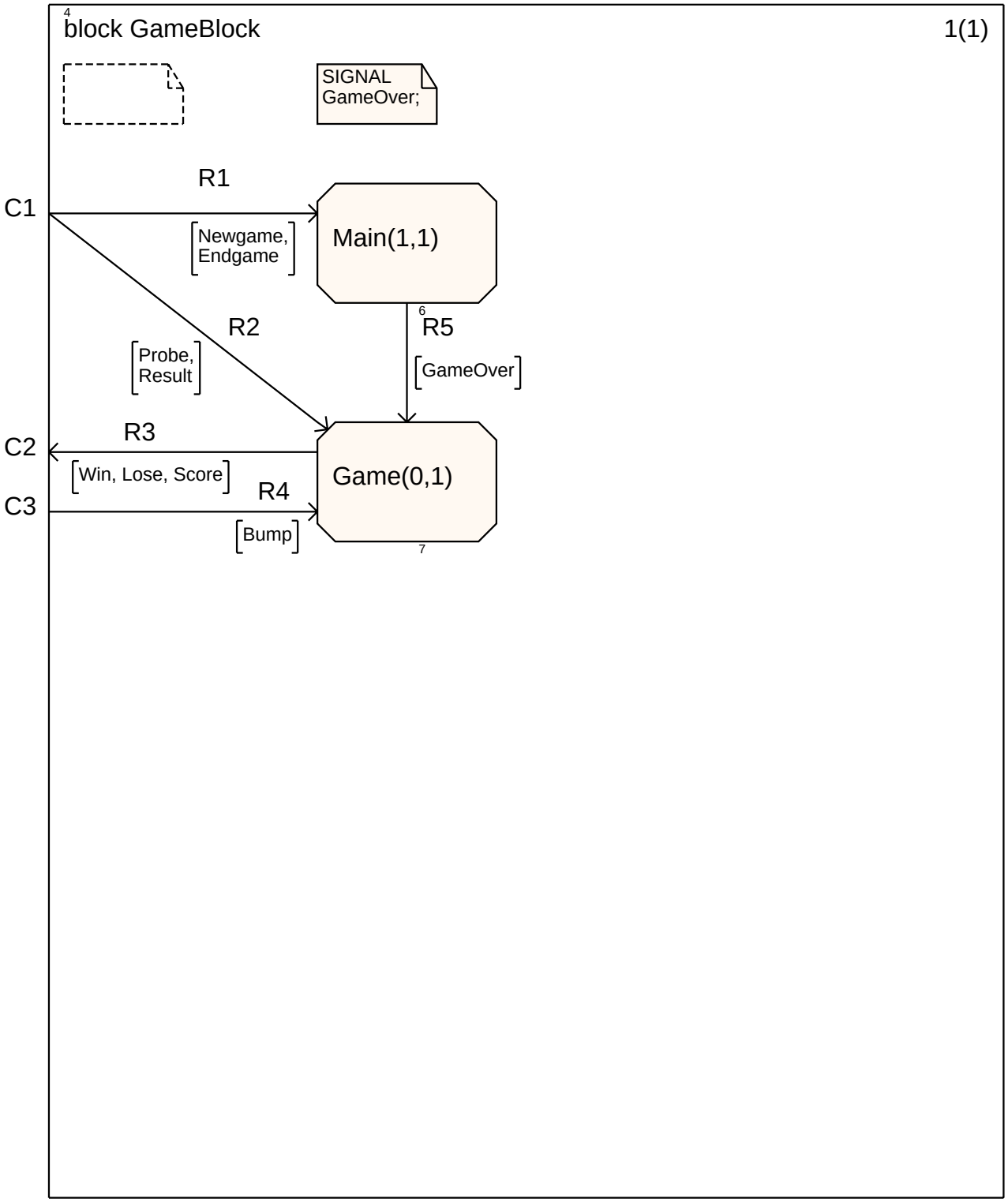
Consiste en un juego sencillo, donde el usuario puede generar las señales Newgame, Endgame, Probe y Result. Las 2 primeras señales son para iniciar (ignorada si ya se juega) y terminar (ignorada si no se juega) un juego.

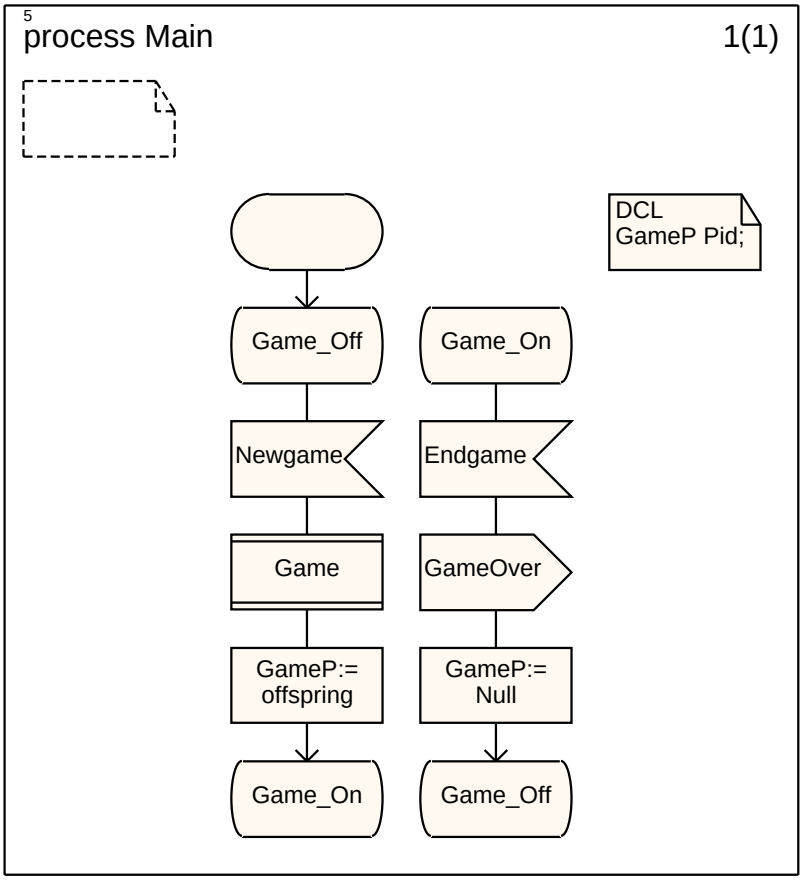
5.4 Ejemplo: Demongame

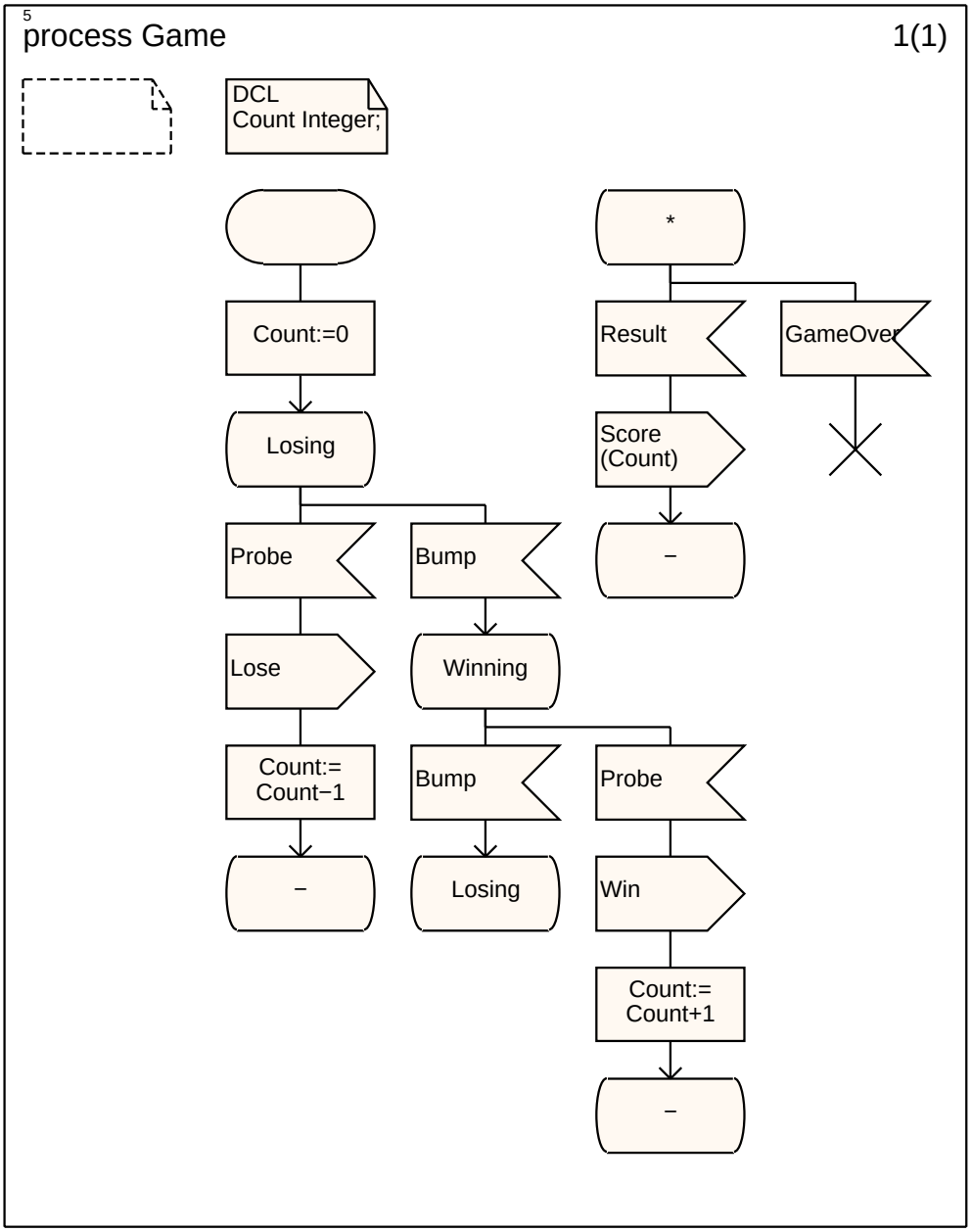
Las reglas del juego son sencillas: Un *demon*, representado por el proceso *Demon*, cambia el estado del sistema (*winning*, *losing*) de vez en cuando. El usuario debe adivinar cuando el estado es *winning*. Si el usuario prueba (señal *Probe*) cuando el estado es *winning*, gana un punto. Si el usuario prueba cuando el estado es *losing*, pierde un punto. Para ver el marcador el usuario produce la señal *Result*, contestada por la señal *Score*, que contiene un parámetro de tipo entero con el resultado actual.

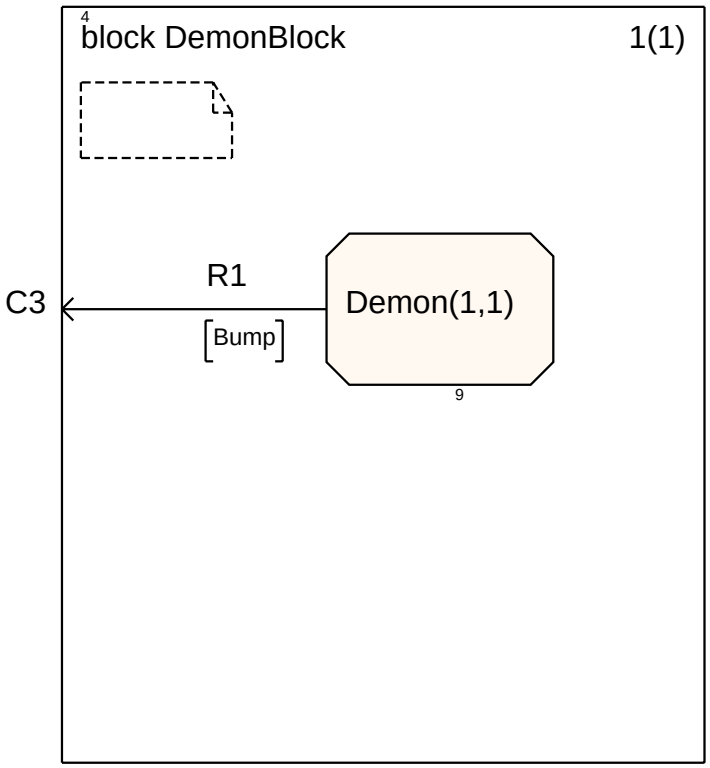
SDT	rw	C:\users\fburrull\IngenieriaProtocolos\SDL\demongame\demongame.sdt		
	rw	C:\users\fburrull\IngenieriaProtocolos\SDL\demongame\		
My first SDL system		3		
DemonGame		4	rw	DemonGame.ssy
DemonGame				
GameBlock		5	rw	GameBlock.sbk
Main		6	rw	Main.spr
Game		7	rw	Game.spr
DemonBlock		8	rw	DemonBlock.sbk
Demon		9	rw	Demon.spr
Other Documents		10		
DemonGame		11	rw	DemonGame.msc


system DemonGame
1(1)
SIGNAL
Newgame, Probe, Result, Endgame,
Win, Lose, Score(Integer), Bump;
C1
Newgame, Probe,
Result, Endgame
C2
Win, Lose, Score
GameBlock
5
Bump
C3
DemonBlock
8


4
block GameBlock
1(1)
SIGNAL
GameOver;
R1
C1
Newgame,
Endgame
Main(1,1)
6
R5
R2
Probe,
Result
GameOver
R3
C2
Win, Lose, Score
Game(0,1)
R4
C3
Bump
7

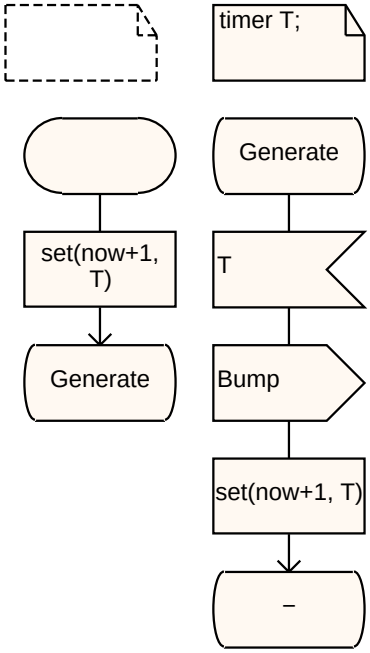

5
process Main
1(1)
DCL
GameP Pid;
Game_Off
Newgame
Game
GameP:=
offspring
Game_On
Game_On
Endgame
GameOver
GameP:=
Null
Game_Off

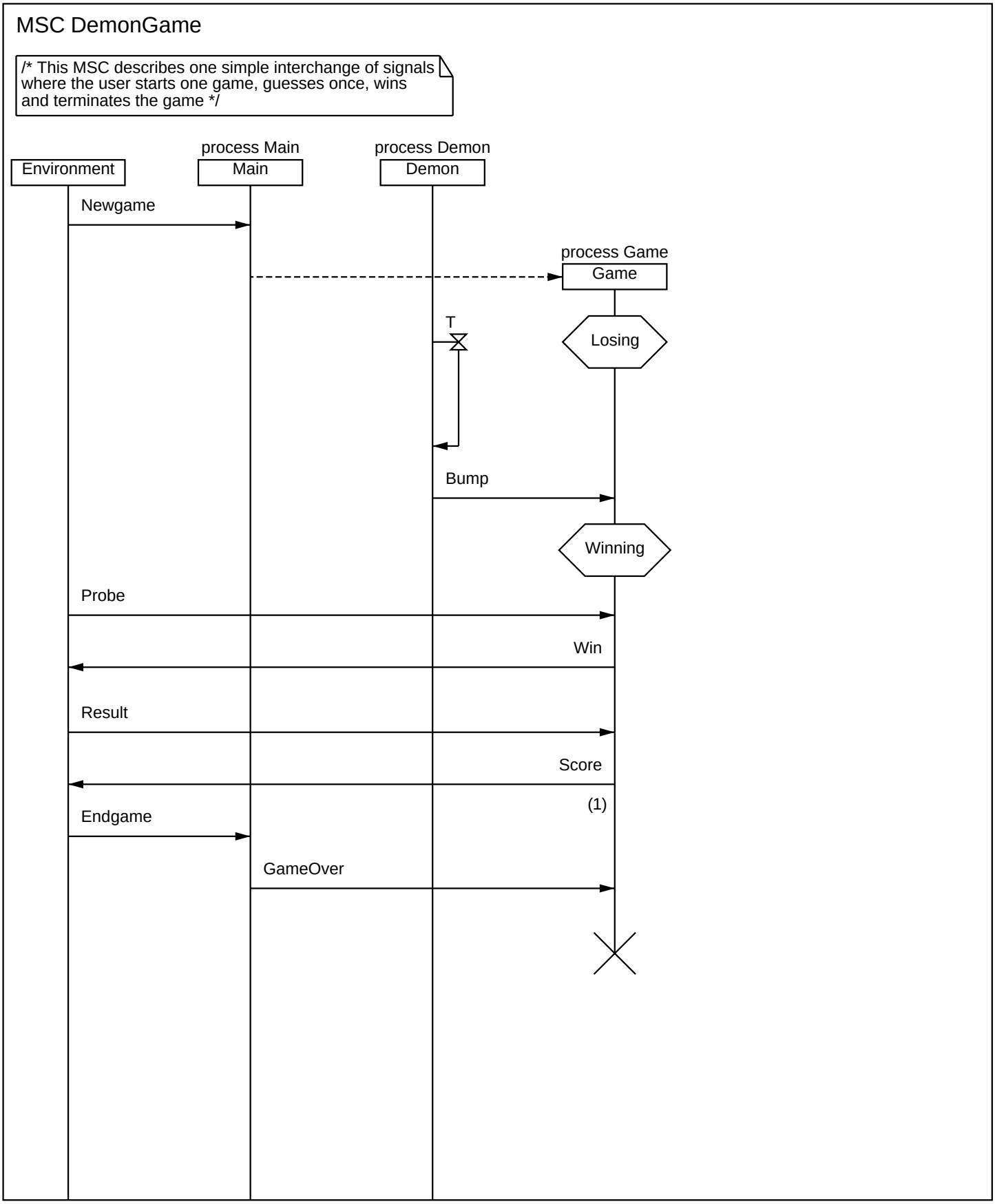

5
process Game
1(1)
DCL
Count Integer;
Count:=0
Losing
Probe
Bump
Lose
Winning
Count:=
Count−1
Bump
Probe
−
Losing
Win
Count:=
Count+1
−
*
Result
GameOver
Score
(Count)
−


4
block DemonBlock
1(1)
R1
C3
Demon(1,1)
Bump
9

8

process Demon




MSC DemonGame
/* This MSC describes one simple interchange of signals where the user starts one game, guesses once, wins and terminates the game */
Environment
process Main
Main
process Demon
Demon
Newgame
process Game
Game
T
Losing
Bump
Winning
Probe
Win
Result
Score
(1)
Endgame
GameOver