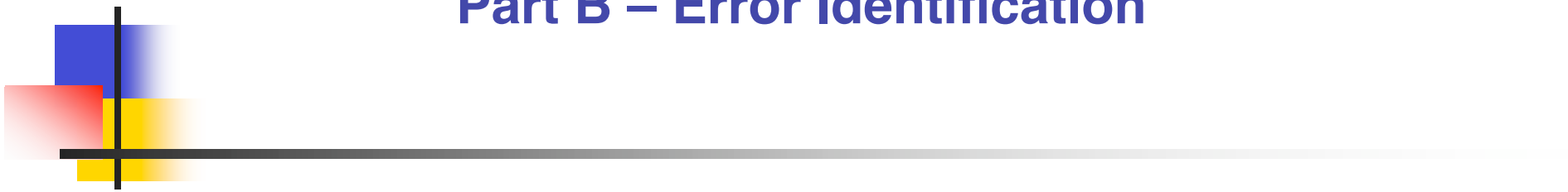
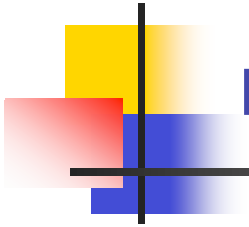


State-Based Testing
Part B – Error Identification

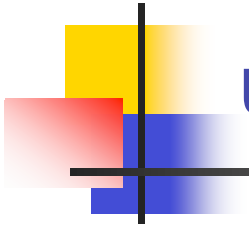
Generating test cases for complex behaviour

Reference: Robert V. Binder
Testing Object-Oriented Systems: Models, Patterns, and Tools
Addison-Wesley, 2000, Chapter 7



Expanding the statechart

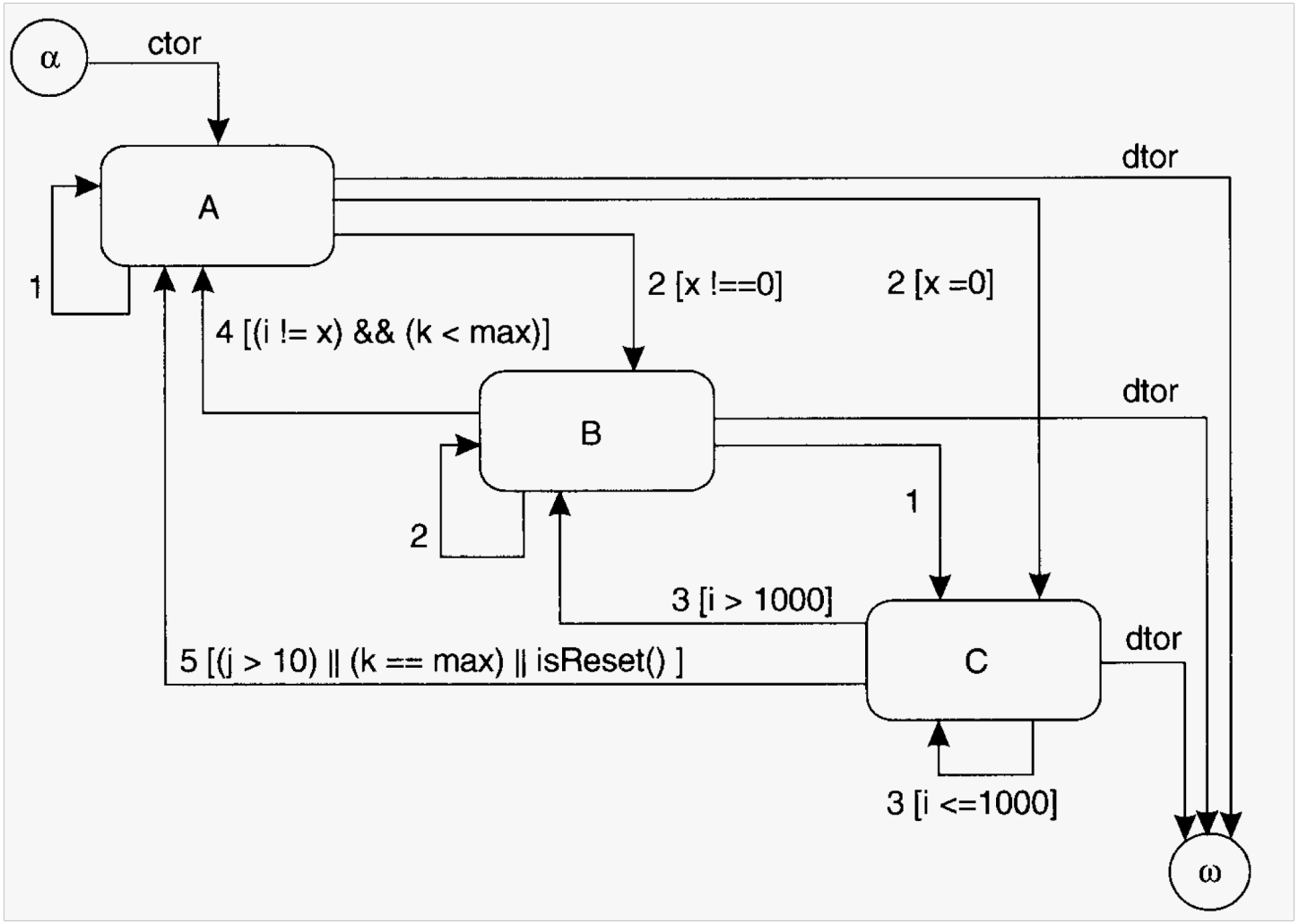
- Statecharts are great for communication, reducing clutter etc.
- They might hide subtle bugs
  - **e.g. entering a sub-state rather than a super-state**
- We need to expand them to full transition diagrams for testing purposes
  - **Expansion makes implicit transitions explicit, so they are not lost**
  - **Expansion is a flat view**
    - **Includes everything from inheritance in OO and sub-states in statecharts**
- An automatable process



Unspecified Event/State Pairs

- State machine models will not include all events for all states
- Implicit transitions may be illegal, ignored, or a specification omission
- Accepted illegal events lead to bugs called **sneak paths**
- For testing purposes, we cannot ignore implicit behaviour
  - **Develop a Response Matrix**

Example statechart

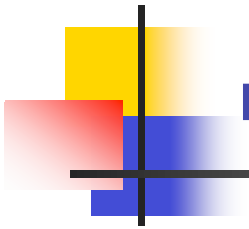


Response matrix

Events and Guards				Accepting State/Expected Response					
				α	A	B	. . .	C	
ctor				✓	6	6	6	6	6
Event 1				☒	✓	✓	. . .	2	6
Event 2	x == 0								
	DC			☒	☒	✓	. . .	2	6
	F			☒	✓	☒	. . .	☒	☒
	T			☒	✓	☒	. . .	☒	☒
Event 3	i <= 1000								
	DC			☒	2	2	. . .	☒	6
	Off			☒	2	2	. . .	✓	☒
	On			☒	2	2	. . .	✓	☒
Event 4	i != x	k < max							
	DC	DC		☒	2	☒	. . .	2	6
	F	F		☒	☒	1	. . .	☒	☒
	F	T		☒	☒	2	. . .	☒	☒
	T	F		☒	☒	2	. . .	☒	☒
	T	T		☒	☒	✓	. . .	☒	☒
Event 5	i > 10	k == max	isReset()						
	DC	DC	DC	☒	2	5	. . .	☒	6
	F	F	F	☒	☒	☒	. . .	5	☒
	F	F	T	☒	☒	☒	. . .	✓	☒
	F	T	F	☒	☒	☒	. . .	✓	☒
	F	T	T	☒	☒	☒	. . .	✓	☒
	T	F	F	☒	☒	☒	. . .	✓	☒
	T	F	T	☒	☒	☒	. . .	✓	☒
	T	T	F	☒	☒	☒	. . .	✓	☒
	T	T	T	☒	☒	☒	. . .	✓	☒
dtor				☒	✓	✓	. . .	✓	2

Key: T = true, F = false, DC = don't care; for Action codes, see Table 7.3

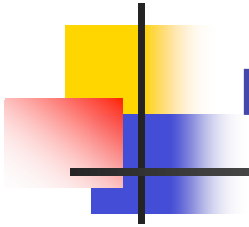
☐ Not applicable ☒ Excluded ☑ Explicitly specified transition



Possible responses to illegal events

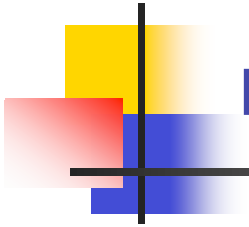
TABLE 7.3 Response Codes for Illegal Events

Response Code	Name	Response
0	Accept	Perform the explicitly specified transition
1	Queue	Place the illegal event in a queue for subsequent evaluation and ignore
2	Ignore	No action or state change is to be produced, no error is returned, no exception raised
3	Flag	Return a nonzero error code
4	Reject	Raise an `IllegalEventException`
5	Mute	Disable the source of the event and ignore
6	Abend	Invoke abnormal termination services (e.g., core dump) and halt the process



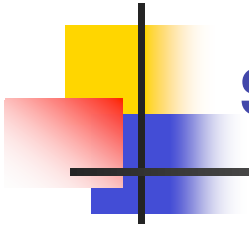
Designing responses to illegal events

- Abstract state should not change
  - **Concrete state may change due to exception handling**
- Illegal event design question
  - **Handle with defensive programming**
    - **Defensive systems**
  - **Handle with precondition contracts**
    - **Cooperative systems**



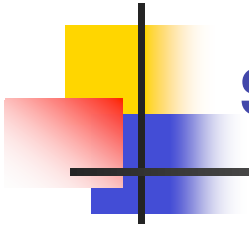
Designing responses to illegal events – 2

- Possible responses
  - **Raise exception**
  - **Treat message as a noop**
  - **Attempt error recovery**
  - **Invoke abnormal termination**
- Tester needs to decide expected responses so actual responses can be evaluated



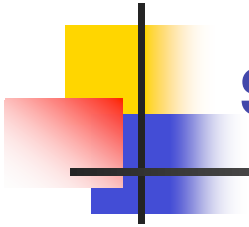
State model validation

- A state model must be complete, consistent, and correct before it is used to generate test cases
- We will look at four validation checklists
  - **Structure checklist**
  - **State name checklist**
  - **Guarded transition checklist**
  - **Robustness checklist**



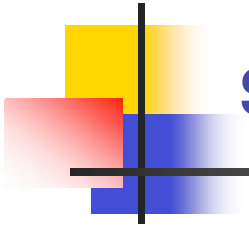
Structure checklist

- There is an initial state with only outbound transitions
- There is a final state with only inbound transitions (if not, explicit reason is needed)
- No equivalent states
- Every state is reachable from the initial state
- The final state is reachable from all states
- Every defined event and every defined action appears in at least one transition



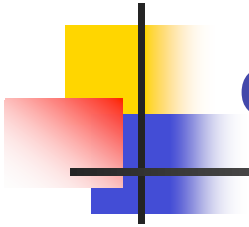
Structure checklist

- Except for the initial and final states, every state has at least one incoming and one outgoing transition
- The events accepted in a particular state are unique or differentiated by mutually exclusive guards
- Complete specification: For every state, every event is accepted or rejected (either explicitly or implicitly)



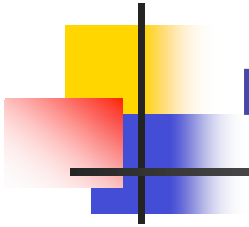
State name checklist

- Poor names are often indications of incomplete or incorrect design
- Names must be meaningful in the context of the application
- If a state is not necessary, leave it out
  - **“Wait states” are often superfluous**
- State names should be passive
- Adjectives are best, past participles are OK



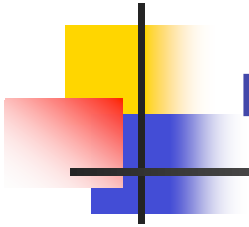
Guarded transition checklist

- The entire range of truth values for a particular event is covered
- Each guard is mutually exclusive of all other guards
- Guard variables are visible
- Guards with three or more variables are modeled with a decision table
- The evaluation of a guard does not cause side effects



Robustness checklist

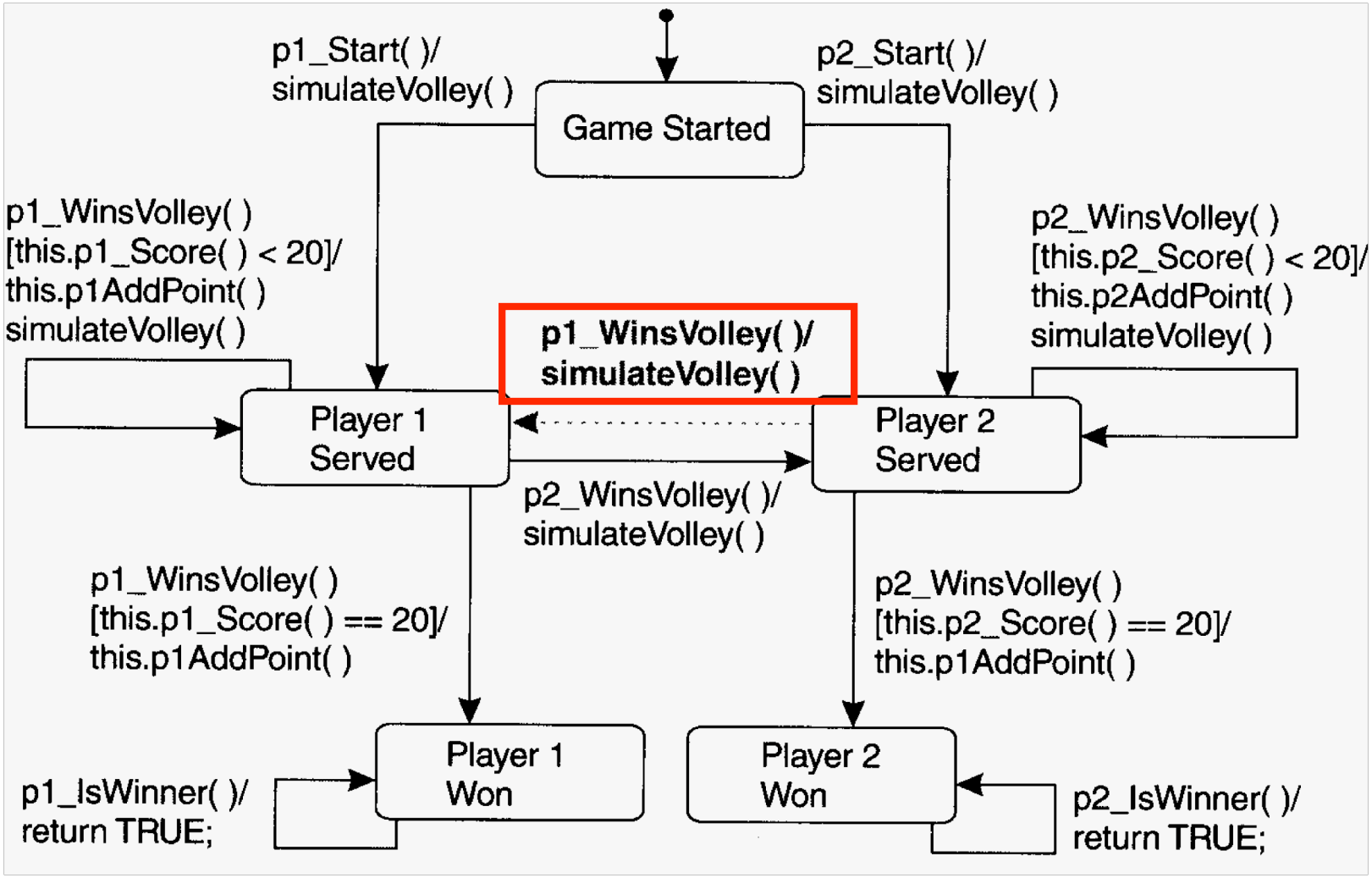
- There is an explicit spec for an error-handling or exception-handling mechanism for implicitly rejected events
- Illegal events do not corrupt the machine (preserve the last good state, reset to a valid state, or self-destruct safely)
- Actions have no side effects on the resultant state
- Explicit exception, error logging, and recovery mechanisms are specified for contract violations



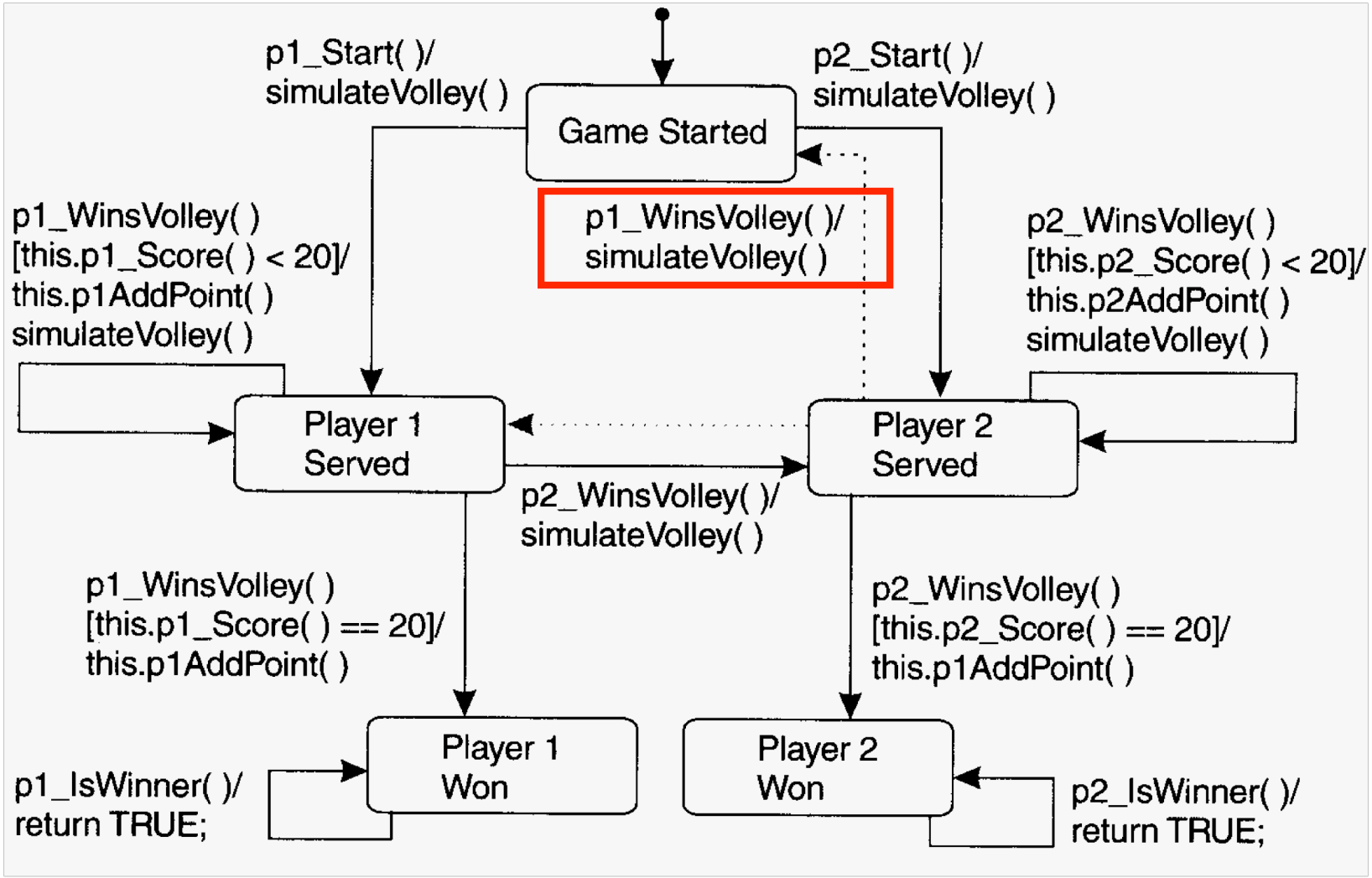
Fault model for state machines

- Control faults: An incorrect sequence of events is accepted, or an incorrect sequence of outputs is produced
  - **Missing transition**
  - **Incorrect transition**
  - **Missing action**
  - **Incorrect action**
  - **Sneak path**
  - **Corrupt state**
  - **Illegal message failure**
  - **Trap door – undefined message/events**

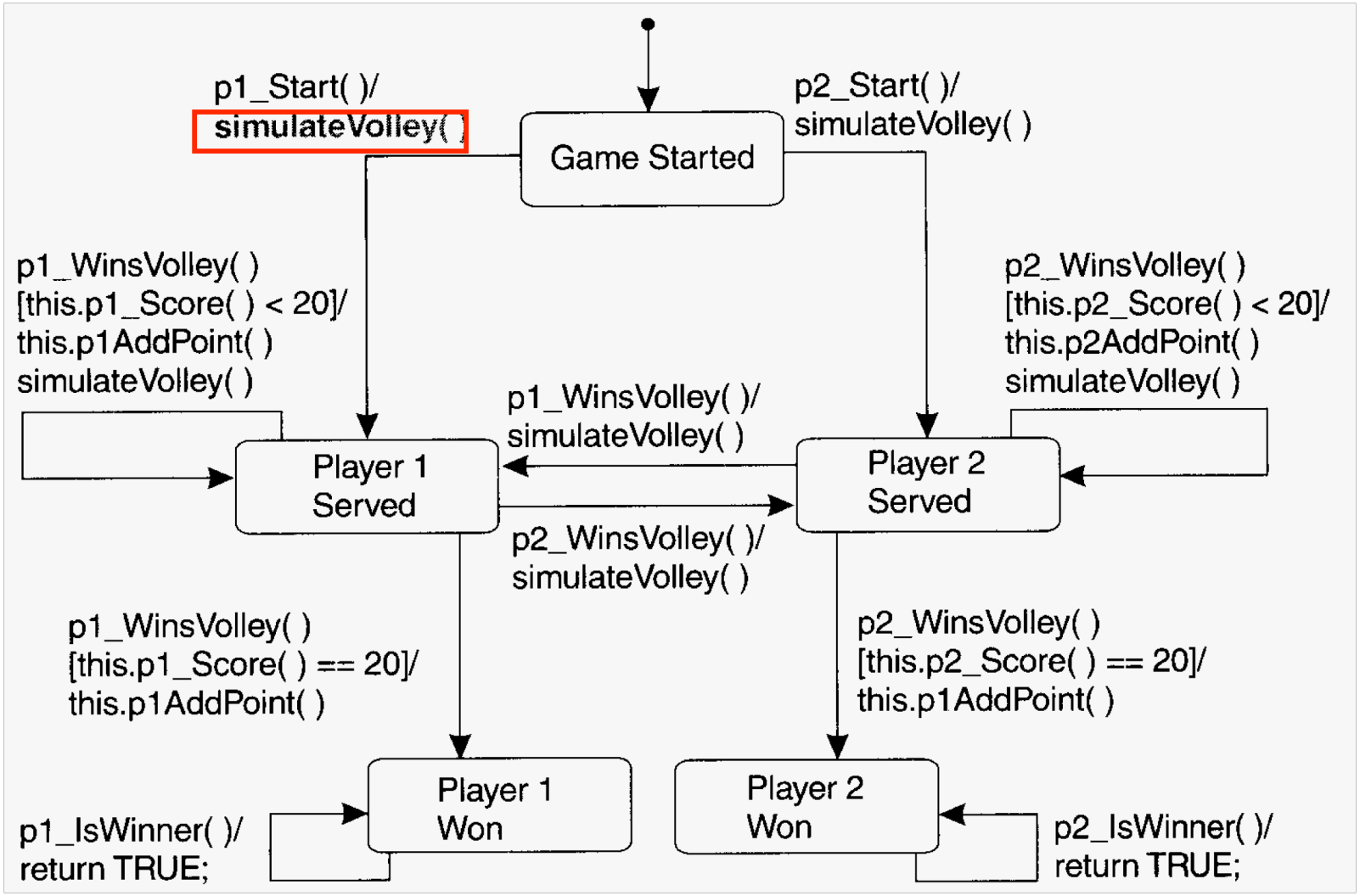
Missing transition



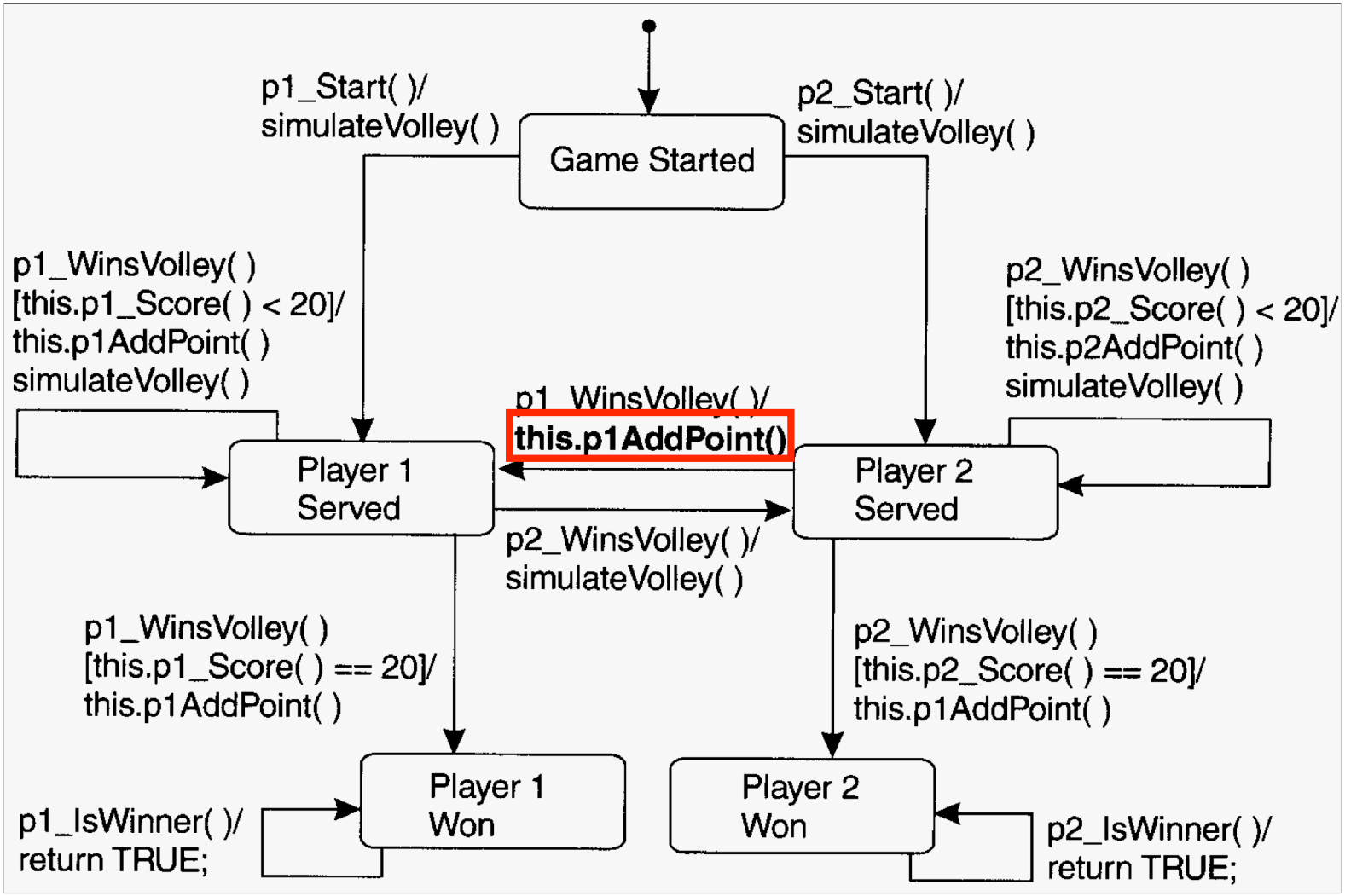
Incorrect transition



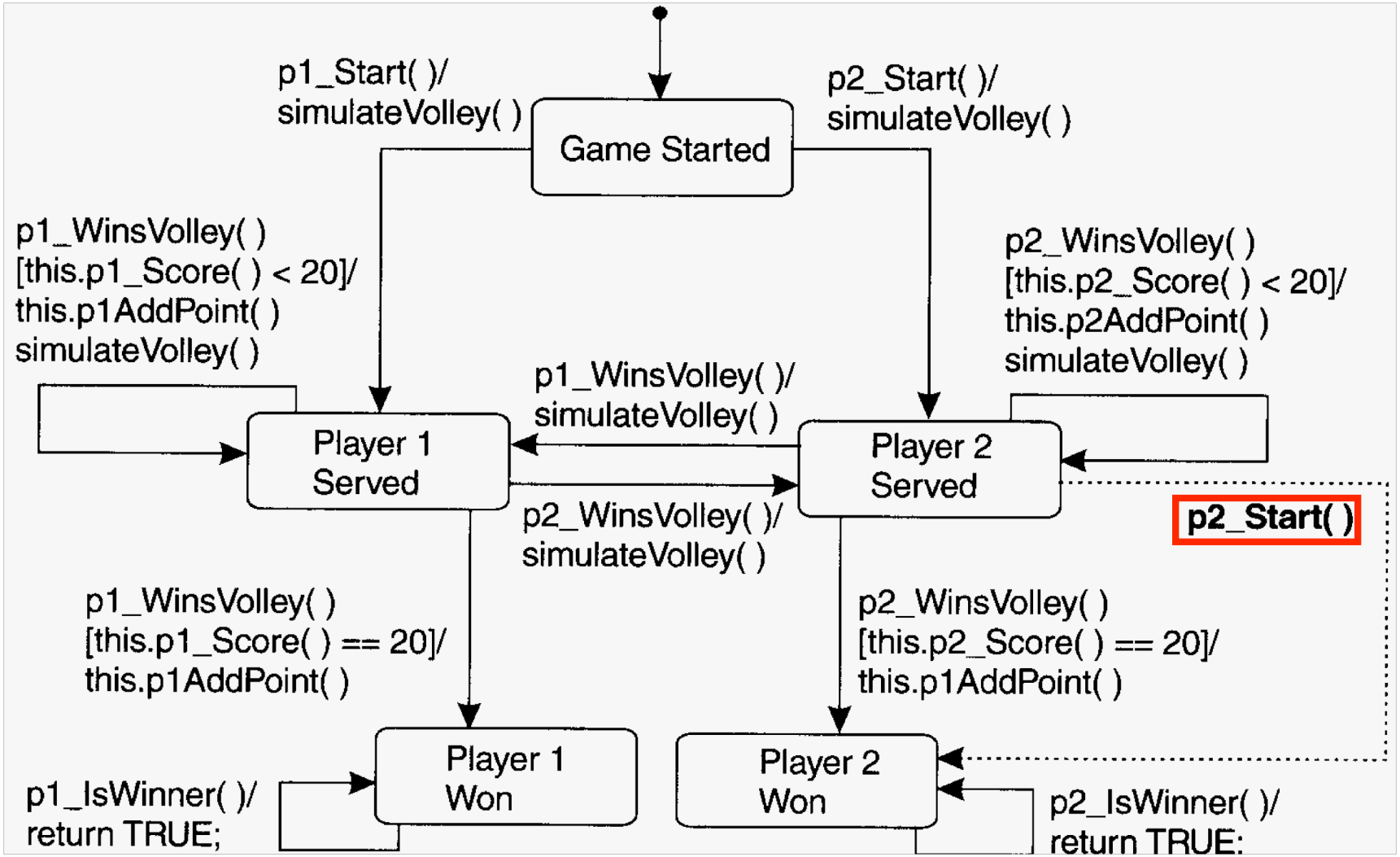
Missing action



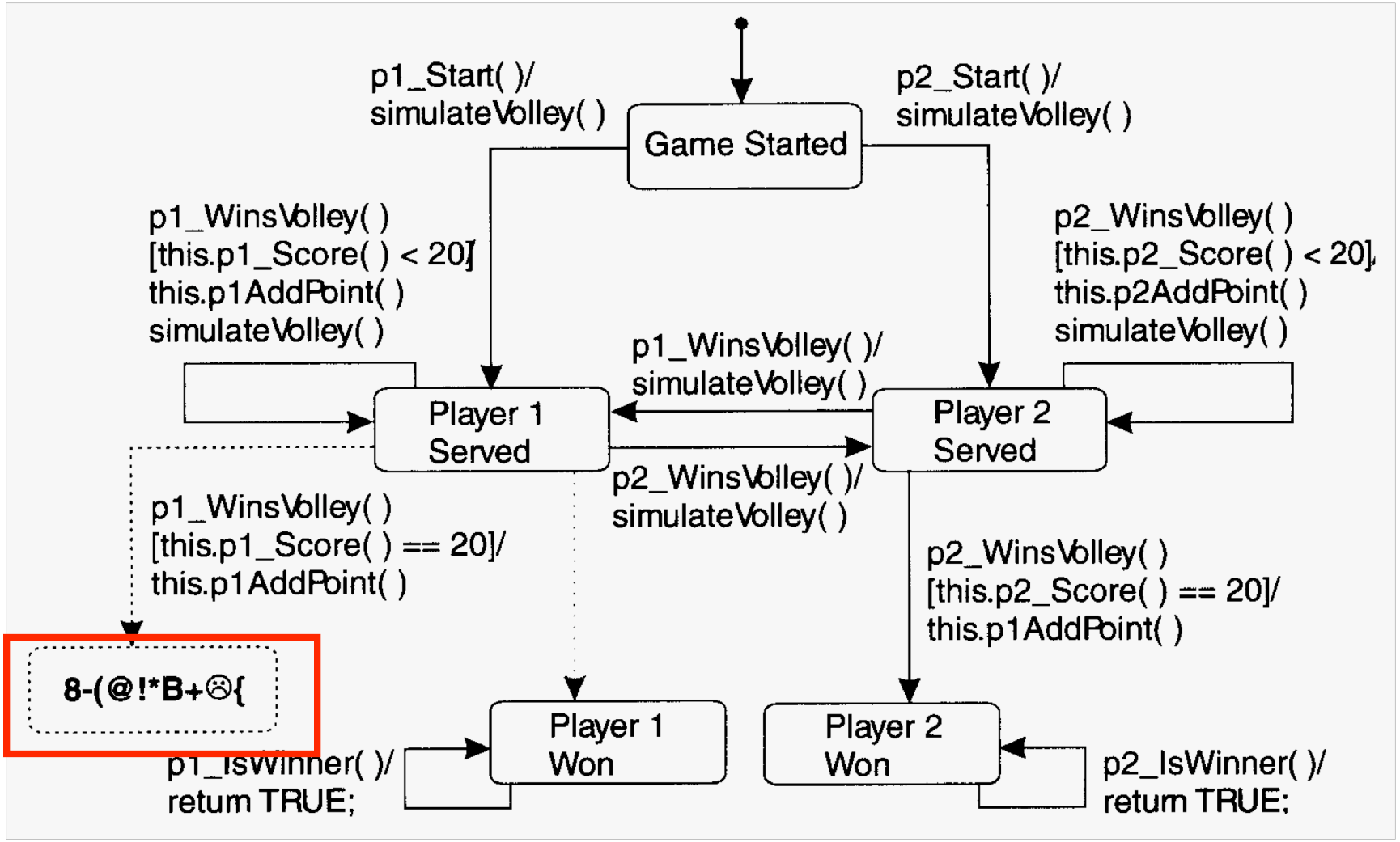
Incorrect action



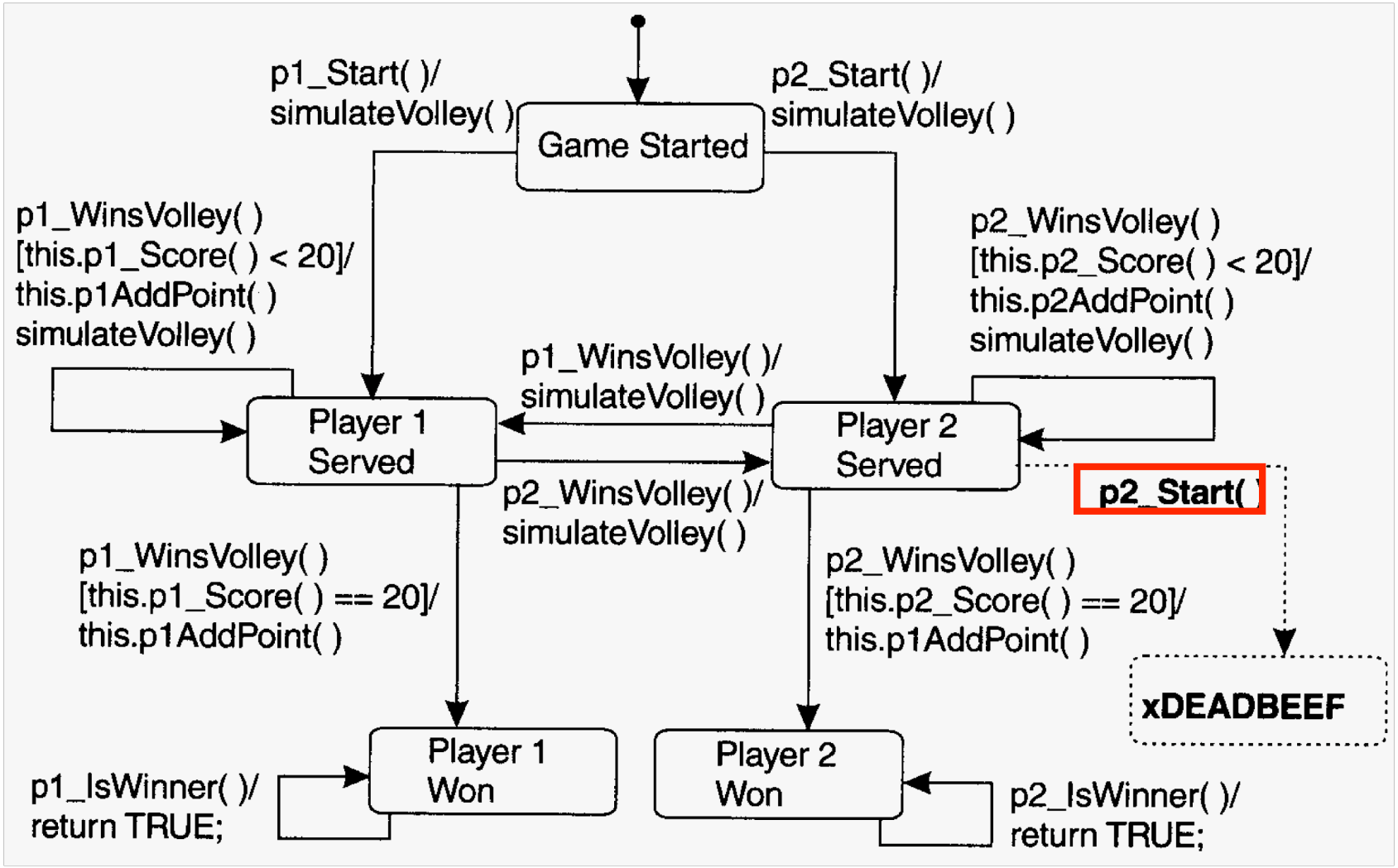
Sneak path


p1_Start( )/
simulateVolley( )
Game Started
p2_Start( )/
simulateVolley( )
p1_WinsVolley( )
[this.p1_Score( ) < 20]/
this.p1AddPoint( )
simulateVolley( )
p2_WinsVolley( )
[this.p2_Score( ) < 20]/
this.p2AddPoint( )
simulateVolley( )
p1_WinsVolley( )/
simulateVolley( )
Player 1
Served
Player 2
Served
p2_WinsVolley( )/
simulateVolley( )
p2_Start( )
p1_WinsVolley( )
[this.p1_Score( ) == 20]/
this.p1AddPoint( )
p2_WinsVolley( )
[this.p2_Score( ) == 20]/
this.p1AddPoint( )
Player 1
Won
Player 2
Won
p1_IsWinner( )/
return TRUE;
p2_IsWinner( )/
return TRUE;

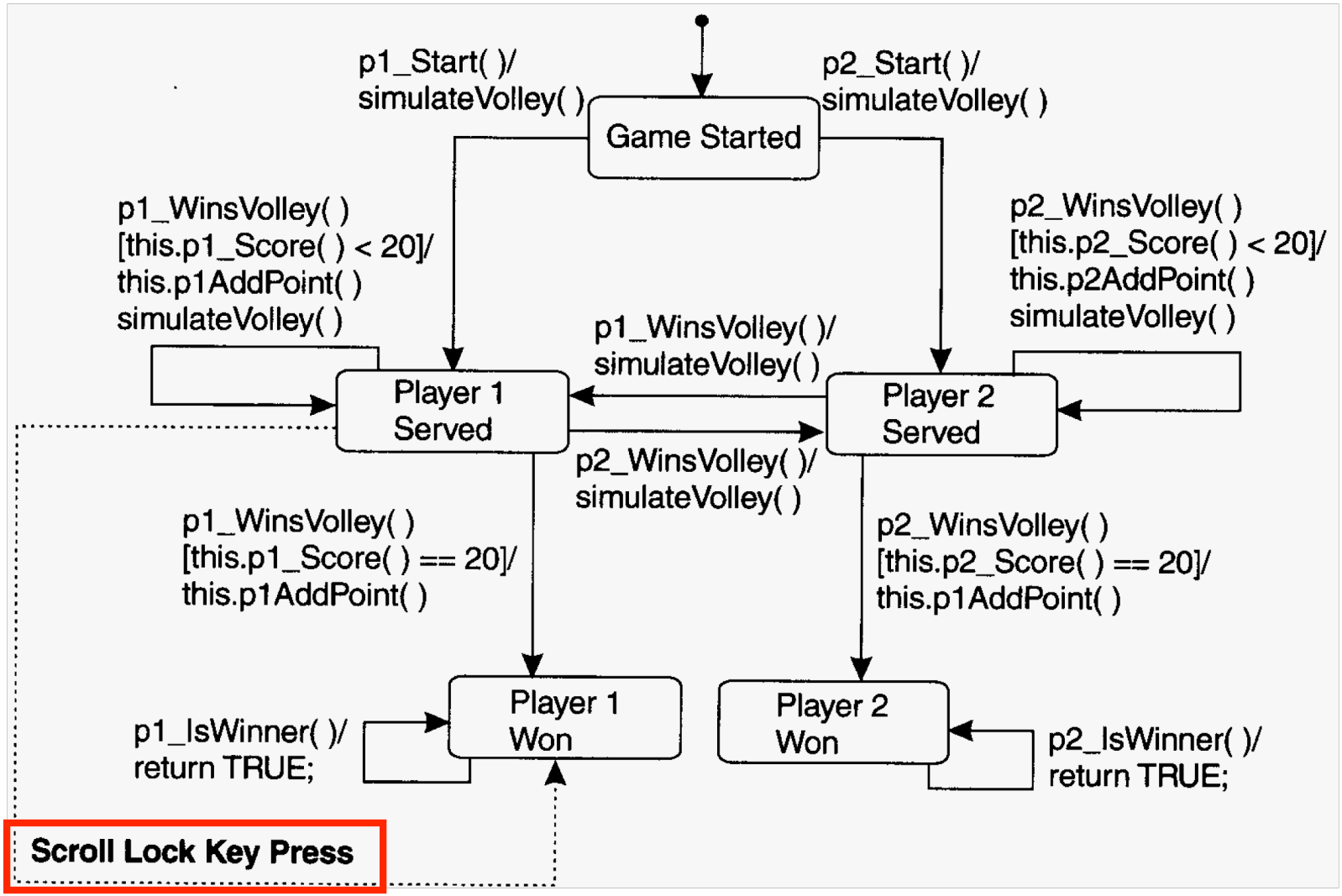
Corrupt state



Sneak path to corrupt state



Trap door


p1_Start( )/
simulateVolley( )
p2_Start( )/
simulateVolley( )
Game Started
p1_WinsVolley( )
[this.p1_Score( ) < 20]/
this.p1AddPoint( )
simulateVolley( )
p2_WinsVolley( )
[this.p2_Score( ) < 20]/
this.p2AddPoint( )
simulateVolley( )
p1_WinsVolley( )/
simulateVolley( )
Player 1
Served
Player 2
Served
p2_WinsVolley( )/
simulateVolley( )
p1_WinsVolley( )
[this.p1_Score( ) == 20]/
this.p1AddPoint( )
p2_WinsVolley( )
[this.p2_Score( ) == 20]/
this.p1AddPoint( )
Player 1
Won
Player 2
Won
p1_IsWinner( )/
return TRUE;
p2_IsWinner( )/
return TRUE;
Scroll Lock Key Press