Санкт-Петербург Россия

Образовательный центр Motorola в СПбГПУ

Описание архитектуры проекта «BattleShips»

Версия D01.00.00.

Документ подготовлен:			
Имя:	А. Егоров Л. Дворянский И. Кривченко А. Савинов	**Адрес электронной почты:**	andreiyegorov@pisem.net dvleonid@land.ru marsiana_k@mail.ru AndreySavinov@mail.ru
Дата:		15 July 2003	
Документ распечатан из файла:		C:\dd_d01_00_00.doc	

История разработки.

Версия	Описание	Дата	Автор
D00.01.00	Начальная версия документа.	09 июля 2003 г.	А. Егоров
D00.01.01	Добавлено описание реализации коммуникационного модуля DataSender.	09 июля 2003 г.	Л. Дворянский
D00.01.02	Добавлено описание реализации обработки приказов, поступающих от стратегий, и обработки перемещения кораблей и выстрелов.	09 июля 2003 г.	А. Савинов
D00.02.00	Добавлено описание реализации стратегии.	10 июля 2003 г.	И. Кривченко
D00.02.01	Добавлено описание реализации коммуникационного модуля Router.	10 июля 2003 г.	А. Егоров
D01.00.00	Первая версия (черновая) документа.	13 июля 2003 г.	А. Егоров И. Кривченко

Содержание.

Список рисунков.

1. Введение.

1.1 Цель.

Данный документ описывает реализацию программного проекта «BattleShips». Этот документ предназначен для всех пользователей, которые хотят детально ознакомиться с реализацией отдельных модулей или всего проекта в целом.

Документ содержит описание реализации всех уровней проекта: системы в целом, блоков симулятора и стратегий, модулей симулятора и интерфейса между ними.

1.2 Сокращения.

GUI..... Graphical User Interface

MSC ..... Message Sequence Chart (ITU-T z.120)

SDL ..... Specification and Description Language (ITU-T z.100)

1.3 Соглашения.

- «Стратегия» - модуль проекта, управляющий поведением ему принадлежащих кораблей.
- «Симулятор» - модуль, отвечающий за прием и отправление сигналов стратегиям, синхронизацию информации о кораблях каждого флота, управление синхронизацией процесса отображения текущего положения кораблей и выстрелов на графическом интерфейсе.
- «Клиент» - рабочая станция или терминал, на котором запускается GUI.
- «Сервер» - рабочая станция или терминал, на котором запускается непосредственно симулятор со стратегиями.
- Термины «сообщения» и «сигнал» обозначают одно и то же.
- «Флот» - группа кораблей, принадлежащих одной стратегии.
- «Мель» - глубина, меньшая посадки корабля, но не суша.
- «Так проверки» - такт симуляции, на котором проверяется возможность окончания процесса симуляции.
- «Поле видимости» - участок карты, который попадает в зону обзора, по крайней мере, одного корабля данного флота.
- «Коллизия» - возможность столкновения кораблей на следующем такте симуляции или посадка корабля на мель через 2 и менее тактов симуляции.

1.4 Ссылки.

[1] Функциональная спецификация проекта «BattleShips».

[2] Высокоуровневое описание проекта «BattleShips».

2. Описание реализации проекта в целом.

Проект BattleShips можно разделить на три основные части: блок симулятора (Simulator), блоки стратегий (Strategy1 и Strategy2) и графический интерфейс (GUI). Блоки симулятора и стратегий реализованы на языке спецификаций SDL (см. Рисунок 17), а графический интерфейс разработан на объектно-ориентированном языке программирования высокого уровня JAVA.

Согласно п. 2.5.4 документа [1] типы данных, константы и переменные, описанные в приложении B, реализованы в виде отдельного модуля (см. Рисунок 1 - Рисунок 6, Рисунок 9 - Рисунок 16). На рисунках (см. Рисунок 1 - Рисунок 6) представлено описание типов данных, констант и переменных, доступных на уровне системы, т.е. и симулятору и стратегиям. На рисунках (см. Рисунок 9 - Рисунок 16) можно увидеть, как на языке спецификаций SDL сопоставляется имя переменной и числовое значение. Реализация требований, описанных в п. 2.5.3 и п. 2.5.5, можно увидеть на следующих рисунках соответственно: Рисунок 7 и Рисунок 8 - это 2 конфигурационных файла, представленных в текстовой форме. Но есть одно довольно серьезное различие при их использовании: файл «ConfigFile.pr» используется при компиляции проекта, поэтому имена переменных и их значения доступны при разработке проекта. Включение этого файла в проект показано на следующем рисунке: см. Рисунок 1. В то же время данные файла «DataValue.pr» доступны только при исполнении проекта, т.е. при его запуске, поэтому явно нигде не указано вхождение этого файла в проект.

Взаимодействие между блоком симулятора и блоками стратегий происходит с помощью списков сигналов, описанных в п. B.8 документа [1], по каналам SimSt1Int и SimSt2Int соответственно с первой и второй стратегиями (см. Рисунок 17). Кроме того, эти блоки имеют возможность отправлять сообщения messageLog и messageError (см. п. B.7 документа [1]) в «окружающую среду» по каналам SimMsgInt, St1MsgInt и St2MsgInt соответственно для симулятора, стратегии 1 и стратегии 2 – это необходимо для возможности ведения журнала событий и вывода любой отладочной информации.

Взаимодействие с графическим интерфейсом блок симулятора осуществляет с помощью передачи информации по сокетам (см. п. 2.3.1.2 документа [2]). Реализация представлена на следующих рисунках: см. Рисунок 103 - Рисунок 108, см. Код процесса DataSender (блок симулятора). Более подробное описание реализации будет приведено ниже.

2.1 Реализация на уровне блоков.

2.1.1 Блок симулятора (Simulator).

- Это блок, содержащий реализацию симулятора проекта BattleShips и отвечающий за симуляцию морского боя.
- Типы данных, константы и переменные:
  - Перечислимые типы:

Название типа	Значения	Описание
tErrorCode	EC_OK EC_FAULT	Результат выполнения процедуры или процесса.

  - Структуры данных:

Название типа	Поле	Тип поля	Описание
tShotParameters			Параметры выстрела:
	TypeWeaponry	tNameWeaponry	тип оружия
	CurrentCoordinates	tCoordinates	текущие координаты
	FinishCoordinates	tCoordinates	координаты цели
tShipParameters			Параметры корабля:
	ShipId	tShipId	идентификатор корабля
	StrategyId	tStrategyId	идентификатор стратегии
	TypeShip	tNameShip	тип
	Speed	tSCMaxSpeed	скорость
	Heading	tHeading	направление
	Resource	tSCMaxResource	ресурс
	Coordinates	tCoordinates	координаты

	Ammunition	tArrayAmountAmmunition	количество боеприпасов
	VisibleToStrategy	tArrayVisible	виден ли корабль для стратегий

- Хэш-таблицы:

Название типа	Тип индекса	Тип значения	Описание
tArrayVisible	tStrategyId	Boolean	Для каждой стратегии выставляется флаг видимости.
tArraySI	tStrategyId	Boolean	Для каждой стратегии выставляется флаг ее присутствия.

- Списки:

Название типа	Тип компонент	Описание
tStringShipParameters	tShipParameters	Список кораблей с их характеристиками для каждого флота.
tStringShotParameters	tShotParameters	Список выпущенных снарядов, но не достигших конечной цели.

- Сигналы:

Имя сигнала	Типы полей	Описание
SendData		Начало передачи данных GUI.
InitSocket		Инициализация соединения с GUI.
SendDataComplete		Окончание передачи данных GUI:
	tErrorCode	результат выполнения операций.
InitSocketComplete		Окончание инициализации соединения с GUI:
	tErrorCode	результат выполнения операций.

- Переменные:

Название переменной	Тип	Описание
gShips	tStringShipParameters	Массив кораблей, доступных стратегиям.
gShots	tStringShotParameters	Массив выстрелов.
gASI	tArraySI	Массив стратегий, ведущих морской бой.

Блок симулятора состоит из трех модулей (см. Рисунок 18 - Рисунок 20):

- главный модуль – процесс Main (см. Код процесса Main (блок симулятора).); этот модуль отвечает за хранение обработку данных обо всех флотах и выстрелах, прием сигналов от стратегий и отправление им ответов, управление процессом симуляции морского боя.
- коммуникационный модуль – процесс DataSender (см. Код процесса DataSender (блок симулятора).); этот модуль отвечает за процесс взаимодействия с графическим интерфейсом: установление соединения, отправление данных о карте, кораблях и выстрелах.
- коммуникационный модуль – процесс Router (см. Код процесса Router (блок симулятора).); этот модуль отвечает за прием сигналов от симулятора и стратегий и перенаправление их стратегиям и симулятору соответственно.

Из рисунка (см. Рисунок 20) видно, что сигналы, поступающие от стратегий из каналов SimSt1Int и SimSt2Int, разводятся соответственно по путям chRtrSt1 и chRtrSt2 в процесс Router. Процесс Main посылает и принимает сигналы процессам Router и DataSender соответственно по путям chMnRtr и chMnDS. Кроме того, процесс Main может посылать сигналы messageLog и messageError по пути chSimMsg в канал SimMsgInt. Более подробное описание функционирования этих модулей можно найти ниже.

2.1.2 Блок стратегий (Strategy).

Каждый из блоков стратегий состоит только из одного процесса (см. Рисунок 21), отвечающего за поведение стратегии во время симуляции морского боя, и его структура приведена ниже (рассмотрим на примере блока Strategy1 – для блока Strategy2 архитектура точно такая же, разница лишь в индексе: вместо «1» везде стоит индекс «2»):

Процесс Main1 получает сигналы от симулятора по пути chStSim через канал SimSt1Int и посылает сигналы messageLog и messageError в «окружающую среду» по пути chStMsg в канал St1MsgInt. Более подробное описание функционирования этого модуля можно найти ниже.

2.2 Реализация модуля Main (блок симулятора).

Дальнейшее описание реализации этого процесса будет проведено с самого высшего уровня (непосредственно сам процесс Main) до наиболее глубоко вложенных процедур. Так же при описании реализации либо процесса, либо произвольной процедуры буду указывать ссылки на соответствующие рисунки, приведенные в приложениях данного документа.

2.2.1 Процесс Main.

- Данный процесс является главным в работе системы в целом. Он отвечает за синхронизацию работы блоков стратегий, симулятора и GUI. Также он ответственен за начало и окончание процессов симуляции.
- Типы данных, константы и переменные:
  - Константы:

Название константы	Тип	Значение	Описание
NAMESHIP_NUM	Integer	5	Количество кораблей различного типа.
CHECK_ITER	Integer	10	Через какое количество тактов симуляции делать проверку на возможность ничейного исхода симуляции морского боя.

  - Перечислимые типы:

Название типа	Значения	Описание
tErrorName	EN_OK EN_COULD_NOT_OPEN_FILE EN_SHIP_NOT_FOUND EN_ERROR_PROPERTY EN_NOT_ENOUGH_AMMO EN_INIT_SOCKET_FAULT EN_SEND_DATA_FAULT EN_WRONG_MAP EN_GEN_POINTS_FAULT EN_GET_COORD_FAULT	Код ошибки.

  - Переменные:

Название переменной	Тип	Описание
gWeaponryCharacter	tArrayWeaponryCharacter	Массив с характеристиками для каждого типа оружия.
gShipsCharacter	tArrayShipsCharacter	Массив с характеристиками для каждого типа корабля.
gShips	tStringShipParameters	Массив кораблей, доступных стратегиям.
gShots	tStringShotParameters	Массив выстрелов.
gASI	tArraySI	Массив стратегий, ведущих морской бой.
errcode	tErrorCode	Код выполнения операций.
errname	tErrorName	Код ошибки.
simtime	Integer	Номер такта симуляции.
victory	Boolean	Флаг присуждения победы.

Реализация процесса Main (см. Рисунок 41 - Рисунок 45) велась согласно списку задач, указанному в п. 4 документа [1], и интерфейсу, описанному в п. 2.3.1 документа [2].

После описания различных вспомогательных процедур и объявления новых типов данных, переменных и констант, описание которых приведено выше, начинается непосредственно описание поведения процесса Main.

Работа процесса начинается с установки глобальной переменной gASI в значения TRUE для каждой из стратегий в функции SetSI. Эта переменная является индикатором, по которому можно определить есть ли

еще хоть один корабль этой стратегии. Т.о. когда все корабли i-ой стратегии будут уничтожены, значение переменной gASI(i) примет значение FALSE. После установки этой переменной, ее необходимо экспортировать, т.е. чтобы любой процесс, далее использующий эту переменную, работал уже с обновленной копией ее.

После этого происходит вызов функции InitDataShipsWeaponry для чтения информации из конфигурационного файла с заполнением соответствующих полей глобальных структур gWeaponryCharacter и gShipsCharacter. В зависимости от кода возврата (была ошибка при выполнении процедуры или нет) мы либо прекращаем выполнение процесса Main, при этом посылаем всем стратегиям сигнал LoseStrategy для завершения работы процессов стратегий и освобождаем все занятые ресурсы, либо продолжаем выполнение.

Затем осуществляется экспорт переменных gWeaponryCharacter и gShipsCharacter и вызов функции InitStrategyFleet для инициализации флота каждой из стратегий (инициализация переменной gShips). При ошибке в выполнении этой процедуры мы аналогично выходим из процесса Main, а при успешном завершении выполнения процедуры экспортируем переменные gShips и gShots и посылаем стратегиям сигнал InitDataComplete, показывающий, что флот каждой из стратегий был успешно проинициализирован.

После этого посылаем сигнал InitSocket процессу DataSender и ждем ответа в качестве сигнала InitSocketComplete с кодом выполнения. Если инициализация прошла успешно, то продолжаем выполнение, в противном случае завершаем процесс.

Затем формируются и отправляются сигналы ShipsCurrentState и VisibleShips соответственно с информацией о своем флоте и видимых кораблях противника. Это происходит при вызове функций SendCurrentState, SetVisibility и SendVisibleShips.

По завершении выполнения этих трех функций отправляем сигнал SendData процессу DataSender и ждем прихода сигнала SendDataComplete с кодом выполнения. Если действия по передаче информации выполнились успешно, то начинаем выполнение такта симуляции с отправки сигнала StartSimulation всем стратегиям. Все действия связанные с обработкой одного такта симуляции выполняются в процедуре SimIteration. Если при выполнении процедуры возникли ошибки, то выходим из процесса, иначе снова отправляем сигнал SendData для того, чтобы процесс DataSender смог выслать обновленную информацию графическому интерфейсу. Если же в процессе выполнения процедуры SimIteration был выставлен флаг victory в значение TRUE, то значит, что определилась стратегия – победитель, и процесс симуляции заканчивается. Если же этот флаг не был выставлен, то процесс симуляции продолжается и снова посылается группа сигналов ShipsCurrentState и VisibleShips для каждой стратегии с помощью вызова функций SendCurrentState, SetVisibility и SendVisibleShips.

После этого делается проверка на ничейный результат. Если проверка прошла и действительно зафиксирован ничейный результат, то снова повторяется процедура отправки сообщения процессу Router, а после этого симуляция завершается, в противном случае симулятор уходит на еще один так симуляции, начинающийся с отправки сигнала StartSimulation.

2.2.2 Процедура InitDataShipsWeaponry.

- Инициализация переменных gShipsCharacter и gWeaponryCharacter значениями, прочитанными из конфигурационного файла «DataValue.pr». По окончании работы возвращает код выполнения (см. Рисунок 46, Рисунок 47).
- Типы данных, переменные и константы:
  - Переменные:

Название переменной	Тип	Описание
srcFile	TextFile	Путь и имя файла со значениями параметров кораблей и снарядов.

2.2.3 Процедура InitStrategyFleet.

- Проверка карты на возможность размещения флотов, инициализация каждого флота и сохранение данных в переменной gShips (см. Рисунок 48, Рисунок 49).
- Типы данных, переменные о константы:
  - Константы:

Название константы	Тип	Значение	Описание
OK	Integer	1	Результат выполнения функции.

- Переменные:

Название переменной	Тип	Описание
shipParam	tShipParameters	Параметры корабля.
shipType	tNameShip	Тип корабля.
shipId	Integer	Идентификатор корабля.
nAS	Integer	Счетчик по количеству стратегий.
nST	Integer	Счетчик по типам кораблей.
i	Integer	Счетчик по кораблям определенного типа.
maxShips	Integer	Максимальное количество кораблей определенного типа.
Radius	Integer	Радиус, в котором могут быть расположены все корабли флота.
Deep	Integer	Максимальная посадка корабля во флоте.
pX	Integer	Абсцисса центра, вокруг которого можно разместить весь флот.
pY	Integer	Ордината центра, вокруг которого можно разместить весь флот.

Выполнение процедуры начинается с вызова функции checkMap, которая проверяет карту. Если проверка завершилась ошибкой, то сразу же выходим из этой процедуры, предварительно выставив значение глобальной переменной errname в значение кода ошибки, в противном же случае вызываем функцию InitRD, которая вычисляет значения минимальное значение радиуса круга, в который можно поместить все корабли, и максимальное значение глубины посадки корабля. При успешном выполнении вызывается еще одна процедура generatePoints, которая ищет на карте точки, вокруг которых можно разместить весь флот стратегии. Если такие точки найдены, то дальше выполняется цикл по количеству стратегий, в котором происходит инициализация всех кораблей каждой стратегии.

2.2.4 Процедура InitRD.

- Вычисление минимального значения радиуса круга, в который можно поместить все корабли, и максимального значения глубины посадки корабля (см. Рисунок 50). Радиус круга вычисляется по формуле $r = \left[\frac{\sqrt{N}}{2} \right] + 1$, где N – количество кораблей во флоте. Глубина же выбирается максимальной из всех возможных по всем типам кораблей.
- Типы данных, переменные и константы:
  - Переменные:

Название переменной	Тип	Описание
r	Integer	Максимальное количество кораблей во флоте.

2.2.5 Процедура GetMaxShips.

- Возвращает максимальное количество кораблей определенного типа в одном флоте любой из стратегий (см. Рисунок 51).
- Типы данных, переменные и константы:
  - Переменные:

Название переменной	Тип	Описание
Name	tNameShip	Тип корабля.

2.2.6 Процедура GetCoordinate.

- Эта процедура находит точку с координатами, причем глубина в этой точке должна быт не меньше глубины посадки корабля (см. Рисунок 52, Рисунок 53).
- Типы данных, переменные и константы:
  - Переменные:

Название переменной	Тип	Описание
coord	tCoordinates	Координаты корабля.
cDeep	Integer	Текущая глубина.
tDeep	Integer	Глубина в данной точке карты.
cCoord	tCoordinates	Текущие координаты.
xMax	Integer	Максимальная координата по оси абсцисс.
xMin	Integer	Минимальная координата по оси абсцисс.
yMax	Integer	Максимальная координата по оси ординат.
yMin	Integer	Минимальная координата по оси ординат.
IsCoord	Boolean	Флаг – были ли найдены координаты корабля.

В этой процедуре реализовано следующее правило расположения кораблей: из двух подходящих точек, т.е. в обеих точках может быть расположен корабль, выбирается та точка, у которой глубина наименьшая. Это сделано так, потому что, если размещать мелководные корабли на большие глубины, то может не хватить глубоководных позиций для всех кораблей с большой посадкой.

2.2.7 Процедура IsBusyPoint.

- Эта процедура возвращает значение либо TRUE, либо FALSE в зависимости от того, занята уже или нет выбранная точка для размещения корабля (см. Рисунок 54).
- Типы данных, переменные и константы:
  - Переменные:

Название переменной	Тип	Описание
length	Integer	Количество кораблей в списке gShips.
iBP	Integer	Счетчик по всем кораблям.

2.2.8 Процедура SetVisibility.

- Установка для каждого корабля, в области видимости какой из стратегий он находится (см. Рисунок 55).
- Типы данных, переменные и константы:
  - Переменные:

Название переменной	Тип	Описание
n	Integer	Количество кораблей.
ins	Integer	Счетчик по всем кораблям.
jns	Integer	Счетчик по всем кораблям.
r	Integer	Расстояние между кораблями.

В начале вызывается функция zeroizeVisibility, которая сбрасывает все значения видимости кораблей каждой из стратегий. После этого идет цикл по все кораблям, в котором идет еще один цикл по всем кораблям, начиная со следующего корабля в списке. На каждой итерации второго цикла вычисляется расстояние между двумя кораблями и проверяется, попадает ли один из кораблей в область видимости другого корабля. Если да, то выставляется флаг TRUE, иначе FALSE.

2.2.9 Процедура zeroizeVisibility.

- В этой процедуре организован цикл по всем кораблям, в котором для каждого значения массива VisibleToStrategy выставляется значение FALSE (см. Рисунок 56).

2.2.10 Процедура SendCurrentState.

- Отправление каждой стратегии сигнала ShipsCurrentState с текущим состоянием ее флота (см. Рисунок 57).
- Типы данных, переменные и константы:
  - Переменные:

Название переменной	Тип	Описание
i	Integer	Счетчик цикла по количеству кораблей.
n	Integer	Количество кораблей.
strId	tStrategyId	Идентификатор стратегии.
shCS	tShipCurrentState	Структура с характеристиками корабля.
shSCS	tStringShipCurrentState	Список характеристик для каждого корабля флота.

В этой процедуре реализован цикл по всем кораблям, в котором создается список кораблей, принадлежащих одной стратегии. После того, как список был сформирован, посылается сигнал ShipsCurrentState. Для удобства реализации данной функции на начальном этапе инициализации флота, был предусмотрен следующий порядок следования кораблей: во-первых, для каждой из стратегии корабли идут в одинаковом порядке, во-вторых, сначала идут все корабли одной стратегии, затем уже идут корабли другой стратегии. Это сделано для того, чтобы избежать лишней траты времени на поиск кораблей, принадлежащих определенной стратегии, во всем списке кораблей. Сейчас все корабли одной стратегии идут друг за другом, и если начались корабли другой стратегии, то уже не имеет смысла искать корабли предыдущей стратегии в оставшейся части списка.

2.2.11 Процедура SendVisibleShips.

- Отправление каждой структуре список видимых вражеских кораблей (см. Рисунок 58).
- Типы данных, переменные и константы:
  - Переменные:

Название переменной	Тип	Описание
i	Integer	Счетчик цикла по количеству кораблей.
n	Integer	Количество кораблей.
j	Integer	Счетчик цикла по количеству стратегий.
shES	tEnemyShip	Структура с характеристиками вражеского корабля.
shSES	tStringEnemyShip	Список характеристик для каждого вражеского корабля.

В этой процедуре организован внешний цикл по количеству стратегий, в котором есть внутренний цикл по всем кораблям. Если выполняется условие, что хотя бы один корабль попадает в область видимости другого, т.е. установлен флаг VisibleToStrategy в значение TRUE, то информация об этом корабле заносится в список, который после просмотра всех кораблей в массиве отправляется определенной стратегии, как список видимых для нее кораблей.

2.2.12 Процедура Err2Str.

- В этой процедуре введено взаимнооднозначное сопоставление кода ошибки, хранящегося в глобальной переменной errname, текстовой строке, поясняющей эту ошибку (см. Рисунок 59).

2.2.13 Процедура SimIteration.

- Эта процедура вызывается на каждом такте симуляции из процесса Main. В ней выполняются все основные действия по получению приказов от стратегий и сигнала о завершении отправки приказов; по отправке стратегиям сигналов о принятии приказа к выполнению или его отклонению; по интерпретации и выполнению; по проверки стратегий на проигрыш и выигрыш (см. Рисунок 60 - Рисунок 63).
- Типы данных, переменные и константы:
  - Константы:

Название константы	Тип	Значение	Описание
MAX_SHIP_ID	Integer	(MaxTypeBattleship + MaxTypeCruiser + MaxTypeDestroyer + MaxTypeMissileCutter + MaxTypeRepairBoat)*AmountStrategy	Максимально возможное количество кораблей.

  - Перечислимые типы:

Название типа	Значения	Описание
tLiteralsOrder	SPEED TURN FIRE	Тип приказа.

- Структуры данных:

Название типа	Поле	Тип поля	Описание
tStructOrder			Параметры приказа:
	IsOrder	Boolean	Был ли приказ.
	Order	tLiteralsOrder	Тип приказа.
	IsOrderPass	Boolean	Успешно ли выполнился приказ.
	StrId	tStrategyId	Идентификатор стратегии.
	FA	tFactorAcceleration	Множитель ускорения.
	HC	tHeadingChange	Изменение направления.
	WN	tNameWeaponry	Тип оружия.
	SSL	tStringShotsList	Список выстрелов.

- Хэш-таблицы:

Название типа	Тип индекса	Тип значения	Описание
tArrayOrder	tShipId	tStructOrder	Список с приказами для каждого корабля.

- Переменные:

Название переменной	Тип	Описание
st	Integer	Такт симуляции.
strASI	tArraySI	Список флагов для каждой стратегии.
shId	tShipId	Идентификатор корабля.
strId	tStrategyId	Идентификатор стратегии.
shFA	tFactorAcceleration	Множитель ускорения.
shHC	tHeadingChange	Изменение направления.
shWN	tNameWeaponry	Тип оружия.
shSSL	tStringShotsList	Список выстрелов.
shipOrders	tArrayOrder	Список с приказами для кораблей.

В самом начале выполнения процедуры происходит обнуление структуры приказов для каждого из кораблей (shipOrders) и обнуление списка стратегий (strASI), от которых будем ждать приход сигнала FinishSimulation.

После этого попадаем в состояние, в котором можем принимать только четыре сигнала: OrderSpeedRequest, OrderTurnRequest, OrderFireRequest, FinishSimulation. Причем в этом состоянии будем находиться до тех пор, пока каждая из стратегий не пришлет сигнал FinishSimulation.

При получении одного из трех сигналов OrderSpeedRequest, OrderTurnRequest, OrderFireRequest вносится информация в структуру записи приказов shipOrders, после чего вызывается соответствующая функция-обработчик OrderSpeed, OrderTurn, OrderFire. И при успешном выполнении процедуры выставляется флаг IsOrderPass структуры shipOrders в значение TRUE. Здесь реализовано так, что принимается только один приказ для каждого корабля.

После того, как от всех стратегий пришел сигнал FinishSimulation, начинается обработка пришедших приказов. Сначала высылается ответ на пришедший приказ, т.е. высылаются сигналы OrderSpeedResponse или OrderTurnResponse, или OrderFireResponse, или OrderSpeedReject, или OrderTurnReject, или OrderFireReject. Затем вызываются три функции для отработки приказа: ChangeCoordinate, CollisionShip, UpdateShots. После чего вызывается функция CheckStrategies, в которой проверяются все стратегии на проигрыш и выигрыш.

2.2.14 Процедура NullingSI.

- В этой процедуре реализован цикл, в котором обнуляется флаг прихода сигнала FinishSimulation для каждой стратегии (см. Рисунок 64).
- Типы данных, переменные и константы:
  - Переменные:

Название переменной	Тип	Описание
i	Integer	Счетчик цикла по количеству стратегий.

2.2.15 Процедура IsASIComplete.

- В этой процедуре в цикле по всем стратегиям, которые «выжил», т.е. имеют корабли, на предыдущем шаге (эта информация хранится в глобальной переменной gASI), осуществляется проверка прихода сигнала FinishSimulation. Если сигнал пришел для каждой такой стратегии, то возвращается значение TRUE, иначе FALSE (см. Рисунок 65).
- Типы данных, переменные и константы:
  - Переменные:

Название переменной	Тип	Описание
i	Integer	Счетчик цикла по количеству стратегий.

2.2.16 Процедура CheckStrategies.

- Проверка стратегий на выигрыш и проигрыш (см. Рисунок 66, Рисунок 67).
- Типы данных, переменные и константы:
  - Переменные:

Название переменной	Тип	Описание
i	Integer	Счетчик цикла.
n	Integer	Количество кораблей.
j	Integer	Количество уцелевших стратегий.
ns	Integer	Номер последней живой стратегии.

Сначала в цикле по всем кораблям выставляется флаги в массиве стратегий, т.е. проверяется, какие стратегии уцелели после очередного такта симуляции. Затем в цикле по всем стратегиям проверяется, какие стратегии проиграли на этом такте симуляции, и им высылается сигнал StrategyLose. Так же в этом цикле считается общее количество уцелевших стратегий. Если количество уцелевших стратегий равно нулю, то тогда обнуляется массив выстрелов gShots и выставляется флаг окончания симуляции victory, но никому не высылается сигнал StrategyVictory. Если количество уцелевших стратегий равно единице и массив выстрелов пустой, то тогда выставляется флаг окончания симуляции victory, и выигравшей стратегии посылается сигнал StrategyVictory.

2.2.17 Процедура NullingShipOrders.

- В этой процедуре реализован цикл по кораблям, в котором обнуляются флаги IsOrder и IsOrderPass списка приказов shipOrders (см. Рисунок 68).
- Типы данных, переменные и константы:
  - Переменные:

Название переменной	Тип	Описание
i	Integer	Счетчик цикла по всем кораблям.

2.2.18 Процедура SendResposeOnOrders.

- Отправление ответов стратегиям на их запросы с приказами (см. Рисунок 69).
- Типы данных, переменные и константы:

- Переменные:

Название переменной	Тип	Описание
i	Integer	Счетчик цикла по всем кораблям.

В этой процедуре организован внешний цикл по всем кораблям, в котором на каждой итерации проверяется: был ли приказ для этого корабля. Если не было, то переходим к следующему кораблю. Если все-таки приказ был для этого корабля, то тогда проверяется, какой именно приказ пришел: SPEED, TURN, FIRE. После определения типа приказа, смотрится результат его выполнения: в зависимости от него отправляется либо OrderSpeedResponse, OrderTurnResponse, OrderFireResponse, либо OrderSpeedReject, OrderTurnReject, OrderFireReject.

2.2.19 Процедура OrderFire.

- Обработка сигнала OrderFireRequest: добавление записи в массив gShots и проверка корректность пришедшего сигнала (см. Рисунок 70 и Рисунок 71).
- Типы данных, переменные и константы:
  - Переменные:

Название переменной	Тип	Описание
CurCoor	tCoordinates	Текущая координата выстрела.
NameWeaponry	tNameWeaponry	Тип оружия, которым был произведен выстрел.
shipid	tShipId	Идентификатор корабля, который произвел выстрел.
listshot	tStringShotsList	Список целей, по которым были произведены выстрелы.
i	Integer	Счетчик цикла по списку выстрелов.
countershot	Integer	Количество целей.
leng	Integer	Количество кораблей.
ShotParam	tShotParametrs	Параметры выстрела.
bool	Boolean	Правильны ли параметры сигнала.

В начале обработки вызывается функция GetProperty, проверяющая возможность выстрелить этого корабля данным типом оружия. Функция возвращает переменную типа Boolean. Если значение FALSE, то выдается ошибка EN_ERROR_PROPERTY. Функция GetNumberShip возвращает номер корабля с заданным ID в списке gShots. Функция GetLengShot проверяет соответствие количества выстрелов, количеству пусковых установок.

Если приказ корректен и если стрелять данным оружием можно, то вызывается функция GetAmmunition, которая проверяет наличие боеприпасов. Если боезапас не израсходован, то производиться запись выстрела в массив выстрелов gShots. Функция GetCurrentCoordinate возвращает в переменной CurCor точку, откуда производиться выстрел. Конечную координату вычисляет функция GetFinishCoordinate. Она проверяет: возможен ли выстрел данным оружием на данную дистанцию. Если да, то CurCoor возвращается без изменений. Если нет, то вычисляется точка, куда должен упасть снаряд.

Функция возвращает ошибку, когда приказ отдан несуществующему кораблю или когда кораблю отдан приказ: стрелять оружием, которого у него нет.

2.2.20 Процедура GetCurrentCoordinate.

- По известному ShipId находит корабль и возвращает в переменной CurCoor точку, откуда производиться выстрел (см. Рисунок 72).
- Типы данных, переменные и константы:
  - Переменные:

Название переменной	Тип	Описание
shipid	tShipid	Идентификатор корабля.
CurCoor	tCoordinates	Начальная координата выстрела.
i	Integer	Счетчик цикла.
leng	Integer	Количество кораблей.

2.2.21 Процедура GetProperty.

- По известному ShipId находит корабль и проверяет для него возможность выстрела данным типом оружия (см. Рисунок 73 и Рисунок 74).
- Типы данных, переменные и константы:
  - Переменные:

Название переменной	Тип	Описание
shipid	tShipid	Идентификатор корабля.
i	Integer	Счетчик цикла.
leng	Integer	Количество кораблей.

2.2.22 Процедура GetNumberShip.

- Возвращает номер корабля с заданным ID в списке выстрелов gShots (см. Рисунок 75).
- Типы данных, переменные и константы:
  - Переменные:

Название переменной	Тип	Описание
shipid	tShipid	Идентификатор корабля.
i	Integer	Счетчик цикла.
Number	Integer	Номер корабля с заданным shipid в gShots.

2.2.23 Процедура GetAmmunition.

- Проверяет количество боеприпасов и следит за расходованием снарядов (см. Рисунок 76).
- Типы данных, переменные и константы:
  - Переменные:

Название переменной	Тип	Описание
NumberShip	Integer	Номер корабля в gShots.

2.2.24 Процедура GetLenghtShot.

- Проверка соответствия количества выстрелов количеству пусковых установок. Если количество выстрелов от корабля превышает кол-во пусковых установок, то функция отбрасывает лишние выстрелы (см. Рисунок 77).
- Типы данных, переменные и константы:
  - Переменные:

Название переменной	Тип	Описание
number	Integer	Номер корабля в gShips.
lenght	Integer	Количество выстрелов данным кораблем.
shipname	tNameShip	Тип корабля.

2.2.25 Процедура GetFinishCoordinate.

- Возвращает конечную точку траектории полета снаряда, обрабатывает ситуацию недолета снаряда, вычисляет точку падения снаряда (см. Рисунок 78).
- Типы данных, переменные и константы:
  - Переменные:

Название переменной	Тип	Описание
CurrentCoordinate	tCoordinates	Начальные координаты.
FinishCoordinate	tCoordinates	Конечные координаты.

Coor	tCoordinates	Новые координаты.
Range	Integer	Расстояние между начальными и конечными координатами.
cX	Integer	Начальная координата по оси абсцисс.
cY	Integer	Начальная координата по оси ординат.
fX	Integer	Конечная координата по оси абсцисс.
fY	Integer	Конечная координата по оси ординат.

2.2.26 Процедура OrderTurn.

- Процедура вызывается как обработчик сигнала OrderTurnRequest: находится нужный корабль в списке и изменяет его направление в зависимости от значения Heading. Процедура возвращает ошибку только в том случае, если приказ отдан несуществующему кораблю (см. Рисунок 79 и Рисунок 80).
- Типы данных, переменные и константы:
  - Переменные:

Название переменной	Тип	Описание
Shipid	tShipid	Идентификатор корабля.
Heading	tHeading	Изменение направления движения корабля.
head	tHeading	Значение направления корабля.
i	Integer	Счетчик цикла.
leng	Integer	Количество кораблей.
number	Integer	Новое направление корабля в числовом виде.

2.2.27 Процедура OrderSpeed.

- Процедура вызывается как обработчик сигнала OrderSpeedRequest: изменяет скорость корабля (см. Рисунок 81 и Рисунок 82).
- Типы данных, переменные и константы:
  - Переменные:

Название переменной	Тип	Описание
shipid	tShipid	Идентификатор корабля.
Acceleration	tFactorAcceleration	Коэффициент ускорения.
speed	Integer	Скорость корабля.
i	Integer	Счетчик цикла.
leng	Integer	Количество кораблей.

В первом цикле функция находит в таблице gShots корабль, которому был послан сигнал. Новая скорость вычисляется по правилу: «старая скорость» + «максимальное ускорение для данного типа корабля» * «Acceleration». Далее производиться проверка: не превысила ли скорость максимального или минимального значения. Если да, то производиться корректировка скорости. Функция может выдать ошибку, только в том случае, если приказ отдан не существующему кораблю.

2.2.28 Процедура ChandeCoordinate.

- Вызывается после приема сигналов от всех стратегий для обработки перемещения кораблей, для проверки посадки корабля на мель и связанную с этим потерю ресурса и гибель корабля (см. Рисунок 83).
- Типы данных, переменные и константы:
  - Переменные:

Название переменной	Тип	Описание
i	Integer	Счетчик цикла.
j	Integer	Счетчик цикла.
speed	Integer	Скорость корабля (в клетках).
len	Integer	Общее количество кораблей.

St	tStrngShipParametrs	Часть списка кораблей.

Переменная speed содержит скорость корабля в клетках. Ее введение необходимо, т.к. скорость корабля в клетках равна скорости корабля при направлениях NORTH, EAST, SOUTH, WEST и равна скорости корабля деленной на корень из двух при всех остальных направлениях. После модификации скорости, изменяем координаты корабля. Для этого мы в цикле вызываем функцию ReadHeading. После изменение координат функция проверяет посадку на мель. Если посадки на мель не произошло, то движение продолжается. В противном случае корабль останавливается на мели. Цикл выполняется до тех пор, пока корабль не сядет на мель или пока не проедет количество клеток, равное его скорости.

2.2.29 Процедура ReadHeading.

- Изменяет координаты, проверяет посадку на мель и выход за границы экрана (см. Рисунок 84).
- Типы данных, переменные и константы:
  - Переменные:

Название переменной	Тип	Описание
ship	tShipParameters	Характеристики корабля.
CoorXY	tCoordinates	Новые координаты корабля.
Deep	Integer	Глубина посадки корабля.
PointDeep	Integer	Глубина в точке, куда переместился корабль.
PointDeep2	Integer	Глубина в точке, откуда переместился корабль.
Res	Integer	Ресурс корабля.

В самом начале вызывается функция StepCoor, изменяющая координаты корабля на 1 клетку в зависимости от направления. Значение новых координат записывается в CoorXY. Далее следует проверка: может ли корабль плыть в эту точку (нет ли там мели). Если может, то перемещаем его в эту клетку. Если нет, то проверяем: сидел ли ранее корабль на мели. Если нет, корабль теряет ресурс и останавливается. Если да, то обнуляем скорость корабля.

2.2.30 Процедура StepCoor.

- Изменяет координату корабля на 1 клетку в зависимости от направления (см. Рисунок 85 и Рисунок 86).

2.2.31 Процедура UpdateShots.

- Обрабатывает перемещение снарядов (см. Рисунок 87 и Рисунок 88).
- Типы данных, переменные и константы:
  - Переменные:

Название переменной	Тип	Описание
St	tStringShotParameters	Список выстрелов.
St1	tStringShipParameters	Список кораблей.
Distance	Integer	Расстояние между текущими координатами снаряда и координатами цели.
i	Integer	Счетчик цикла.
j	Integer	Счетчик цикла.
Res	Integer	Ресурс корабля.
lenshot	Integer	Общее количество выстрелов.
lenship	Integer	Общее количество кораблей.

В начале для каждого из снарядов проверяется совпадение начальных и конечных координат. Если координаты совпадают, значит, на прошлом такте снаряд достиг заданной точки, и мы его убираем из списка gShots. Затем мы проверяем, сколько осталось лететь снаряду (вызов процедуры GetDistance), и сравниваем с его скоростью. Если снаряд не может долететь за один ход, вызывается процедура SpeedCoor, которая передвигает снаряд на количество клеток, равное значению скорости снаряда. В противном случае

мы проверяем, есть ли в точке FinishCoordinate корабль. Если корабля нет, то убираем выстрел из списка gShots. Если есть, то в зависимости от типа снаряда добавляем (уменьшаем) ресурс корабля. Если корабль убит, то удаляем его из списка и посылаем сообщение DestroyShip.

2.2.32 Процедура SpeedCoor.

- Изменяет текущие координаты снаряда в зависимости от скорости снаряда, если он не может долететь до цели за один такт (см. Рисунок 89).
- Типы данных, переменные и константы:
  - Переменные:

Название переменной	Тип	Описание
Dis	Integer	Скорость полета.
sqr	Integer	Квадратный корень из суммы квадратов разности конечных и начальных координат.
cX	Integer	Начальная координата по оси абсцисс.
fX	Integer	Начальная координата по оси ординат.
cY	Integer	Конечная координата по оси абсцисс.
fY	Integer	Конечная координата по оси ординат.

2.2.33 Процедура CollisionShip.

- Обрабатывает столкновение кораблей (см. Рисунок 90 - Рисунок 92).
- Типы данных, переменные и константы:
  - Переменные:

Название переменной	Тип	Описание
St	tStringShipParametrs	Часть списка кораблей.
lengship	Integer	Общее количество кораблей.
i	Integer	Счетчик цикла.
j	Integer	Счетчик цикла.
b	Boolean	Были ли на прошлом проходе цикла столкновения кораблей.

Процедура сравнивает координаты кораблей и, если находит совпадения, вызывает функцию Collision, которая в зависимости от направления и скорости кораблей уменьшает их ресурс. В дальнейшем проверяется наличие ресурса у кораблей, и, если он не равен нулю, вызывается процедура Heading, которая отодвигает корабль назад на одну клетку до столкновения.

2.2.34 Процедура Collision.

- В зависимости от направления и скорости корабля уменьшает его ресурс (см. Рисунок 93 - Рисунок 96).
- Типы данных, переменные и константы:
  - Переменные:

Название переменной	Тип	Описание
h1	Integer	Направление первого корабля в числовом эквиваленте.
h2	Integer	Направление второго корабля в числовом эквиваленте.
Res	Integer	Ресурс корабля

Представляет направления в числовом виде и берет их разность. В зависимости от разности, потери ресурса вычисляются по различным формулам (см. п. 3.5.2 документа [1]).

2.2.35 Процедура abs.

- Возвращает модуль числа (см. Рисунок 97).

2.2.36 Процедура Heading.

- Отодвигает корабль назад на одну клетку до столкновения (см. Рисунок 98 и Рисунок 99).

2.2.37 Процедура SetSI.

- В этой процедуре, в цикле, устанавливается флаг gASI для каждой стратегии. Он отображает, что на данном шаге симуляции у этой стратегии есть, по крайней мере, один живой корабль (см. Рисунок 100).
- Типы данных, переменные и константы:
  - Переменные:

Название переменной	Тип	Описание
i	Integer	Счетчик цикла по количеству стратегий.

2.2.38 Процедура KillStrategy.

- В этой процедуре организован цикл по всем стратегиям, в котором проверяется флаг gASI. Если он равен TRUE. то тогда этой стратегии посылается сигнал LoseSrategy (см. Рисунок 101).
- Типы данных, переменные и константы:
  - Переменные:

Название переменной	Тип	Описание
i	Integer	Счетчик цикла по количеству стратегий.

2.2.39 Процедура IsWeaponry.

- В этой процедуре организован цикл по всем кораблям, в котором осуществляется проверка наличия боеприпасов у корабля. Если боеприпасы есть, то происходит выход из процедуры, и система идет еще как минимум на один такт симуляции. Если же ни у одного из кораблей боеприпасов не оказалось, то осуществляется проверка структуры выстрелов gShots. Если она пустая, то происходит выход из процесса симуляции, иначе идем на следующий такт симуляции. (См. Рисунок 102)
- Типы данных, переменные и константы:
  - Переменные:

Название переменной	Тип	Описание
i	Integer	Счетчик цикла по количеству кораблей.
n	Integer	Количество кораблей.

2.3 Реализация модуля DataSender (блок симулятора).

Данный модуль отвечает за взаимодействие с графическим интерфейсом (GUI). В его задачи входит инициализация соединения и синхронизация информации о кораблях и выстрелах. Более детальное описание реализации этого модуля будет проведено, начиная с процесса DataSender, и далее для всех процедур.

2.3.1 Процесс DataSender.

- Процесс по сигналам от процесса Main отсылает данные к GUI и проверяет корректность отправки (см. Рисунок 103 - Рисунок 106).
- Типы данных, переменные и константы:
  - Константы:

Название константы	Тип	Значение	Описание
rcNoErrors	Integer	0	Возвращаемое значение при успешном завершении.
rcCommonFault	Integer	-1	Возвращаемое значение при неуспешном завершении.

- Переменные:

Название переменной	Тип	Описание
tmpRes	Integer	Счетчик цикла.
gShips	tStringShipParameters	Импортируемый массив данных о кораблях.
gShots	tStringShotParameters	Импортируемый массив данных о выстрелах.
impShips	tStringShipParameters	Импортированные данные о кораблях.
impShots	tStringShotParameters	Импортированные данные о выстрелах.

Процесс изначально находится в состоянии Listening. В этом состоянии процесс DataSender ожидает управляющие сигналы от процесса Main.

По приходу сигнала InitSocket DataSender начинает инициализацию соединения. Вызывается функция установления соединения InitConnection. При успешном соединении GUI-клиенту посылается параметр AmountStrategy, обозначающий число стратегий. При удачной посылке параметра GUI-клиенту высылается карта. При успешной посылке карты DataSender считает установку соединения успешной и возвращает процессу Main сигнал InitSocketComplete с параметром EC_OK. Если хотя бы на одном шаге возникла ошибка, то процессу Main возвращается InitSocketComplete с параметром EC_FAULT.

По приходу сигнала SendData начинает пересылку данных GUI-клиенту. Для получения данных, которые необходимо переслать, DataSender первым делом импортирует переменные gShips и gShots и сохраняет их в impShips и impShots. После чего процесс вызывает PushShips для отсылки данных о кораблях, которые берутся из impShips. Если посылка прошла успешно вызывается PushShots для отсылки данных о выстрелах, которые берутся из impShots. Если передача всех данных завершилась успешно - процессу Main отсылается сигнал SendDataComplete с параметром EC_OK. Если на каком-либо шаге передача прошла неуспешно процессу Main отсылается сигнал SendDataComplete с параметром EC_FAULT.

2.3.2 Процедура PushShips.

- Отправка данных о кораблях из импортированной переменной impShips. Возвращает значение, подтверждающее успех или неуспех передачи данных (см. Рисунок 107).
- Типы данных, переменные и константы:
  - Константы:

Название константы	Тип	Значение	Описание
rcNoErrors	Integer	0	Возвращаемое значение при успешном завершении.
rcCommonFault	Integer	-1	Возвращаемое значение при неуспешном завершении.

- Переменные:

Название переменной	Тип	Описание
tmpRes	Integer	Счетчик цикла.
impShips	tStringShipParameters	Импортированные данные о кораблях.

Сначала процедура проверяет длину impShips, если длина нулевая отсылается «заглушка», необходимая GUI, для определения нового такта. Если длина impShips ненулевая, то в цикле вызываются внешние C функции: PushFixedShipPart - для отсылки фиксированной части данных о кораблях; PushVariableShipPart - для отсылки части данных, которая имеет разную длину, зависящую от числа стратегий в симуляторе. В случае удачной отсылки заглушки или массива возвращается значение rcNoError, которое распознается в процессе DataSender, для определения ошибки. В противном случае возвращается значение rcCommonFault, которое распознается в процессе DataSender, для определения успешной передачи.

2.3.3 Процедура PushShots.

- Отправка данных о кораблях из импортированной переменной impShots. Возвращает значение, подтверждающее успех или неуспех передачи данных (см. Рисунок 108).
- Типы данных, переменные и константы:
  - Константы:

Название константы	Тип	Значение	Описание
rcNoErrors	Integer	0	Возвращаемое значение при успешном завершении.
rcCommonFault	Integer	-1	Возвращаемое значение при неуспешном завершении.

- Переменные:

Название переменной	Тип	Описание
tmpRes	Integer	Счетчик цикла.
impShots	tStringShotParameters	Импортированные данные о выстрелах.

Сначала процедура проверяет длину impShots, если длина нулевая, то отсылается «заглушка», необходимая GUI, для определения нового такта. Если длина impShots ненулевая, то в цикле вызывается внешняя C функция: PushShot - для отсылки данных о выстрелах. В случае удачной отсылки заглушки или массива возвращается значение rcNoError, которое распознается в процессе DataSender, для определения ошибки. В противном случае возвращается значение rcCommonFault, которое распознается в процессе DataSender, для определения успешной передачи.

2.3.4 Класс ZSocketData.

- Класс хранит в себе идентификатор ресурса «сокета» - SockID (см. Рисунок 154).

2.3.5 Функция GetSockID.

- Метод возвращает идентификатор сокета (см. Рисунок 157).

2.3.6 Функция SetSockID.

- Метод устанавливает значение идентификатора сокета (см. Рисунок 157).

2.3.7 Класс ZSocket.

- Класс является интерфейсом для простейших операций над сокетом (см. Рисунок 155).

2.3.8 Функция Init.

- Функция инициализирует среду окружения. В Windows для этого необходимо загрузить библиотеку «wsock2.dll», содержащую функции по работе с сокетами (см. Рисунок 158).

2.3.9 Функция Create.

- Функция создает сокет и сохраняет его идентификатор во внутренней переменной SockID (см. Рисунок 158).

2.3.10 Функция Destroy.

- Закрывает сокет и очищает среду окружения, выгружая wsock2.dll (см. Рисунок 158).

2.3.11 Функция ConnectToDomain.

- Функция соединяет сокет с хостом по доменному имени, передаваемому в виде строки через порт (см. Рисунок 158).

2.3.12 Функция ConnectToIP.

- Метод соединяет сокет с хостом по IP-адресу, передаваемому в виде строки, параметр strIPAddr, через порт, указанный в Port. Сначала метод переводит адрес во внутреннее представление, а затем вызывает библиотечную функцию connect с этим адресом. (См. Рисунок 158)

2.3.13 Функция GetSockAddr.

- Функция переводит строку, содержащую IP-адрес и переменную с адресом порта, во внутреннее представления адреса (см. Рисунок 158).

2.3.14 Функция Pop.

- Функция принимает данные из сокета в буфер, заданный указателем на массив символов. Данная функция является неблокирующейся, т.е. не дожидается пока придет нужное число байт и просто возвращает принятое число байт. Переменная MaxSize играет роль ограничителя. (См. Рисунок 158)

2.3.15 Функция Push.

- Функция предназначена для отправки данных в сокет. Для удобства реализованы три разных модификации метода для отправки массивов разных типов: char, u_short, u_long. Данные функции осуществляют перевод данных в сетевой формат (big endian). Функции возвращают true в случае успешной передачи данных и false в обратном случае. (См. Рисунок 158)

2.3.16 Класс ZSSocket.

- Класс является интерфейсом для сокета, используемого на серверной стороне. Предоставляет ряд дополнительных услуг по работе с сокетами, специфичных для серверной стороны. (См. Рисунок 156)

2.3.17 Функция BindToPort.

- Привязывает текущий сокет к порту, передаваемому в переменной Port, и устанавливает прослушивание порта для того (см. Рисунок 159).

2.3.18 Функция WaitForCall.

- Ожидает входящего соединения и возвращает через указатель **CallerAddress** адрес входящего соединения. Метод блокирующийся, т.е. пока соединение не произойдет он не выходит. (См. Рисунок 159)

2.3.19 Функция PopUntil.

- Выталкивает BufSize байтов из сокета, соединенного с клиентом, и сохраняет их в буфере, заданном указателем pBuffer. Метод блокирующийся, т.е. не выходит пока необходимое число байт не будет получено. (См. Рисунок 159)

2.3.20 Функция SetClient.

- Устанавливает идентификатор сокета через pCSocket для передачи данных клиенту (см. Рисунок 159).

2.4 Реализация модуля Router (блок симулятора).

Данный модуль предназначен для обеспечения взаимодействия между блоком симулятора и блоками стратегий. Ниже приведено описание реализации этого модуля, начиная непосредственно с процесса Router.

2.4.1 Процесс Router.

- Принятие сигналов от процесса Main и отправление их блокам стратегий и наоборот (см. Рисунок 22 - Рисунок 25).
- Типы данных, переменные и константы:
  - Переменные:

Название переменной	Тип	Описание
rTime	Integer	Номер такта симуляции.
rShipId	tShipId	Идентификатор корабля.
rStrategyId	tStrategyId	Идентификатор стратегии.
rNameShip	tNameShip	Тип корабля.
rStringEnemyShip	tStringEnemyShip	Список вражеских кораблей.
rStringShipCurrentState	tStringShipCurrentState	Список кораблей флота.
rFactorAcceleration	tFactorAcceleration	Коэффициент ускорения.
rHeadingChange	tHeadingChange	Направление поворота.
rNameWeaponry	tNameWeaponry	Тип оружия.
rStringShotsList	tStringShotsList	Список выстрелов.
gASI	tArraySI	Состояние стратегий.
rASI	tArraySI	Состояние стратегий.

Процесс всегда находится в одном и том же состоянии. Он принимает все сигналы идущие между блоками стратегий и блоком симулятора, обрабатывает их и отправляет дальше. Этот модуль введен для предотвращения отправления одной стратегий сигналов от имени другой стратегии и любых других подмен.

Все проверки выделены в отдельный модуль для уменьшения нагрузки на модуль Main. На данный момент никакие проверки в этом модуле не производятся – их планируется реализовать в будущем.

2.4.2 Процедура SendOrderSpeedRequest.

- По приходу запроса от стратегии производится проверка: жива ли стратеги. Если да, то от нее перенаправляется запрос далее, а если нет, то этот запрос отбрасывается. (См. Рисунок 26)

2.4.3 Процедура SendOrderSpeedResponse.

- По приходу сигнала от модуля Main делается проверка: жива ли стратегия. Если да, то сигнал перенаправляется стратегии, если же нет, то ничего не делается. (См. Рисунок 27)

2.4.4 Процедура SendOrderSpeedReject.

- По приходу сигнала от модуля Main делается проверка: жива ли стратегия. Если да, то сигнал перенаправляется стратегии, если же нет, то ничего не делается. (См. Рисунок 28)

2.4.5 Процедура SendOrderTurnRequest.

- По приходу запроса от стратегии производится проверка: жива ли стратеги. Если да, то от нее перенаправляется запрос далее, а если нет, то этот запрос отбрасывается. (См. Рисунок 29)

2.4.6 Процедура SendOrderTurnResponse.

- По приходу сигнала от модуля Main делается проверка: жива ли стратегия. Если да, то сигнал перенаправляется стратегии, если же нет, то ничего не делается. (См. Рисунок 30)

2.4.7 Процедура SendOrderTurnReject.

- По приходу сигнала от модуля Main делается проверка: жива ли стратегия. Если да, то сигнал перенаправляется стратегии, если же нет, то ничего не делается. (См. Рисунок 32)

2.4.8 Процедура SendOrderFireRequest.

- По приходу запроса от стратегии производится проверка: жива ли стратеги. Если да, то от нее перенаправляется запрос далее, а если нет, то этот запрос отбрасывается. (См. Рисунок 34)

2.4.9 Процедура SendOrderFireResponse.

- По приходу сигнала от модуля Main делается проверка: жива ли стратегия. Если да, то сигнал перенаправляется стратегии, если же нет, то ничего не делается. (См. Рисунок 36)

2.4.10 Процедура SendOrderFireReject.

- По приходу сигнала от модуля Main делается проверка: жива ли стратегия. Если да, то сигнал перенаправляется стратегии, если же нет, то ничего не делается. (См. Рисунок 38)

2.4.11 Процедура SendFinishSimulation.

- По приходу сигнала от стратегии производится проверка: жива ли стратеги. Если да, то от нее перенаправляется запрос далее, а если нет, то это сообщение отбрасывается. (См. Рисунок 31)

2.4.12 Процедура SendVisibleShip.

- По приходу сигнала от модуля Main делается проверка: жива ли стратегия. Если да, то сигнал перенаправляется стратегии, если же нет, то ничего не делается. (См. Рисунок 33)

2.4.13 Процедура SendShipsCurrentState.

- По приходу сигнала от модуля Main делается проверка: жива ли стратегия. Если да, то сигнал перенаправляется стратегии, если же нет, то ничего не делается. (См. Рисунок 35)

2.4.14 Процедура SendStrategyVictory.

- По приходу сигнала от модуля Main делается проверка: жива ли стратегия. Если да, то сигнал перенаправляется стратегии, если же нет, то ничего не делается. (См. Рисунок 37)

2.4.15 Процедура SendStrategyLose.

- По приходу сигнала от модуля Main делается проверка: жива ли стратегия. Если да, то сигнал перенаправляется стратегии, если же нет, то ничего не делается. (См. Рисунок 39)

2.4.16 Процедура SendDestroyShip

- По приходу сигнала от модуля Main делается проверка: жива ли стратегия. Если да, то сигнал перенаправляется стратегии, если же нет, то ничего не делается. (См. Рисунок 40)

2.5 Реализация модуля Main1 (блок стратегии).

Данный модуль отвечает за поведение стратегии. Отправление запросов с приказами симулятору и прием ответов от него. Описание реализации этого модуля приведено ниже, начиная непосредственно с описания процесса Main1.

2.5.1 Процесс Main1.

- Этот процесс является ключевым в модуле стратегии. Главная задача, решаемая этим процессом – синхронизация работы стратегии и симулятора. (См. Рисунок 109 - Рисунок 111)
- Типы данных, переменные и константы:
  - Константы:

Название константы	Тип	Значение	Описание
FirstQuaterX	Integer	3*MaxXCoordinate/4	Координата на оси абсцисс первой целевой точки.
FirstQuaterY	Integer	3*MaxYCoordinate/4	Координата на оси ординат первой целевой точки.
SecondQuaterX	Integer	3*MaxYCoordinate/4	Координата на оси абсцисс второй целевой точки.
SecondQuaterY	Integer	MaxYCoordinate/4	Координата на оси ординат второй целевой точки.
ThirdQuaterX	Integer	MaxYCoordinate/4	Координата на оси абсцисс третьей целевой точки.
ThirdQuaterY	Integer	MaxYCoordinate/4	Координата на оси ординат третьей целевой точки.
FourthQuaterX	Integer	MaxYCoordinate/4	Координата на оси абсцисс четвертой целевой точки.
FourthQuaterY	Integer	3*MaxYCoordinate/4	Координата на оси ординат четвертой целевой точки.

  - Перечислимые типы:

Название типа	Значения	Описание
tOrderType		Тип приказа кораблям:
	_RIGHT	поворот направо
	_LEFT	поворот налево
	_FIRE	выстрел
	_ACCEL	изменение скорости

  - Структуры данных:

Название типа	Поле	Тип поля	Описание
tFire			Структура для хранения информации о ведении огня:
	List	tStringShotsList	список точек выстрела
	Weapon	tNameWeaponry	тип оружия для стрельбы
tCommand			Структура для хранения приказов кораблям:
	OrderType	tOrderType	тип приказа
	isSet	Boolean	флаг выставления приказа
	Fire	tFire	структура для хранения информации о ведении огня
	Accel	tFactorAcceleration	параметр ускорения

  - Хэш-таблицы:

Название типа	Тип индекса	Тип значения	Описание
tArrayCommand	tShipId	tCommand	Массив структур для хранения приказов кораблям.

- Переменные:

Название переменной	Тип	Описание
WeaponryCharacter	tArrayWeaponryCharacter	Характеристики вооружения.
ShipsCharacter	tArrayShipsCharacter	Характеристики кораблей.
MyCenter	tCoordinates	Центр масс кораблей флота.
Tick	Integer	Такт симуляции.
NumMyShips	Integer	Число кораблей флота.
minSpeed	Integer	Максимальная скорость самого медленного корабля флота.
flagFight	Boolean	Признак битвы.
HeadingTarget	tCoordinates	Целевая точка.
ShipsCurrentState	tStringShipCurrentState	Структура с информацией о кораблях флота.
CommandToShip	tArrayCommand	Массив структур для хранения приказов кораблям.
StrategyID	tStartegyId	Идентификатор стратегии.
EnemyCenter	tCoordinates	Центр масс видимых вражеских кораблей.
NumVisibleShips	Integer	Число видимых вражеских кораблей.
EnemyShips	tStringEnemyShip	Структура с информацией о видимых вражеских кораблях.

В начальный момент стратегия находится в состоянии Idle и ожидает прихода сигнала InitDataComplete. Таким образом, пока все данные (характеристики кораблей, вооружения, карта) не проинициализированы и стратегии не выслан этот сигнал, она не начинает свою работу.

Как только сигнал InitDataComplete получен, стратегия экспортирует проинициализированные характеристики кораблей и вооружения и переходит в состояние InitComplete. В этом состоянии она ожидает прихода сигнала ShipCurrentState с информацией о текущем состоянии флота. Как только сигнал получен, стратегия инициализирует данные (описание процедуры UpdateShipParams приведено ниже) и переходит в состояние IdleVisShips. В этом состоянии стратегия ожидает либо прихода сигнала о видимых вражеских кораблях (VisibleShip), если они есть, либо сигнала о начале симуляции – StartSimulation.

Как только пришел сигнал о начале симуляции, стратегия начинает свою работу и реализовывает один такт симуляции, внутри которого осуществляется выдача приказов кораблям (процедура StrategyStep). По окончании работы стратегии она выдает сигнал FinishSimulation – сигнал о том, что передача приказов кораблям закончена. После этого стратегия попадает в состояние IdleRes.

В ответ на выданные команды симулятор выдает либо подтверждение команды (например, OrderSpeedResponse), либо сигнал о невозможности выполнить приказ (например, OrderTurnReject). Когда все отклики на приказы были получены, снова осуществляется ожидание прихода сигнала ShipCurrentState, и запускается обработка очередного такта симуляции.

В случае если из любого состояния процесса приходят сигналы StrategyVictory или StrategyLose, говорящие о том, что стратегия либо выиграла, либо проиграла, процесс Main1 завершает свою работу.

2.5.2 Процедура UpdateShipParams.

- Данная процедура используется для обновления глобальных характеристик флотов, используемых в стратегии, перед началом каждого такта симуляции: числа и центров масс своих и видимых вражеских кораблей (см. Рисунок 112).
- Типы данных, переменные и константы:
  - Переменные:

Название переменной	Тип	Описание
TempCenterX	Real	Промежуточное значение координаты по оси абсцисс центра масс кораблей флота.
TempCenterY	Real	Промежуточное значения координаты по оси ординат центра масс кораблей флота.

i	Integer	Счетчик цикла.

В данной процедуре перед началом нового такта симуляции выполняются следующие перевычисления внутренних переменных стратегии: обнуляется число видимых вражеских кораблей (NumVisibleShips), обнуляется центр масс видимых вражеских кораблей (EnemyCenter), вычисляется центр масс кораблей своего флота (MyCenter), вычисляется число кораблей флота(NumMyShips), вычисляется максимальная скорость самого медленного корабля во флоте. Вызов данной процедуры осуществляется непосредственно перед приёмом сигнала StartSimulation. Подсчёт центра масс флота происходит по формуле, приведенной в п. 2.4.1 документа [2].

2.5.3 Процедура EnemyCenter.

- Данная процедура вычисляет центр масс видимых вражеских кораблей на каждом такте симуляции, если видимые вражеские корабли есть (см. Рисунок 113).
- Типы данных, переменные и константы:
  - Переменные:

Название переменной	Тип	Описание
TempCenterX	Real	Промежуточное значение координаты по оси абсцисс центра масс кораблей флота.
TempCenterY	Real	Промежуточное значение координаты по оси ординат центра масс кораблей флота.
i	Integer	Счетчик цикла.

2.5.4 Процедура StrategyStep.

- Основная алгоритмическая процедура стратегии. Она реализует один такт управления флотом. В ней содержатся все основные используемые в стратегии алгоритмы. (См. Рисунок 114 - Рисунок 117)
- Типы данных, переменные и константы:
  - Перечислимые типы:

Название типа	Значения	Описание
tTurnDirection		Возможные направления движения кораблей:
	LEFT	налево
	RIGHT	направо
	STRAIGHT	прямо
tCollision		Тип коллизии:
	NO_COLLISION	нет коллизии
	LAND_COLLISION	посадка на мель
	ENEMY_SHIP_COLLISION	столкновение с вражеским кораблем
	MY_SHIP_COLLISION	столкновение с кораблем своего флота

  - Переменные:

Название переменной	Тип	Описание
ShipsCurrState	tStringShipCurrentState	Структура с информацией о кораблях флота.
index	Integer	Индекс корабля в структуре ShipsCurrState.
CollisionType	tCollision	Тип коллизии.

Для того чтобы определить стратегическое поведение флота, необходимо установить, есть ли в поле видимости стратегии вражеские корабли или их нет. Для этого в этой части процедуры проверяется количество видимых вражеских кораблей (NumVisibleShips). Если оно нулевое, то стратегия идет по пути реализации поведения кораблей в отсутствие вражеских кораблей. Иначе – реализуется поведение кораблей при наличии видимых противников.

Если враги имеются, то устанавливается вспомогательный флаг flagFight, и стратегия идет по пути атаки врагов. Если на предыдущем такте flagFight был установлен, а видимых противников нет, то такая ситуация

соответствует потере противника из поля видимости или реализации первого такта симуляции. Для поиска противника все корабли флота осуществляют объезд карты по часовой стрелке по квадрату, углы которого соответствуют центрам четвертей карты (целевые точки). Для нахождения начальной точки движения (HeadingTarget) определяется ближайшая к центру масс флота целевая точка.

В случае если центр масс флота достигает области целевой точки (см. п. 2.4.3 документа [2]), определяется новая целевая точка.

Вообще данный алгоритм является общим для поведения кораблей при наличии и отсутствии противника. Алгоритм представляет собой цикл, в котором перебираются последовательно все корабли флота (все элементы структуры ShipsCurrentState), и по состоянию корабля на текущий такт определяется, какие именно приказы необходимо ему выдать в соответствии с выбранной линией поведения.

В начальный момент для каждого корабля определяется, находится ли он на мели или нет. Для этого проверяется глубина карты в точке положения корабля. Если она меньше, чем проходная глубина для данного типа корабля, то корабль пытается съехать с мели (процедура OnLand). Если корабль не на мели, то корабль реализует действия по атаке противника (процедура EnemyFound).

После того, как команда кораблю установлена, для корабля определяется, приведет ли данная команда к какому-либо типу коллизии. Если коллизии не возникает (NO_COLLISION), то симулятору высылается сигнал о выдаче соответствующего приказа кораблю (процедура OutputOrder). Если же возникает какой-либо вид коллизии (столкновение с другим кораблем – ENEMY_SHIP_COLLISION, MY_SHIP_COLLISION, посадка на мель – LAND_COLLISION), то стратегия пытается разрешить коллизию в процедуре SolveCollision.

После перебора всех кораблей флота осуществляется выход из процедуры. Такой алгоритм позволяет выдать необходимые приказы (если они нужны) всем кораблям флота в соответствие с выбранной линией поведения флота.

Эта часть аналогична предыдущей за исключением того, что она соответствует поведению кораблей флота при отсутствии видимых вражеских кораблей. Соответственно вместо процедуры EnemyFound вызывается процедура EnemyNotFound.

2.5.5 Процедура SolveCollision.

- Данная процедура разрешает коллизии, возникающие при движении кораблей: посадку на мель, столкновение со своим или вражеским кораблем (см. Рисунок 118 и Рисунок 119).

Если тип коллизии – ENEMY_SHIP_COLLISION, то проверятся наличие вооружения у корабля. Если вооружения нет, то никакого приказа кораблю не отдается (поле isSet элемента массива CommandToShip для данного корабля устанавливается как false), и он идет «на таран».

Далее проверяется тип приказа для корабля. Если кораблю был установлен приказ на увеличение скорости, то приказ отменятся, и проверяется, будет ли в этом случае происходить коллизии. Если никаких коллизий не происходит, то приказ кораблю не выдается, и он продолжает плыть без изменения скорости.

Во всех остальных случаях кораблю сначала устанавливается приказ повернуть направо. Если это приводит к коллизии, устанавливается приказ поворота налево. Если попытка повернуть налево тоже приводит к коллизии, то кораблю отдается приказ затормозить с максимальным ускорением, и осуществляется выход из процедуры.

2.5.6 Процедура isCollision.

- Определяет, возникнет ли в процессе движения корабля какие-либо виды коллизии (см. Рисунок 120 и Рисунок 121).
- Типы данных, переменные и константы:
  - Переменные:

Название переменной	Тип	Описание
MyShipNextState	tShipCurrentSate	Состояние рассматриваемого в процедуре корабля флота на следующем такте симуляции.
OtherShipNextState	tShipCurrentSate	Состояние любого другого корабля флота на следующем такте симуляции.
NextPoint	tCoordinates	Вспомогательная переменная с координатами корабля на следующем такте симуляции.
NextHead	tHeading	Вспомогательная переменная с направлением корабля на

		следующем такте симуляции.
IncX	Integer	Инкремент скорости по оси абсцисс.
IncY	Integer	Инкремент скорости по оси ординат.
i	Integer	Счетчик цикла.
Deep	Integer	Глубина карты в точке положения корабля.
maxDeep	Integer	Максимально возможная глубина для прохода корабля.
flag	Boolean	Флаг, выставляющийся в случае езды корабля по диагонали.
MyShipType	tNameShip	Тип корабля.

Сначала по установленному кораблю приказу (или текущему состоянию корабля, если приказа кораблю не было отдано) определяется его состояние на следующем такте симуляции (процедура NextShipState).

После этого определяется, сядет ли корабль на мель через 2 такта симуляции. Для этого в каждой точке его движения проверяется глубина карты и сравнивается с проходной глубиной для соответствующего типа корабля. Если в какой-либо точке глубина меньше проходной, то процедура возвращает тип коллизии LAND_COLLISION.

Затем проверятся возможность столкновения с вражеским кораблем на следующем такте, если в поле видимости флота есть вражеские корабли. Для этого для каждого вражеского корабля определяется его положение на следующем такте симуляции и сравнивается с положением проверяемого в процедуре корабля. Если положения кораблей совпадают, процедура возвращает тип коллизии ENEMY_SHIP_COLLISION.

После этого проверяется возможность столкновения рассматриваемого корабля с кораблем своего флота на следующем такте симуляции. Аналогично столкновению с вражескими кораблями, происходит перебор в цикле всех кораблей своего флота, и для каждого корабля флота, кроме текущего, вычисляется положение на следующем такте и сравнивается с положением исследуемого корабля. Если положения совпадают, процедура возвращает тип коллизии MY_SHIP_COLLISION.

Если ни один их типов коллизий не происходит, то процедура возвращает тип коллизии NO_COLLISION.

2.5.7 Процедура NewHeadingChange.

По заданному состоянию корабля и цели его движения определяет, необходим ли поворот (и куда, если необходим) для движения к заданной цели (см. Рисунок 122 и Рисунок 123).

- Типы данных, переменные и константы:
  - Хэш-таблицы:

Название типа	Тип индекса	Тип значения	Описание
tCos	Integer	Real	Массив значений косинусов углов между векторами.

  - Переменные:

Название переменной	Тип	Описание
OurShCurrState	tShipCurrentSate	Состояние рассматриваемого в процедуре корабля флота на текущем такте симуляции.
TargetPoint	tCoordinates	Цель для движения корабля.
ReturnDirect	tTurnDirection	Возвращаемое процедурой значение направления.
NextHding	tHeading	Новое направление при повороте корабля.
RightPoint	tCoordinates	Точка, где окажется корабль на новом такте симуляции при повороте направо.
LeftPoint	tCoordinates	Точка, где окажется корабль на новом такте симуляции при повороте налево.
StraightPoint	tCoordinates	Точка, где окажется корабль на новом такте симуляции при движении без изменения направления.
i	Integer	Счетчик цикла.
indx	Integer	Счетчик цикла.
min	Real	Вспомогательная переменная.
Cos	tCos	Массив значений косинусов углов между векторами.
CurrSpeed	tSCMaxSpeed	Текущая скорость корабля.

Для определения направления движения к цели используется процедура CosBetweenVectors, которая использует формулу вычисления угла между векторами. В качестве первого вектора используется вектор, направленный от точки положения корабля до цели, а в качестве второго – вектор, направленный от точки положения корабля до точки, в которой окажется корабль при соответствующем повороте (налево, направо или при движении вперед). При этом в случае, если текущая скорость корабля – нулевая, для верного определения угла поворота для расчетов принимается значение скорости, равное 2, что соответствует проходу корабля на одну клетку.

В процедуре определяются три значения косинусов, и из них выбирается максимальное значение, что соответствует минимальному углу к цели. Процедура возвращает значение поворота (LEFT, RIGHT, STRAIGHT), соответствующее минимальному углу поворота.

2.5.8 Процедура CosBetweenVectors.

- Вычисляет косинус угла между двумя векторами (см. Рисунок 124).
- Типы данных, переменные и константы:
  - Переменные:

Название переменной	Тип	Описание
StartPoint	tCoordinates	Координаты начала векторов.
EndPoint1	tCoordinates	Координаты конца первого вектора.
EndPoint2	tCoordinates	Координаты конца второго вектора.

В начале процедуры осуществляется проверка на нулевую длину какого-либо из векторов. Если вектора с нулевой длиной существуют (начальная и конечная координаты вектора совпадают), то в этом направлении кораблю ехать не нужно, так как его положение не изменится. В этом случае возвращается минимально возможное значение косинуса: -2. Затем осуществляется проверка на коллинеарность векторов (вектора параллельны и сонаправлены). Если условие коллинеарности выполняется (вектора параллельны), то возвращается максимально возможное значение косинуса: 2. Во всех других случаях значение косинуса вычисляется по формуле, представленной в п. 2.4.5 документа [2].

2.5.9 Процедура NextDirection.

- По заданному направлению движения корабля и повороту (LEFT, RIGHT) определяется значение нового направления движения (см. Рисунок 125).
- Типы данных, переменные и константы:
  - Переменные:

Название переменной	Тип	Описание
CurrHding	tHeading	Направление корабля.
Dir	tHeading	Значение поворота.
NewHeading	tHeading	Новое направление корабля.
Numbr	Integer	Вспомогательная переменная.

2.5.10 Процедура NextPosition.

- Процедура возвращает позицию корабля на следующем такте симуляции по его текущим значениям координат, направления, скорости и типу корабля (см. Рисунок 126).
- Типы данных, переменные и константы:
  - Переменные:

Название переменной	Тип	Описание
CurrPos	tCoordinates	Текущая позиция корабля.
Direct	tHeading	Текущее направление корабля.
Speed	tSCMaxSpeed	Текущая скорость корабля.
NameShip	tNameShip	Тип корабля.
NextPos	tCoordinates	Позиция корабля на следующем такте симуляции.

IncX	Integer	Инкремент скорости по оси абсцисс.
IncY	Integer	Инкремент скорости по оси ординат.
i	Integer	Счетчик цикла.
Deep	Integer	Глубина карты в точке положения корабля.
maxDeep	Integer	Максимально возможная глубина для прохода корабля.
flag	Boolean	Флаг, выставляющийся в случае езды корабля по диагонали.
onLand	Boolean	Флаг, выставляющийся в случае, если корабль находится на мели.

Алгоритм поиска позиции корабля заключается в следующем. Сначала по направлению корабля определяется величина инкрементирования координат корабля по осям X и Y. В случае если движение корабля осуществляется по диагонали карты, вспомогательная переменная flag (проинициализированная в 1) принимает значение $\sqrt{2}$ для осуществления правильного вычисления позиции. Затем в цикле по $i \in (1, fix(\frac{Speed}{flag}))$, что соответствует максимальному числу клеток, на которое должен продвинуться корабль, к координатам корабля прибавляются соответствующие значения инкрементов по осям. В случае выхода за границы карты или посадки на мель корабль не меняет положения, поэтому осуществляется выход из процедуры с текущими на момент выхода координатами корабля. Если корабль при входе в процедуру уже находится на мели, то инкрементирование продолжается. Это было сделано с целью определения угла поворота корабля для съезда с мели.

2.5.11 Процедура NextShipState.

- Данная процедура определяет состояние корабля (скорость, направление и координаты) на следующем такте симуляции в зависимости от установленных кораблю команд. В качестве входных параметров используется индекс корабля в структуре высылаемых симулятором кораблей ShipsCurrentState. (См. Рисунок 127)
- Типы данных, переменные и константы:
  - Переменные:

Название переменной	Тип	Описание
ShipIndex	Integer	Индекс корабля в структуре ShipsCurrState.
ShipNextState	tShipCurrentSate	Состояние корабля на следующем такте симуляции.
OldSpeed	tSCMaxSpeed	Текущая скорость корабля.
OldHeading	tHeading	Текущее направление корабля.
OldCoords	tCoordinates	Текущая позиция корабля.
TypeShip	tNameShip	Тип корабля.
TempSpeed	Integer	Вспомогательная переменная.

Если команда кораблю не была отдана или команда была о ведении огня (тип приказа - _FIRE), то его скорость и направление остаются прежними, и позиция вычисляется на основе текущих значений скорости и направления с помощью процедуры NextPosition. Если кораблю был отдан какой-либо другой приказ (_ACCEL, _LEFT, _RIGHT), то по типу приказа определятся будущее состояние корабля: в случае изменения скорости корабля (тип приказа - _ACCEL) его направление остается прежним, скорость пересчитывается по формуле, приведенной в п. 2.4.6 документа [2]. Новые координаты вычисляются на основании текущего направления и новой скорости. В случае поворота (тип приказа - _LEFT или _RIGHT) скорость корабля остается прежней, меняется его направление, и новые координаты вычисляются на основании нового направления.

2.5.12 Процедура EnemyNotFound.

- Данная процедура вызывается из процедуры StrategyStep при отсутствии в области видимости кораблей своего флота хотя бы одного вражеского корабля (см. Рисунок 128).
- Типы данных, переменные и константы:
  - Переменные:

Название переменной	Тип	Описание
ShCurrSt	tShipCurrentSate	Состояние рассматриваемого в процедуре корабля флота

		на текущем такте симуляции.
k	Integer	Индекс корабля в структуре ShipsCurrState.
ChangeDirection	tTurnDirection	Изменение направления корабля при движении к цели.
accel	Integer	Вспомогательная переменная.
ShipName	tNameShip	Тип корабля.
MaxAccel	tSCMaxAcceleration	Максимально возможное ускорение корабля.
factorAccel	tFactorAcceleration	Параметр в приказе об ускорении.

В начале осуществляется коррекция направления корабля для движения к целевой точке с помощью вызова процедуры NewHeadingChange. В случае если направление корабля необходимо скорректировать (процедура NewHeadingChange возвращает направление поворота LEFT или RIGHT), полю OrderType структуры CommandToShip присваивается соответственно значение _LEFT или _RIGHT (устанавливается приказ на поворот).

В случае неизменности курса корабля ему необходимо двигаться в поисках противника со средней скоростью (minSpeed), соответствующей максимальной скорости самого медленного корабля флота. Следовательно, если скорость корабля равна средней скорости флота, приказов ему не отдается. Если скорость корабля отличается от средней, то ему отдается приказ _ACCEL.

2.5.13 Процедура EnemyFound.

- Описывает алгоритм поведение корабля флота при наличии видимых вражеских кораблей (см. Рисунок 129 - Рисунок 132).
- Типы данных, переменные и константы:
  - Структуры данных:

Название типа	Поле	Тип поля	Описание
tShootWeaponry			Структура для хранения оружия атаки:
	TypeWeaponry	tNameWeaponry	тип оружия
	isWeaponry	Boolean	флаг наличия оружия для атаки
tShootList			Структура точек выстрела:
	SortParam		параметр для критерия обстрела
	Number	Integer	номер корабля в структуре с кораблями (ShipsCurrState или EnemyShips)

  - Списки:

Название типа	Тип компонент	Описание
tStringShootList	tShootList	Последовательность точек выстрелов.

  - Переменные:

Название переменной	Тип	Описание
ShCurrSt	tShipCurrentSate	Состояние рассматриваемого в процедуре корабля флота на текущем такте симуляции.
k	Integer	Индекс корабля в структуре ShipsCurrState.
NumPointsToShoot	Integer	Число точек выстрела.
i	Integer	Счетчик цикла.
j	Integer	Счетчик цикла.
DistBetweenShips	Integer	Расстояние между кораблями.
NextHeading	tTurnDirection	Коррекция направления движения к выбранной цели.
CanShootWeaponry	tShootWeaponry	Структура для хранения оружия атаки.
BestWeaponry	tNameWeaponry	Оружие для атаки.
ShootPoint	tShootList	Структура точек выстрела.
Temp	tShootList	Вспомогательная переменная.
ShootList	tStringShootList	Список точек выстрелов для корабля.
PointToOrder	tCoordinates	Очередная точка выстрела для приказа о ведении огня.
DestroyFlag	Boolean	Флаг, выставляющийся в случае наличия поврежденных

		кораблей во флоте.

Процедура реализует поведение кораблей двух групп: REPAIR_BOAT и всех остальных типов кораблей.

Для корабля REPAIR_BOAT в первую очередь определяется, есть ли среди кораблей флота те, чей ресурс меньше максимального. Если такие корабли есть и у REPAIR_BOAT есть вооружение, то реализуется процедура активации REPAIR_BOAT (процедура ActivateREPAIR). Если у корабля REPAIR_BOAT нет вооружения или во флоте нет поврежденных кораблей, то проверяется, состоит ли флот только из кораблей REPAIR_BOAT. Если во флоте нет кораблей другого типа, то корабль REPAIR_BOAT плывет к центру масс видимых вражеских кораблей «на таран». Если же во флоте есть корабли других типов, то корабль REPAIR_BOAT плывет к центру масс кораблей своего флота.

Для всех остальных типов кораблей выбирается оружие для атаки (процедура CanShootEnemyCenter). Если такое оружие не было найдено, корабль плывет к центру масс видимых вражеских кораблей. Если оружие для атаки было найдено, то: определяется максимальное количество точек для выстрела за один такт как максимум из числа установок и числа боеприпасов, составляется список кораблей, которые может поразить выбранное оружие. Список сортируется по увеличению расстояния от корабля, который собирается производить обстрел. Определяется количество кораблей (величина NumPointsToShoot), в которые нужно выстрелить, как максимум из максимального количества точек для выстрела и размера списка кораблей для атаки. Для корабля устанавливается команда _FIRE со списком точек выстрела. Точки выстрела представляют собой положения на следующем такте симуляции первых NumPointsToShoot кораблей из отсортированного списка кораблей для атаки.

2.5.14 Процедура ActivateREPAIR.

- Данная процедура описывает поведение ремонтного корабля в ситуации, когда в поле видимости стратегии имеются вражеские корабли (см. Рисунок 133 и Рисунок 134).
- Типы данных, переменные и константы:
  - Переменные:

Название переменной	Тип	Описание
NextShipSt	tShipCurrentSate	Состояние корабля на следующем такте симуляции.

Сначала происходит сортировка кораблей своего флота по ресурсу в сторону его увеличения (в случае присутствия кораблей с ресурсом, отличающимся от максимального). Далее происходит проверка на движение корабля: если он двигается с ненулевой скоростью, то для того, чтобы выстрелить, ему надо остановиться. Именно для этой цели кораблю присваивается ускорение, соответствующее максимальному замедлению. Затем составляется список для ремонта, исходя из количества ремонтных установок на корабле, после чего осуществляется непосредственно само лечение. В случае отсутствия во флоте «раненых» кораблей, ремонтнику отдаётся приказ двигаться за своим флотом. Подробно алгоритм выглядит следующим образом: составляется список кораблей, чей ресурс меньше максимального, и дальность вооружения достаточна для починки этих кораблей. Если список получается пуст (достаточно близких кораблей нет), то во всем флоте ищется самый слабый корабль, и REPAIR_BOAT плывет к нему. Если список не пуст, то корабль REPAIR_BOAT останавливается, список кораблей сортируется по возрастанию отношения $\frac{\mathrm{Re}\,source}{Max\,\mathrm{Re}\,source}$. Затем определяется максимальное количество точек для выстрела за один такт как максимум из числа установок и числа боеприпасов. После этого определяется количество кораблей (величина NumPointsToShoot), в которые нужно выстрелить, как максимум из максимального количества точек для выстрела и размера списка поврежденных кораблей. И, наконец, для корабля REPAIR_BOAT устанавливается команда _FIRE со списком точек выстрела. Точки выстрела представляют собой положения на следующем такте симуляции первых NumPointsToShoot кораблей из отсортированного списка поврежденных кораблей.

2.5.15 Процедура CanShootEnemyCenter.

- Данная процедура осуществляет выбор оружия для поражения врага при атаке (см. Рисунок 135).
- Типы данных, переменные и константы:
  - Переменные:

Название переменной	Тип	Описание
Weapon	tShootWeaponry	Возвращаемая структура с выбранным оружием для атаки и возможностью атаковать
Dist	Integer	Расстояние между кораблями

Сначала происходит проверка на наличие у корабля оружия с максимальной мощностью. Если оно имеется у корабля (это MISSILE), выбирается для стрельбы именно оно. В случае его отсутствия выбирается оружие с меньшей мощностью (TORPEDO). В случае отсутствия торпед, выбираются снаряды (SHELL). Выбор конкретного оружия происходит только в том случае, если дальность полёта этого типа оружия не меньше расстояния между нашим кораблём и тем, который хотим поразить.

2.5.16 Процедура SortShootList.

- Данная процедура сортирует список точек выстрелов кораблей по полю SortParam структуры ShootList в порядке возрастания (см. Рисунок 136).

2.5.17 Процедура OutputOrder.

- Выдача сигналов симулятору для управления поведением кораблей на очередном такте симуляции (см. Рисунок 137).

Так как эта процедура осуществляет выдачу приказов кораблям, то в начальный момент происходит проверка поля isSet структуры CommandToShip для каждого корабля на true (проверяется, была ли для корабля установлена команда). В случае совпадения (приказ был установлен кораблю) осуществляется дальнейший анализ с проверкой типа команды (повернуть, ускориться или выстрелить), и выдачей команды симулятору посредством одного из сигналов OrderTurnRequest, OrderSpeedRequest или OrderFireRequest с соответствующими параметрами.

2.5.18 Процедура FindStartPoint.

- В данной процедуре происходит вычисление начальной точки, к которой начнут движение корабли флота при отсутствии противника (см. Рисунок 138).
- Типы данных, переменные и константы:
  - Хэш-таблицы:

Название типа	Тип индекса	Тип значения	Описание
Dist	Integer	Integer	Массив расстояний.

  - Переменные:

Название переменной	Тип	Описание
d	Dist	Массив расстояний между центром масс флота и каждой из целевых точек.
count	Integer	Вспомогательная переменная.
i	Integer	Счетчик цикла.
min	Integer	Вспомогательная переменная.

Выбор происходит между 4-мя точками, являющимися центрами четырёх четвертей карты. Оценивается расстояние от центра масс флота до одной из этих точек и выбирается минимальное. Точка, соответствующая минимальному расстоянию, указывается в качестве целевой точки движения кораблей (HeadingTarget).

2.5.19 Процедура FindHeadingTarget.

- В этой процедуре происходит выбор очередной целевой точки для движения кораблей флота из четырёх, которые являются центрами четырёх четвертей, на которые разбита карта. Движение от одной точке к другой происходит по часовой стрелке, соответственно этому перебираются целевые точки. (См. Рисунок 139)

2.5.20 Процедура OnLand.

- Процедура вызывается в случае наезда кораблём на мель. В ней реализована попытка съезда корабля с суши. (См. Рисунок 140)

- Типы данных, переменные и константы:
  - Переменные:

Название переменной	Тип	Описание
MyLandShip	tShipCurrentSate	Состояние рассматриваемого в процедуре корабля флота на текущем такте симуляции (севшего на мель).
k	Integer	Индекс корабля в структуре ShipsCurrState.
i	Integer	Счетчик цикла.
j	Integer	Счетчик цикла.
PathCoord	tCoordinates	Точка, глубина в которой достаточна для прохода корабля (точка съезда).
Direct	tTurnDirection	Коррекция направлен я корабля к точке съезда с мели..
flagPath	Boolean	Флаг установлен, если корабль может съехать с мели (найдена точка съезда).
factorAccel	Real	Параметр в приказе об ускорении.

Для съезда корабля проверяются все близлежащие (на расстоянии одной клетки) от корабля точки на необходимую глубину. Если глубина в какой-то соседней точке превышает минимальную глубину для данного типа корабля (flagPath выставлен), то кораблю устанавливается приказ съехать на соседнюю клетку.

2.6 Описание реализации внешних функций.

В данном разделе приведено описание типов данных, переменных и констант, а также описание реализации внешних функций (см. Рисунок 141).

2.6.1 Функция getDistance.

- Возвращает расстояние между двумя точками на карте (см. Рисунок 142).

2.6.2 Функция sdl_sqrt.

- Вычисляет квадратный корень из числа (см. Рисунок 143).

2.6.3 Функция getHeading.

- Возвращает случайным образом выбранное направление движения корабля (см. Рисунок 144).

2.7 Описание реализации функций для работы с картой.

- В данном разделе описаны типы данных, переменные и константы, а так же функции, необходимые для работы с картой (см. Рисунок 145).
- Типы данных, переменные и константы:
  - Константы:

Название константы	Тип	Значение	Описание
MAX_FLEETS	Integer	8	Максимальное количество стратегий.

  - Структуры данных:

Название типа	Поле	Тип поля	Описание
stMap			Структура для хранения карты:
	MaxX	int	количество клеток по оси абсцисс
	MaxX	int	количество клеток по оси ординат
	MapVal	char *	глубины для каждой клетки карты

  - Переменные:

Название переменной	Тип	Описание
Map	stMap *	Карта.
MapPath	const char *	Путь к карте в иерархии проекта.
FleetCoords[MAX_FLEETS][2]	int	Координаты размещения флотов для каждой стратегии.
MapReaded	int	Флаг о прочтении карты.
CoordsCalculated	int	Флаг о вычислении корординат.

2.7.1 Функция getPath.

- Возвращает указатель на строку, содержащую путь к карте (см. Рисунок 146).

2.7.2 Функция readMap.

- Читает структуру карты из файла и заполняет переменную Map (см. Рисунок 147).

2.7.3 Функция checkMap.

- Проверяет, была ли уже прочитана карта (см. Рисунок 148).

2.7.4 Функция getDeepXY.

- Возвращает значение глубины в точке с переданными ей координатами (см. Рисунок 149).

2.7.5 Функция generatePoints.

- Вычисляет координаты точек на карте для каждой стратегии, вокруг которых возможно разместить флот, причем ни один из кораблей не будет сидеть на мели (см. Рисунок 150).

2.7.6 Функция getFleetPointX.

- По преданному идентификатору стратегии возвращает координату точки по оси абсцисс, вокруг которой можно разместить весь флот (см. Рисунок 151).

2.7.7 Функция getFleetPointY.

- По преданному идентификатору стратегии возвращает координату точки по оси абсцисс, вокруг которой можно разместить весь флот (см. Рисунок 152).

2.7.8 Функция FreeAll.

- Освобождает все ресурсы, занятые под карту (см. Рисунок 153).

Приложение А -- Код проекта «BattleShips» на уровне системы и модулей.

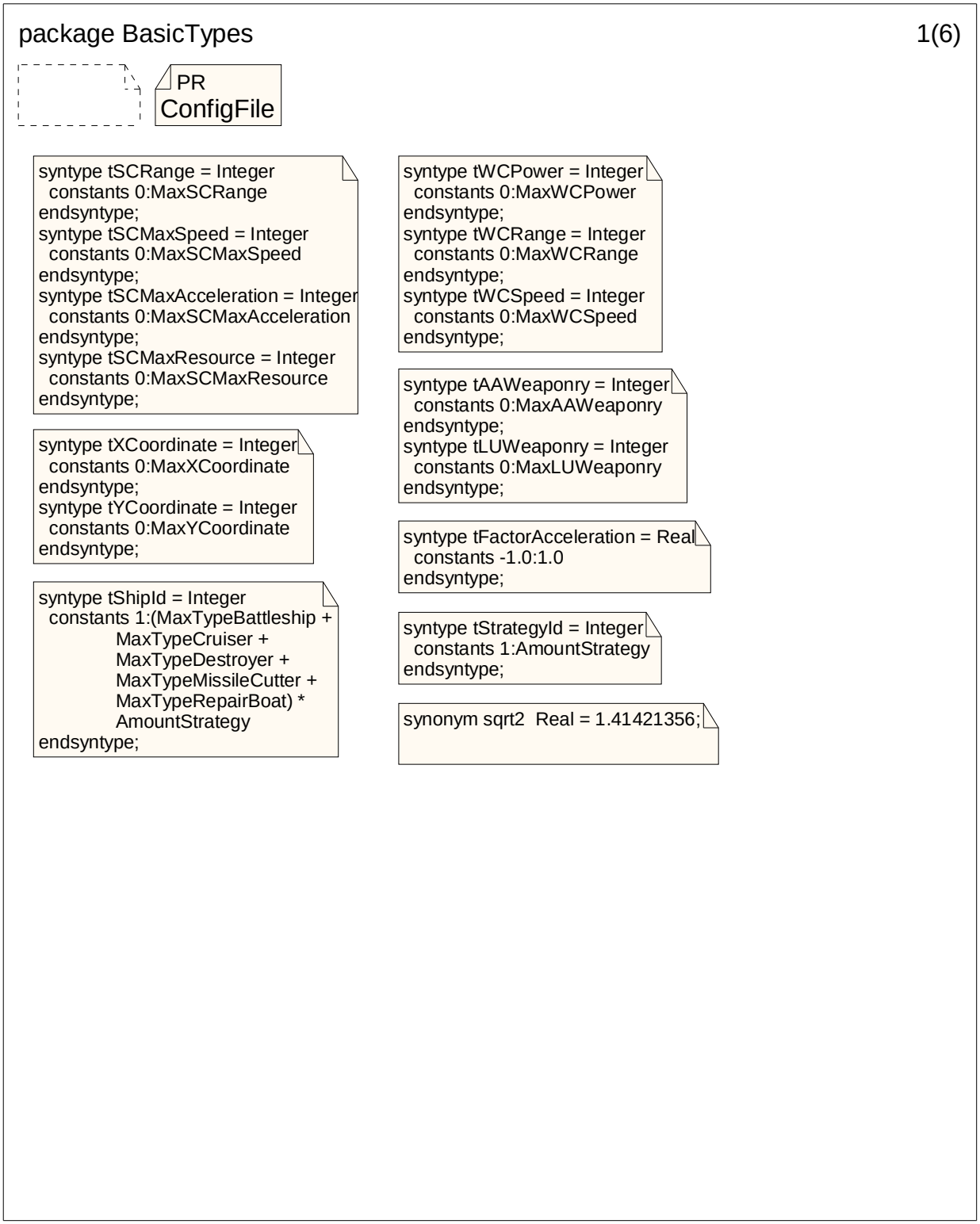

Рисунок 1. Описание типов данных, переменных и констант (1).

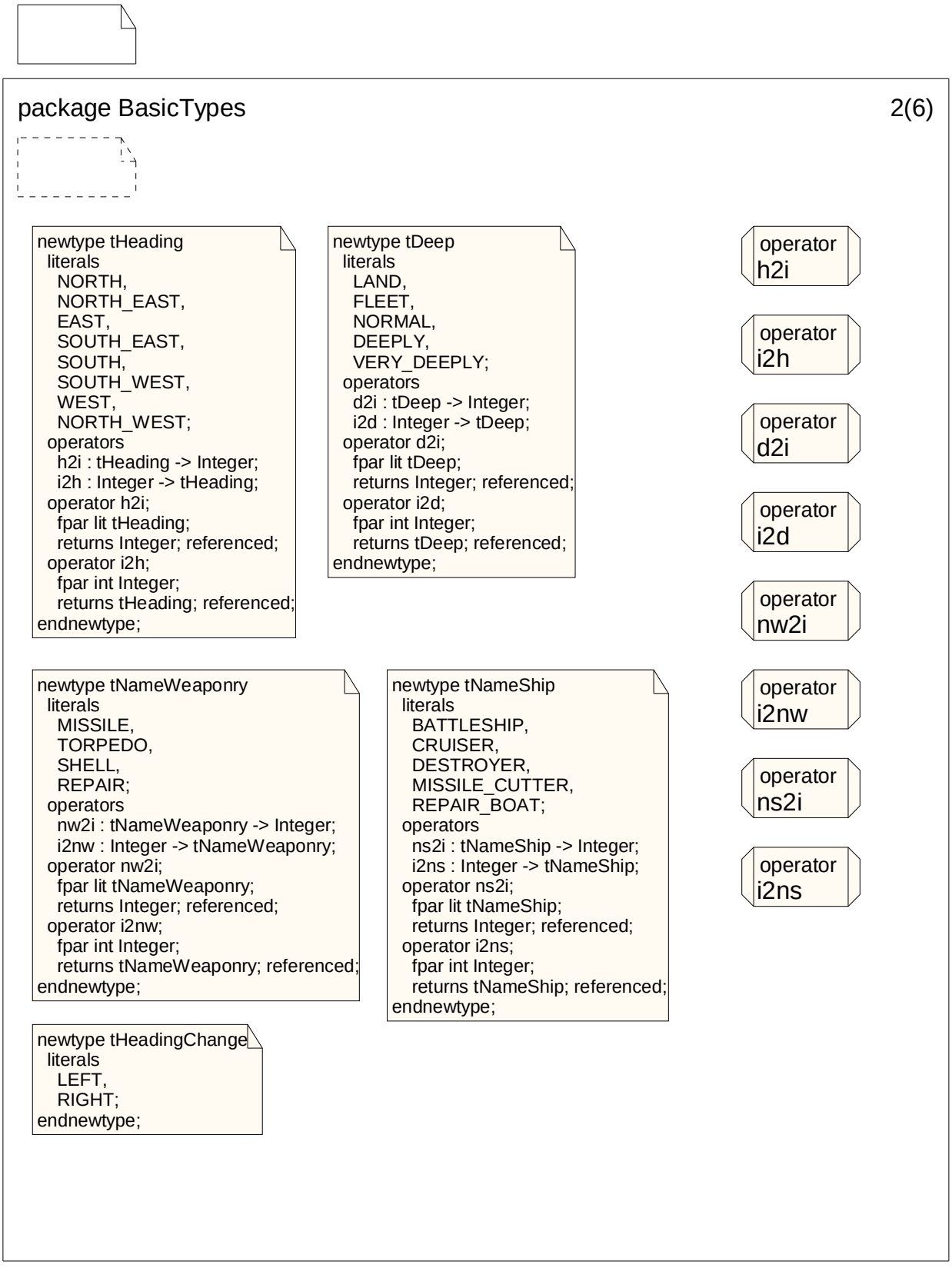

Рисунок 2. Описание типов данных, переменных и констант (2).

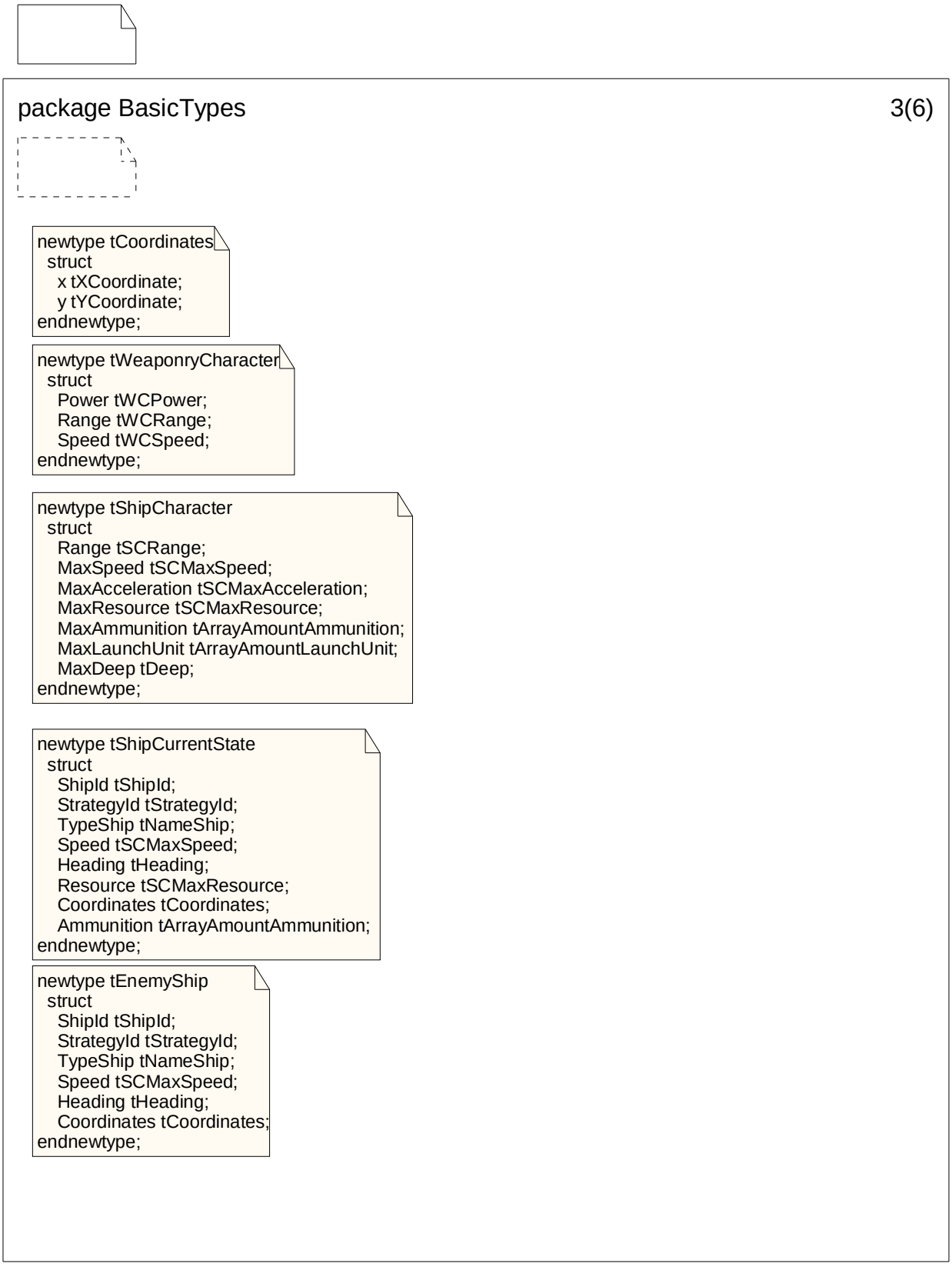

Рисунок 3. Описание типов данных, переменных и констант (3).

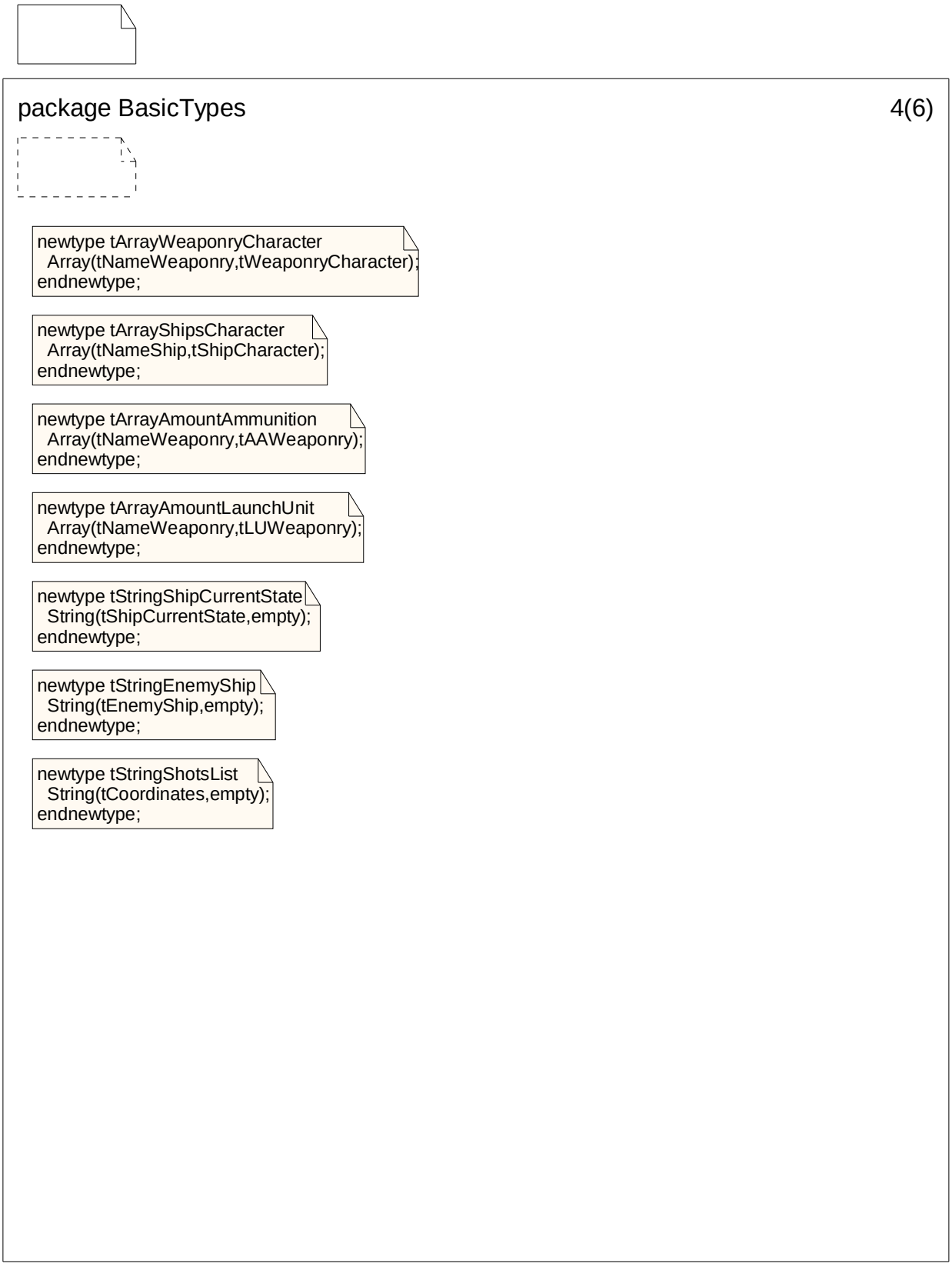

Рисунок 4. Описание типов данных, переменных и констант (4).

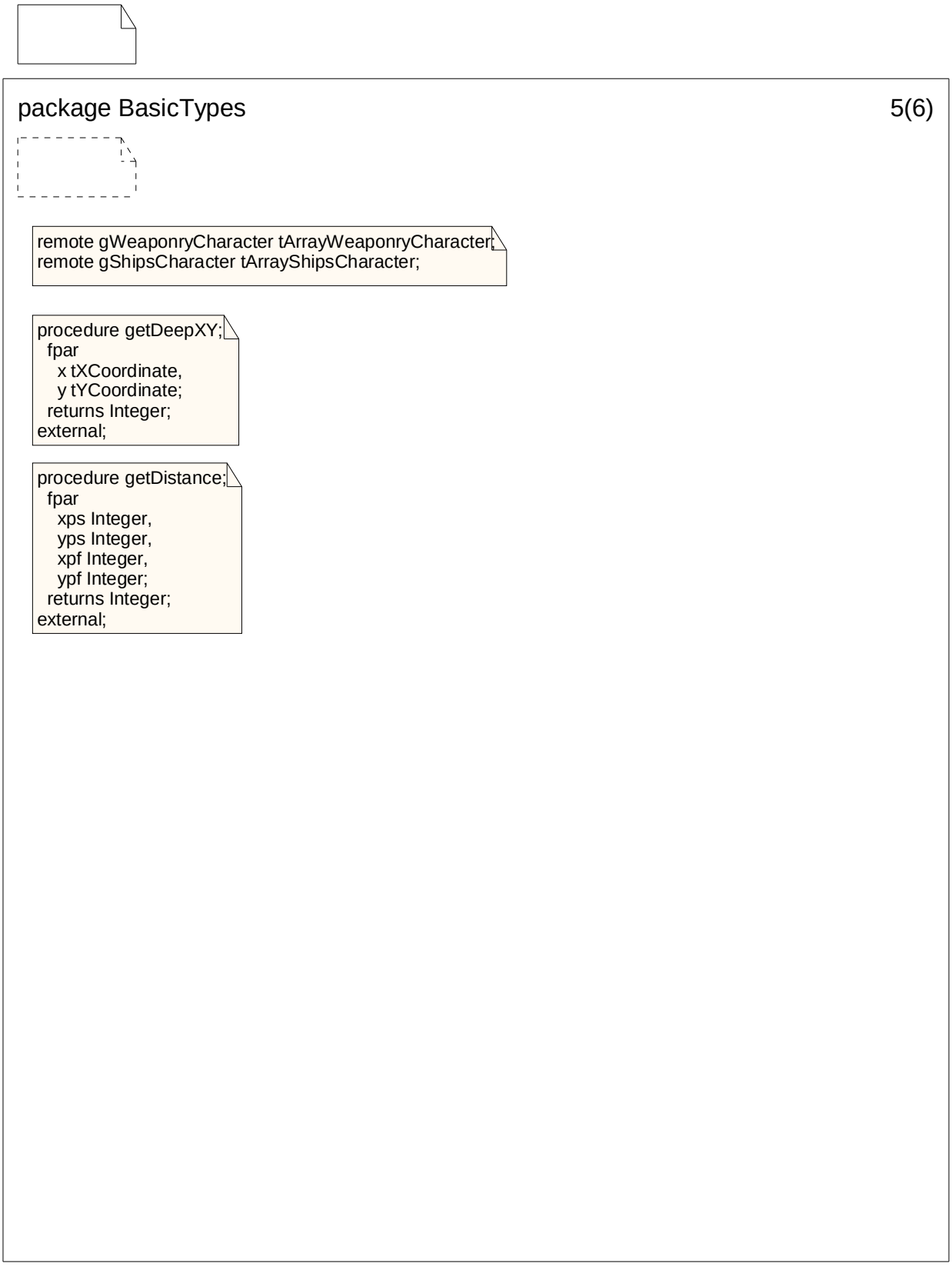

Рисунок 5. Описание типов данных, переменных и констант (5).

Рисунок 6. Описание типов данных, переменных и констант (6).

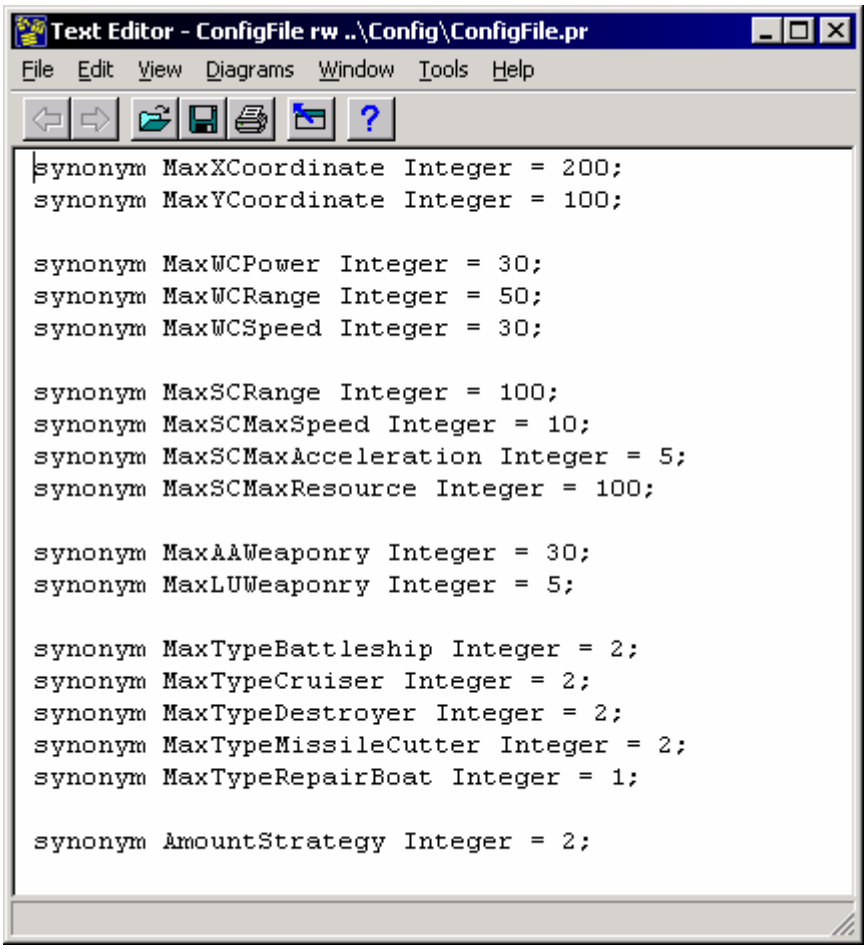

Рисунок 7. Конфигурационный файл (1).

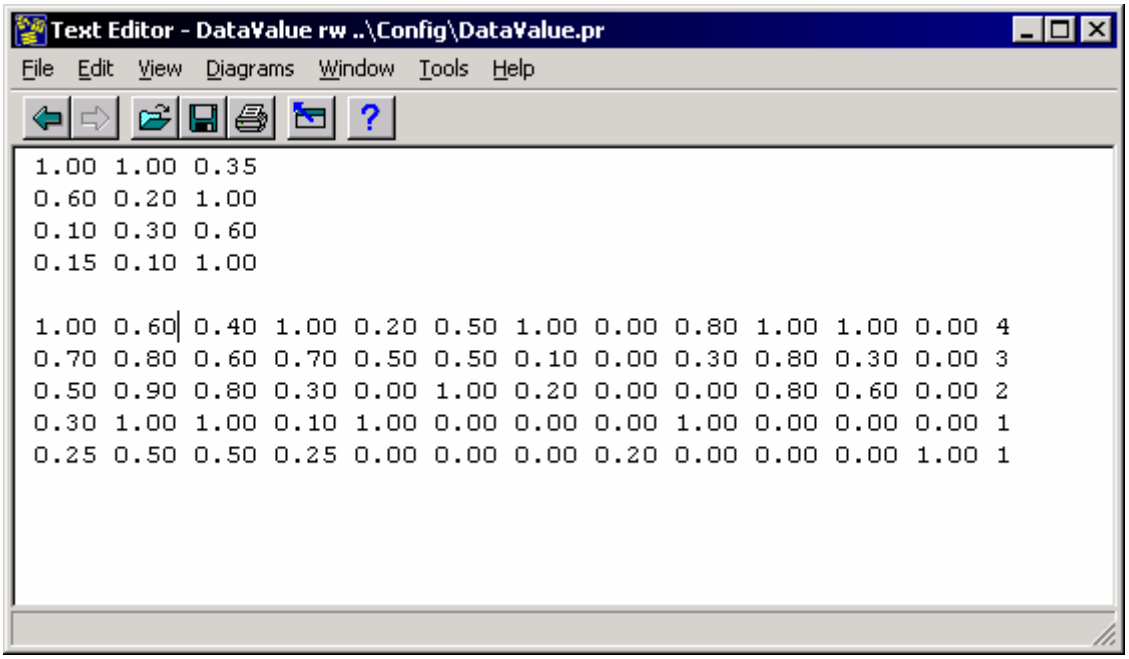

Рисунок 8. Конфигурационный файл (2).

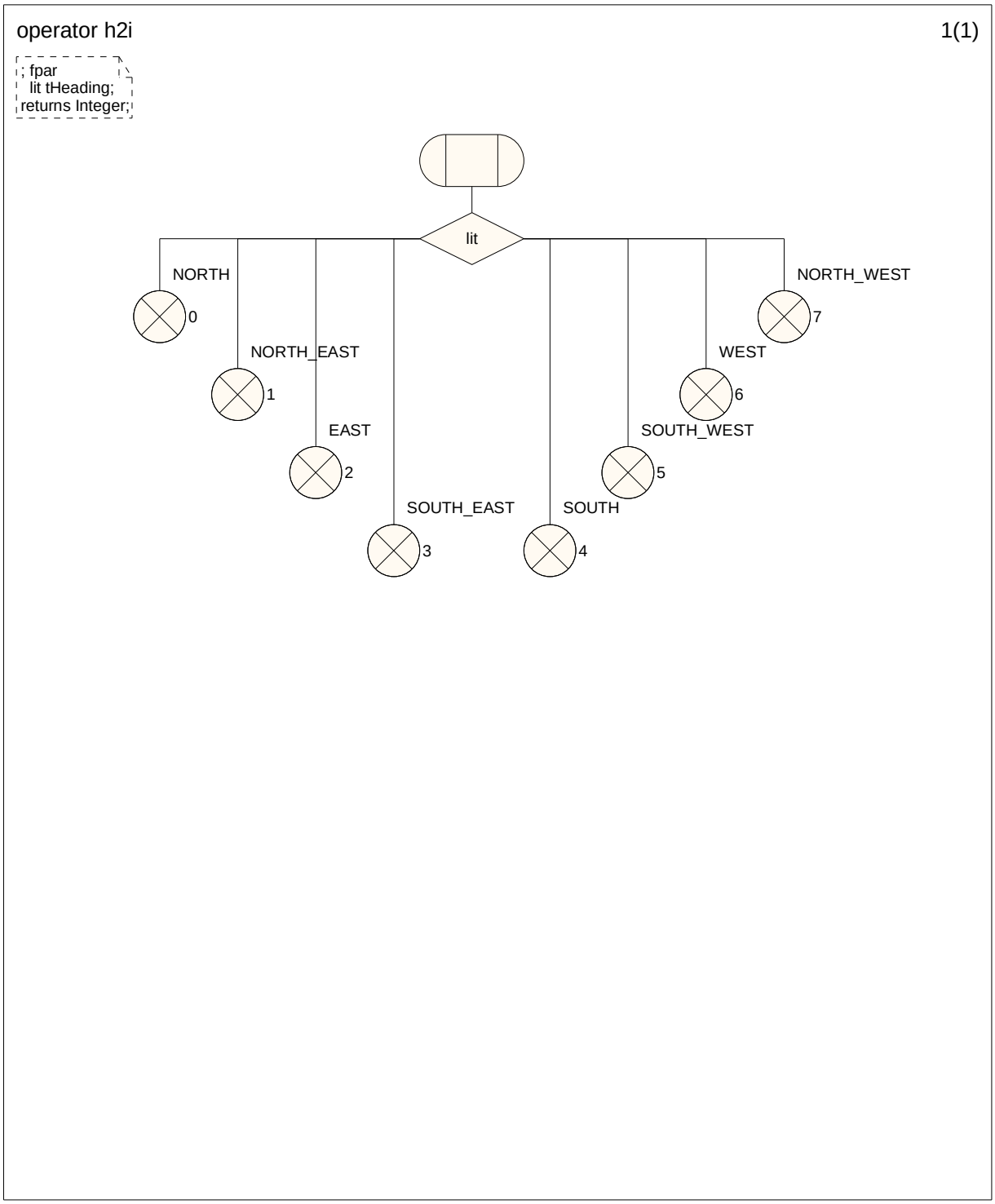

Рисунок 9. Оператор h2i.

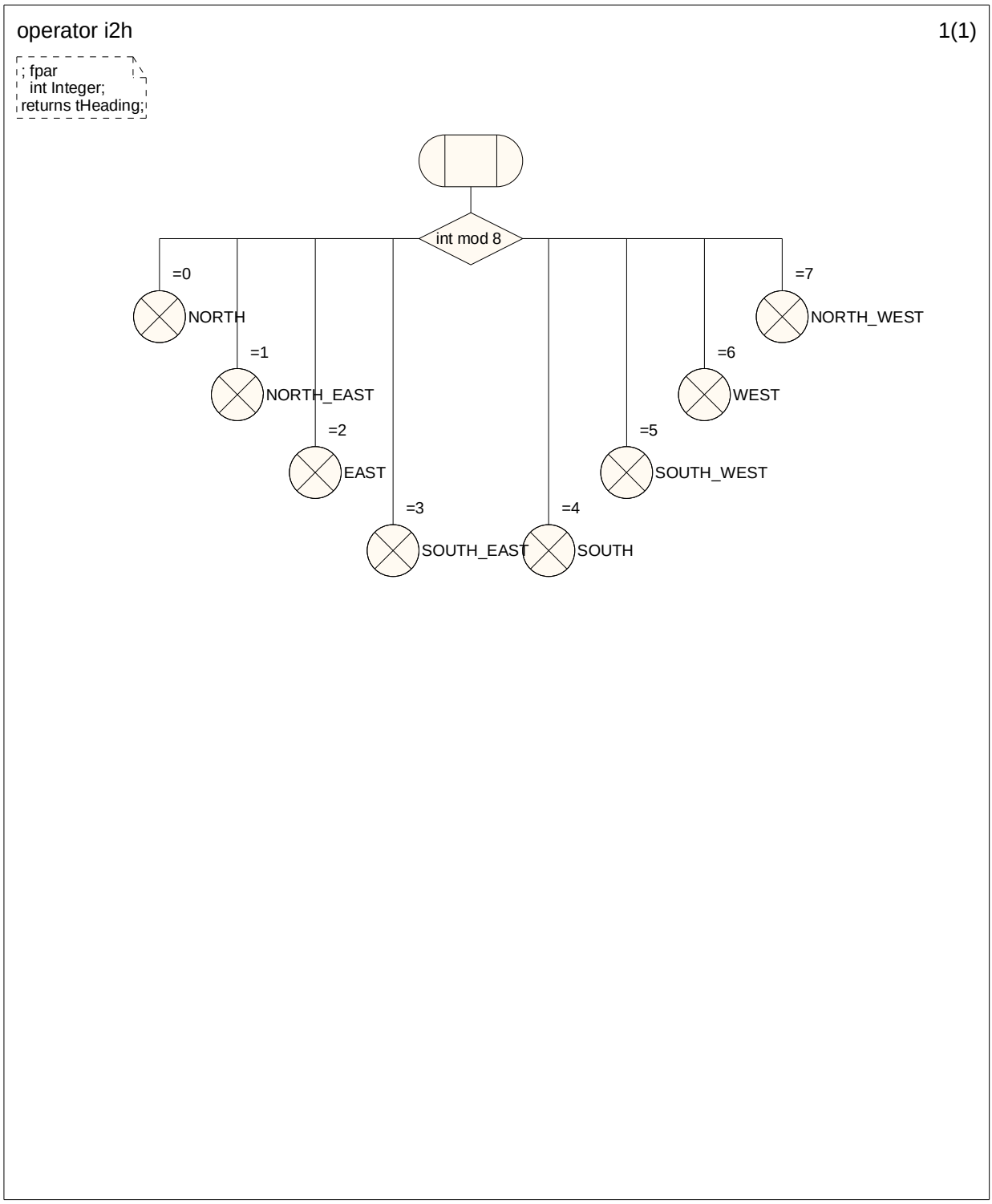

Рисунок 10. Оператор i2h.

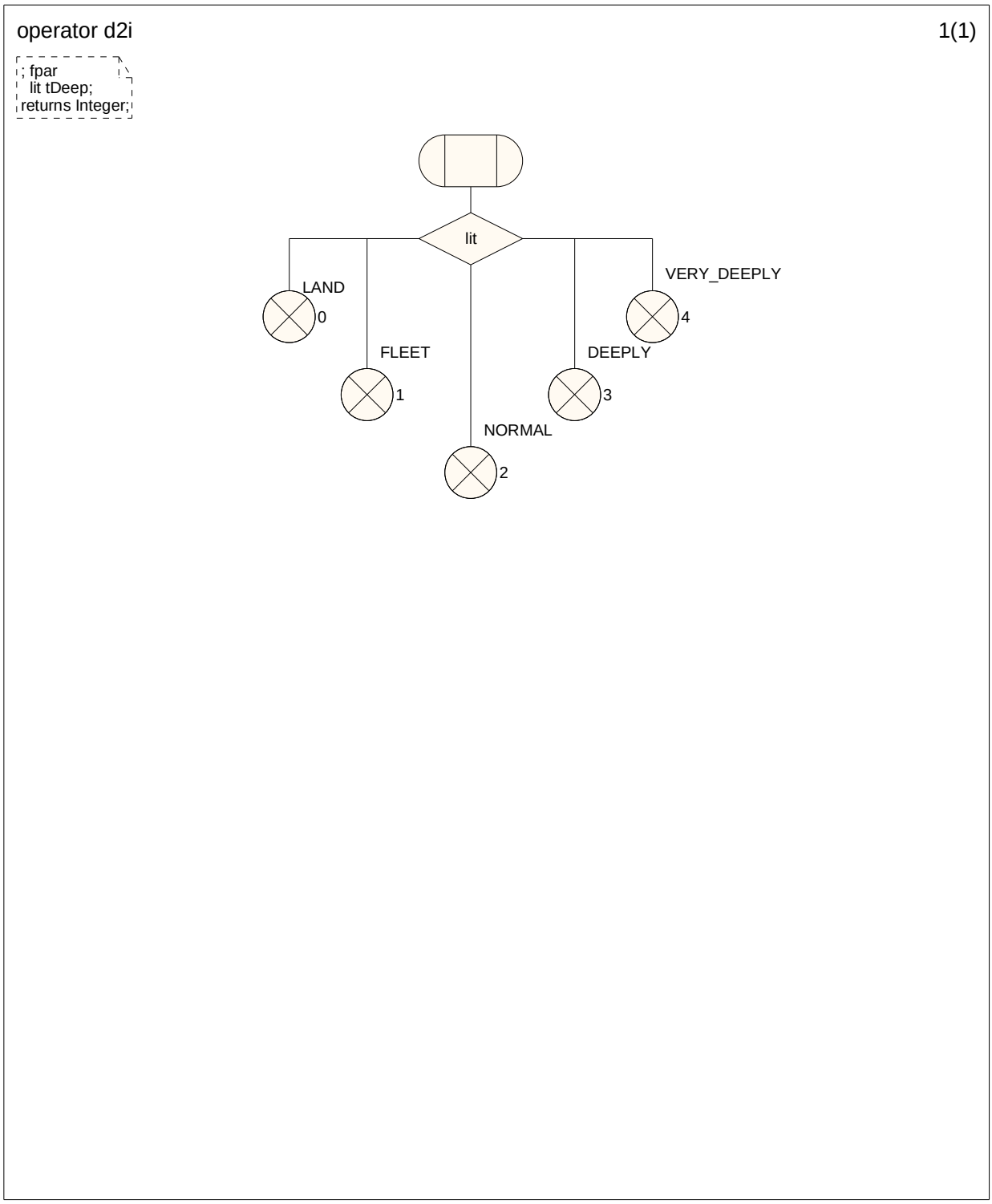

Рисунок 11. Оператор d2i.

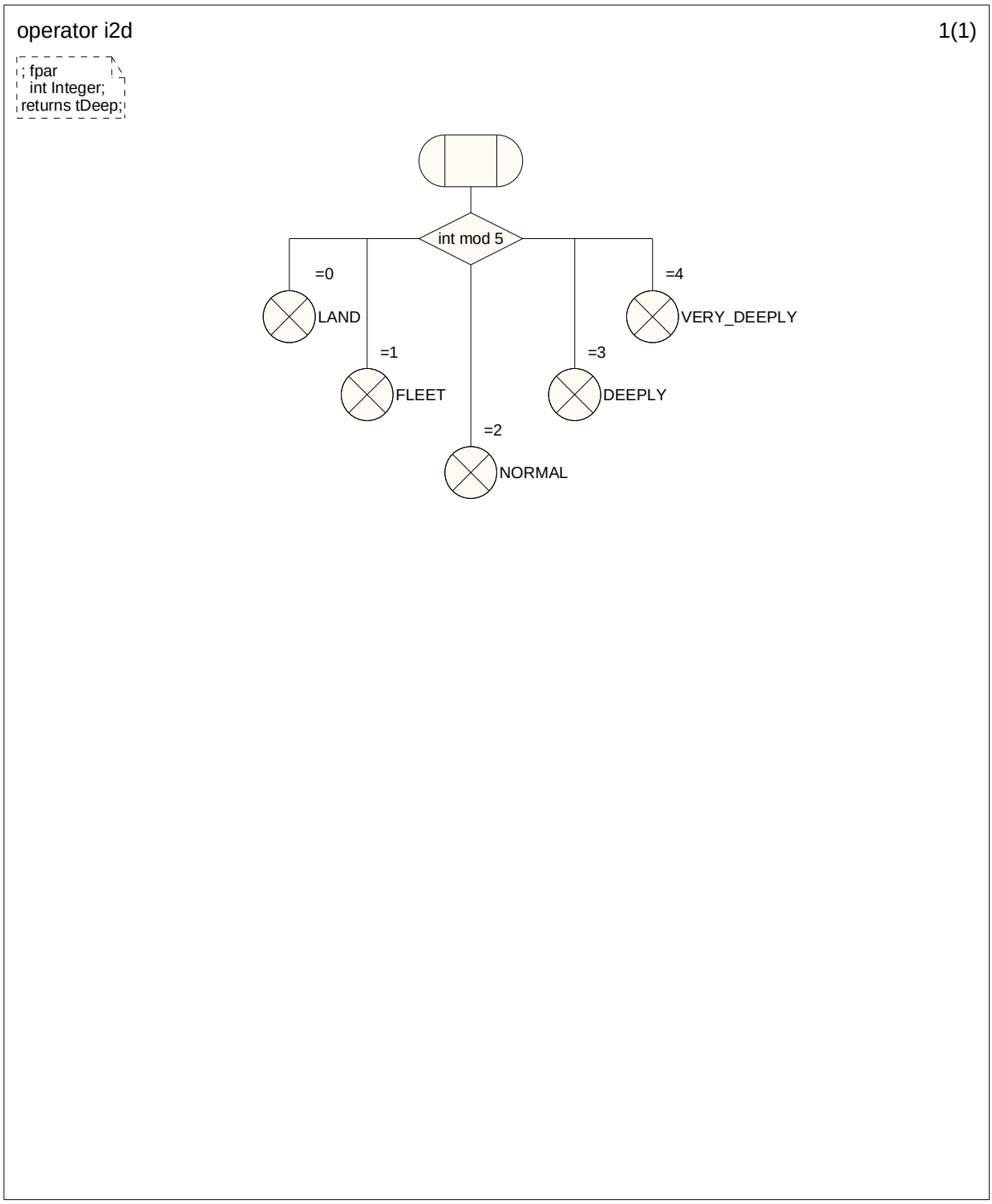

Рисунок 12. Оператор i2d.

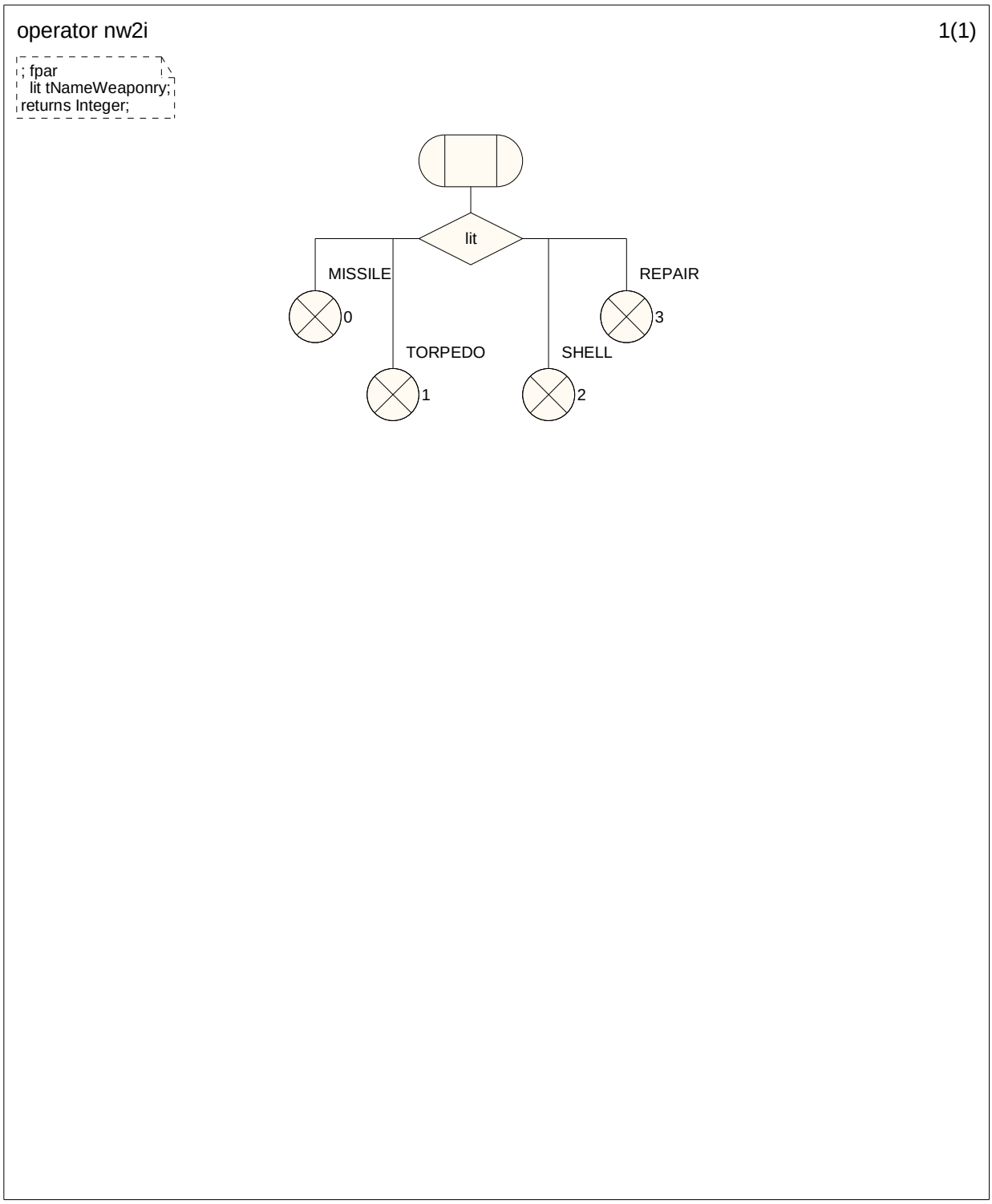

Рисунок 13. Оператор nw2i.

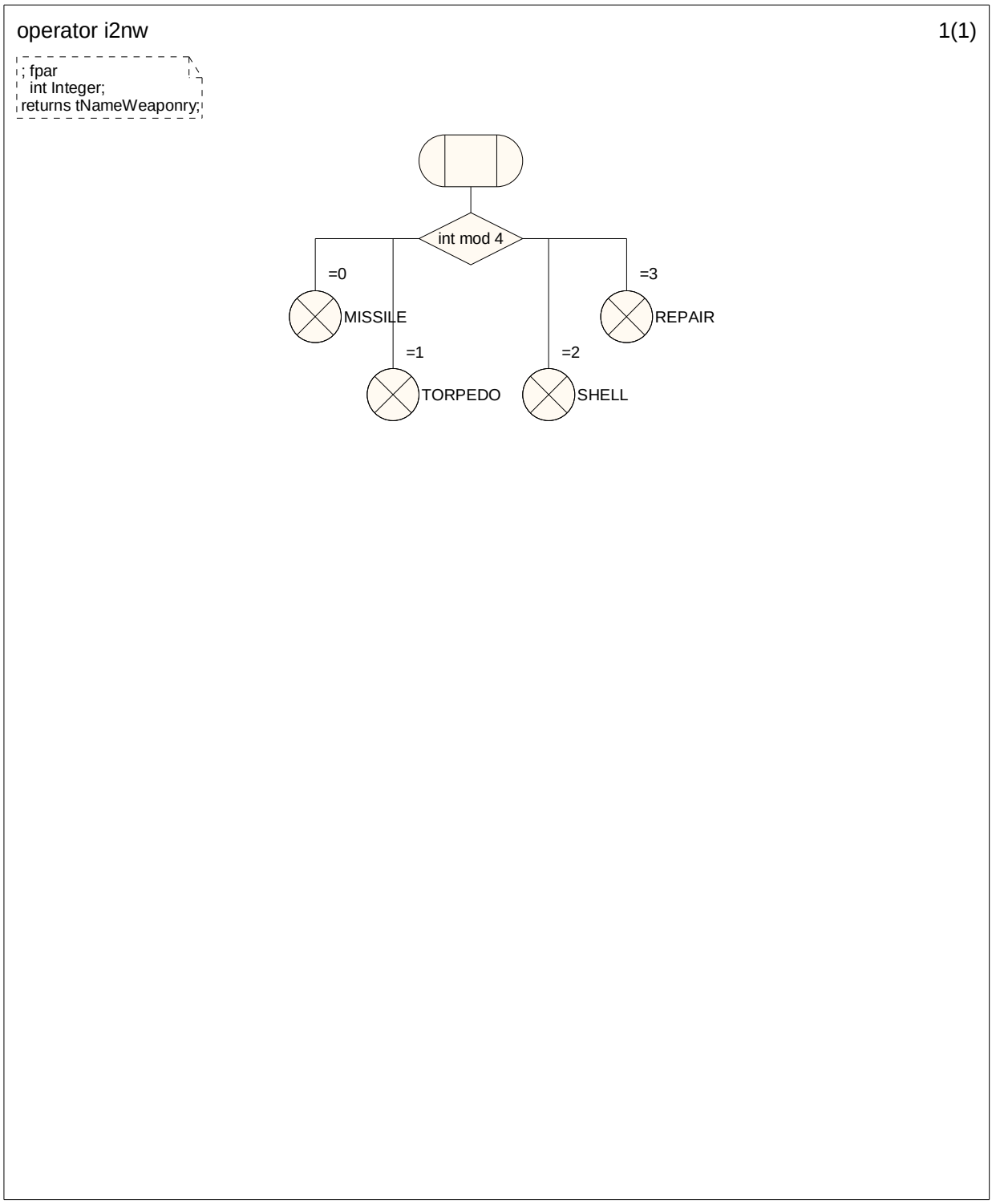

Рисунок 14. Оператор i2nw.

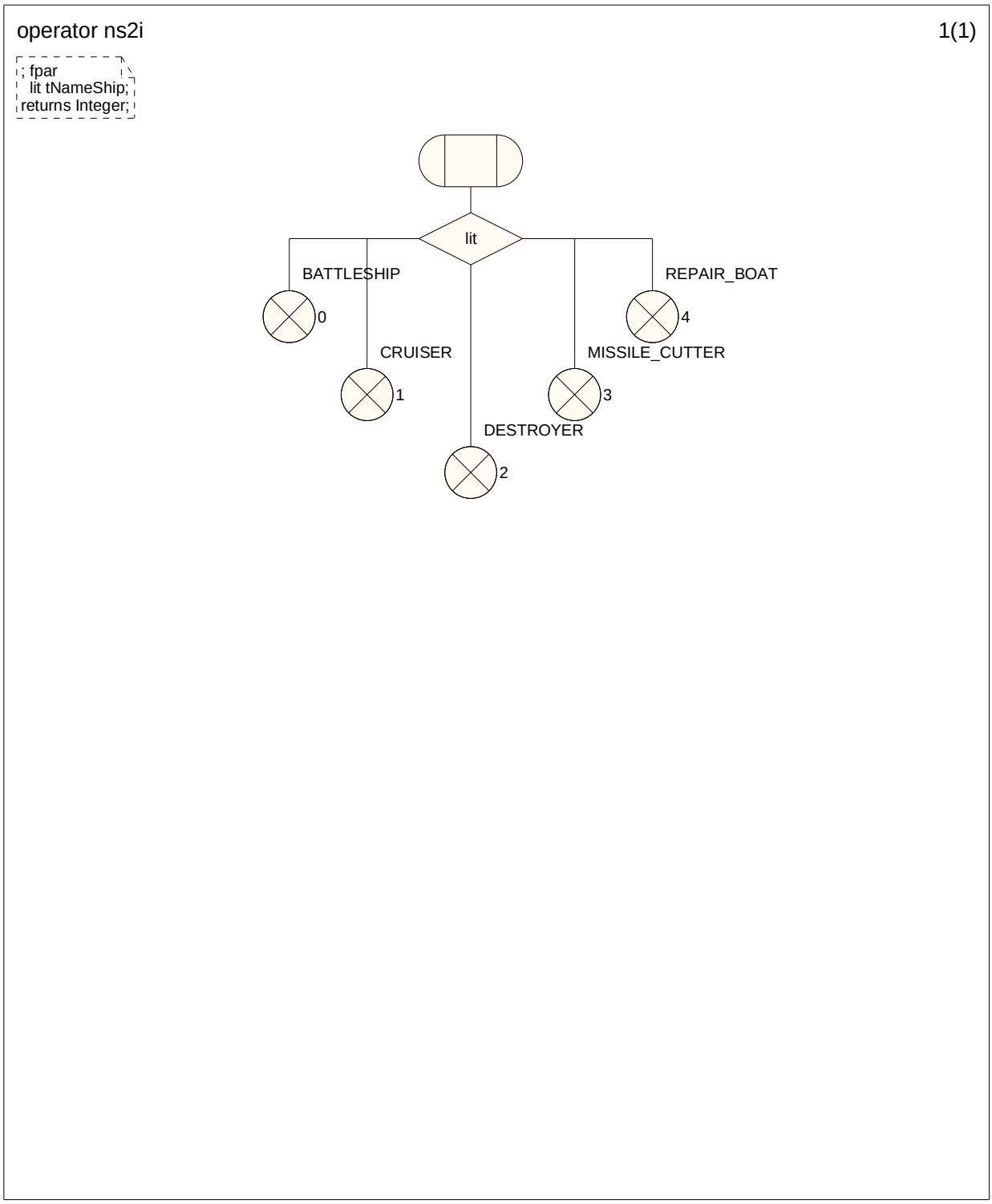

Рисунок 15. Оператор ns2i.

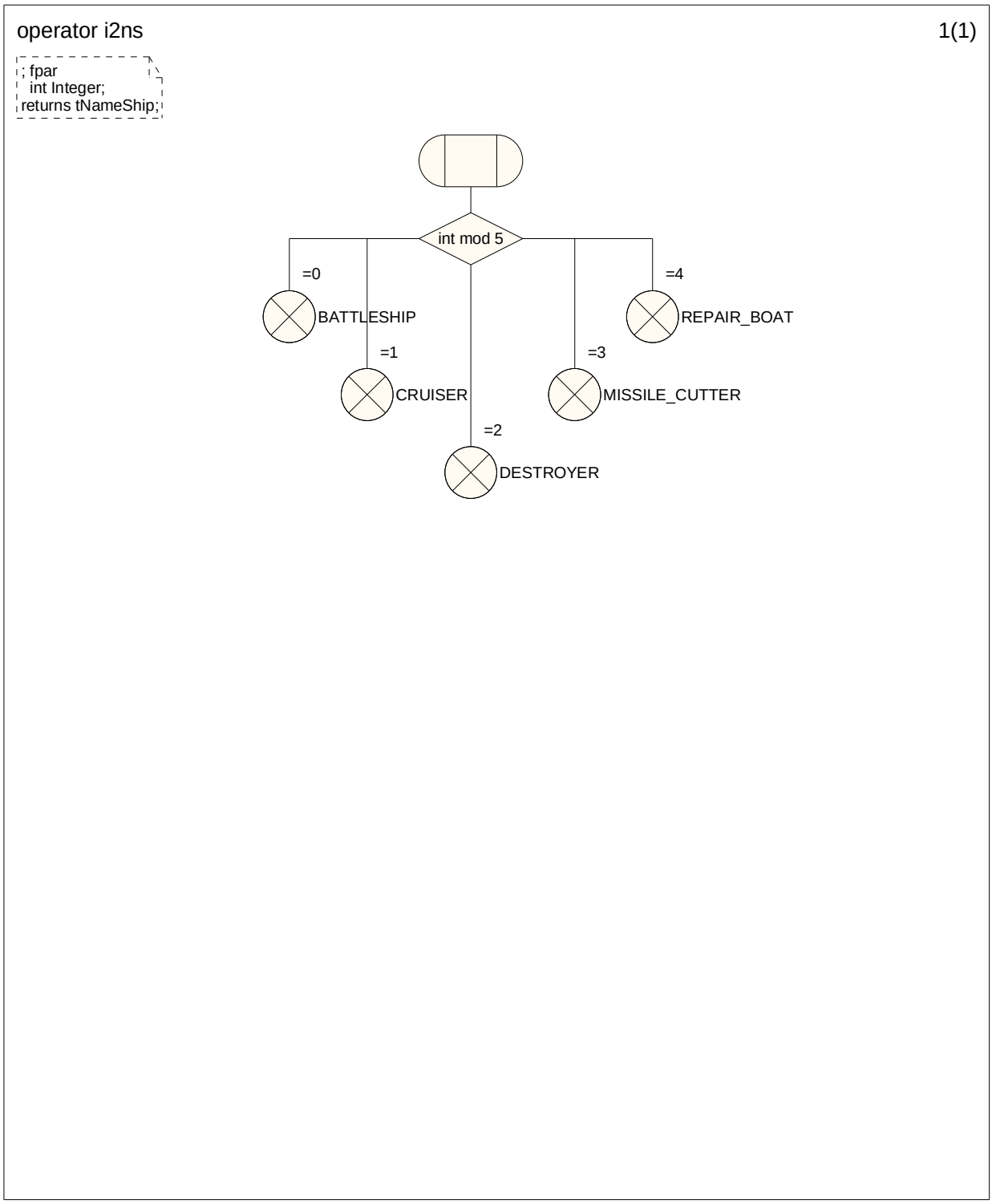

Рисунок 16. Оператор i2ns.

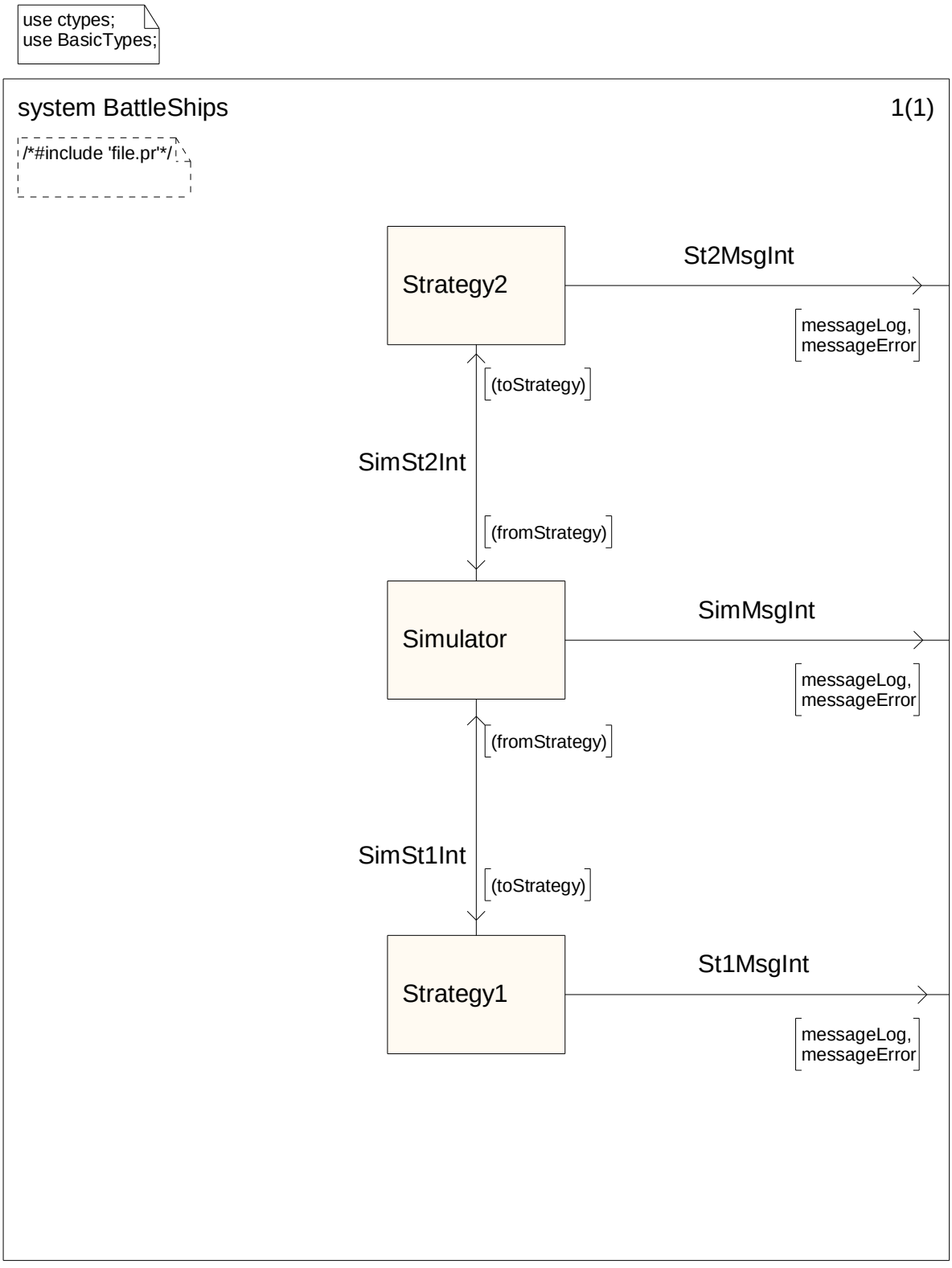

Рисунок 17. Система «BattleShips».

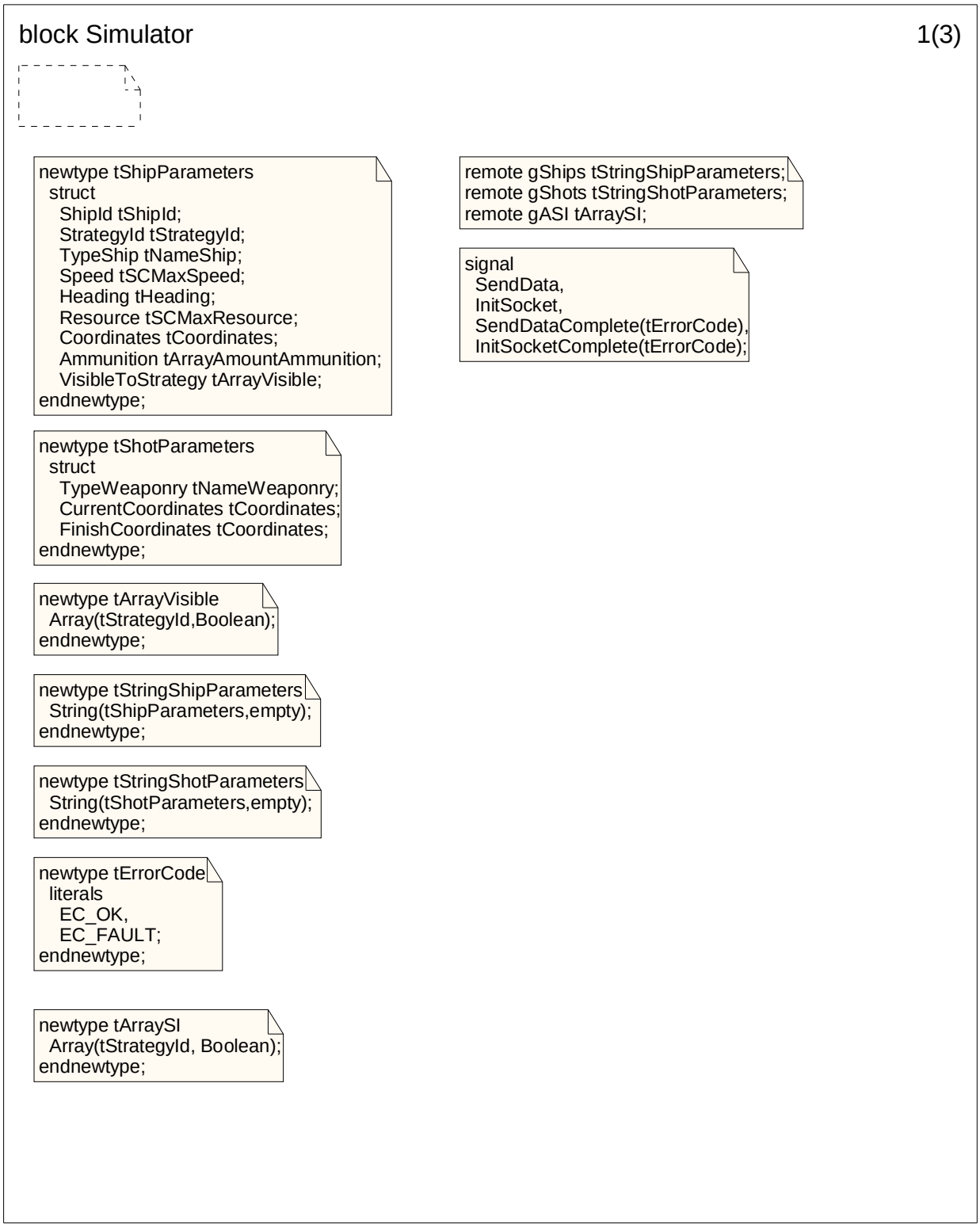

Рисунок 18. Блок симулятора (1).

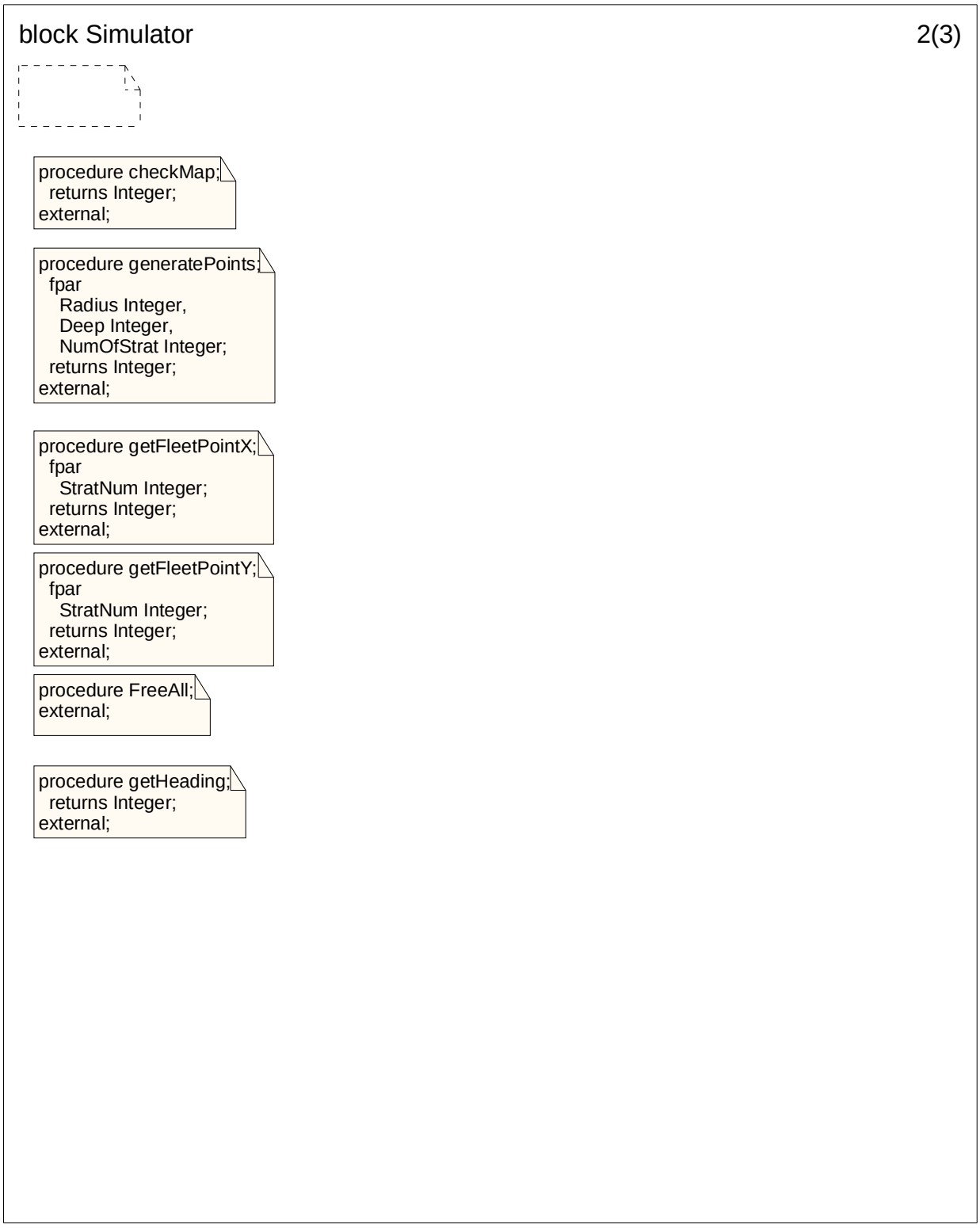

Рисунок 19. Блок симулятора (2).

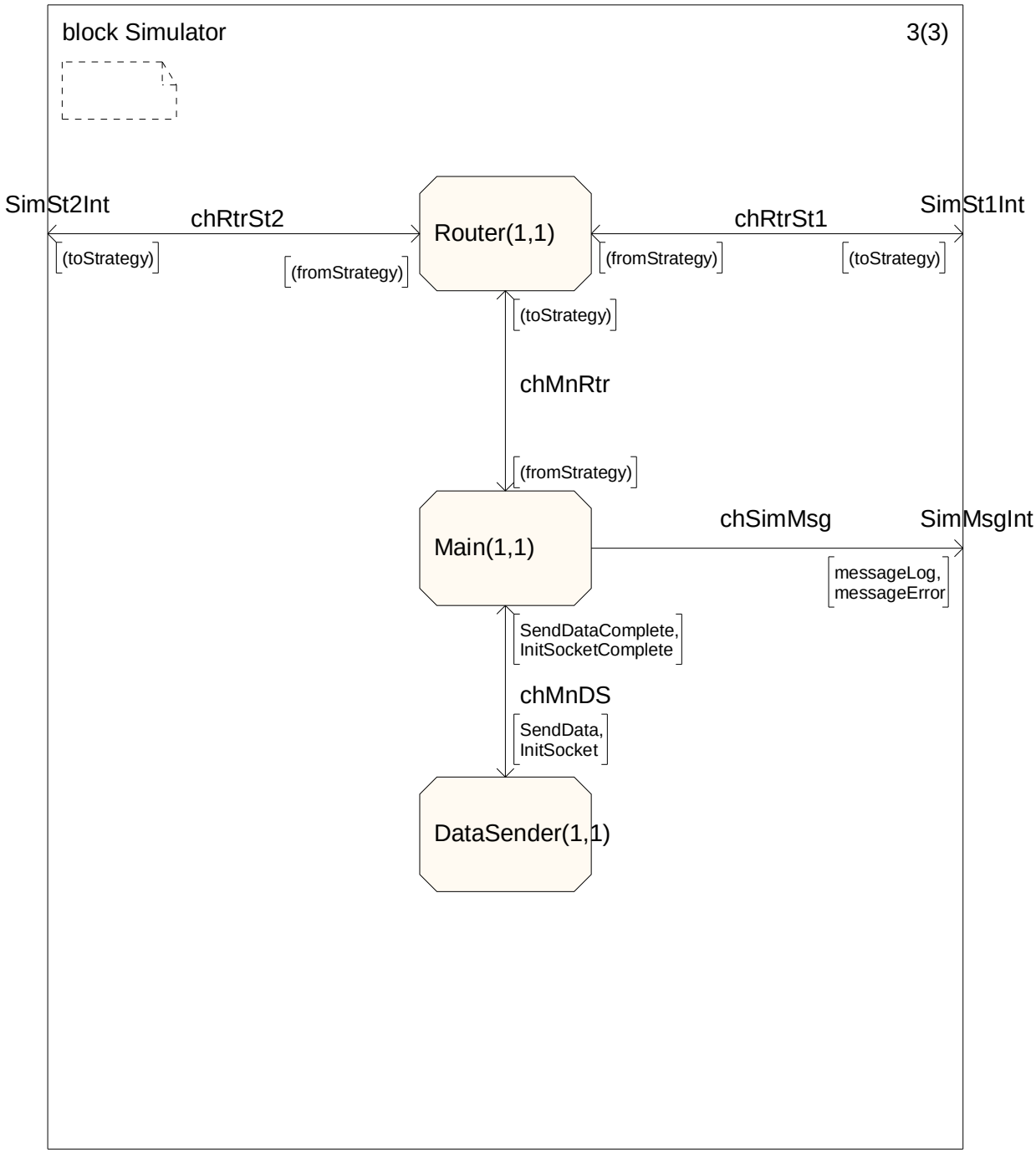

Рисунок 20. Блок симулятора (3).

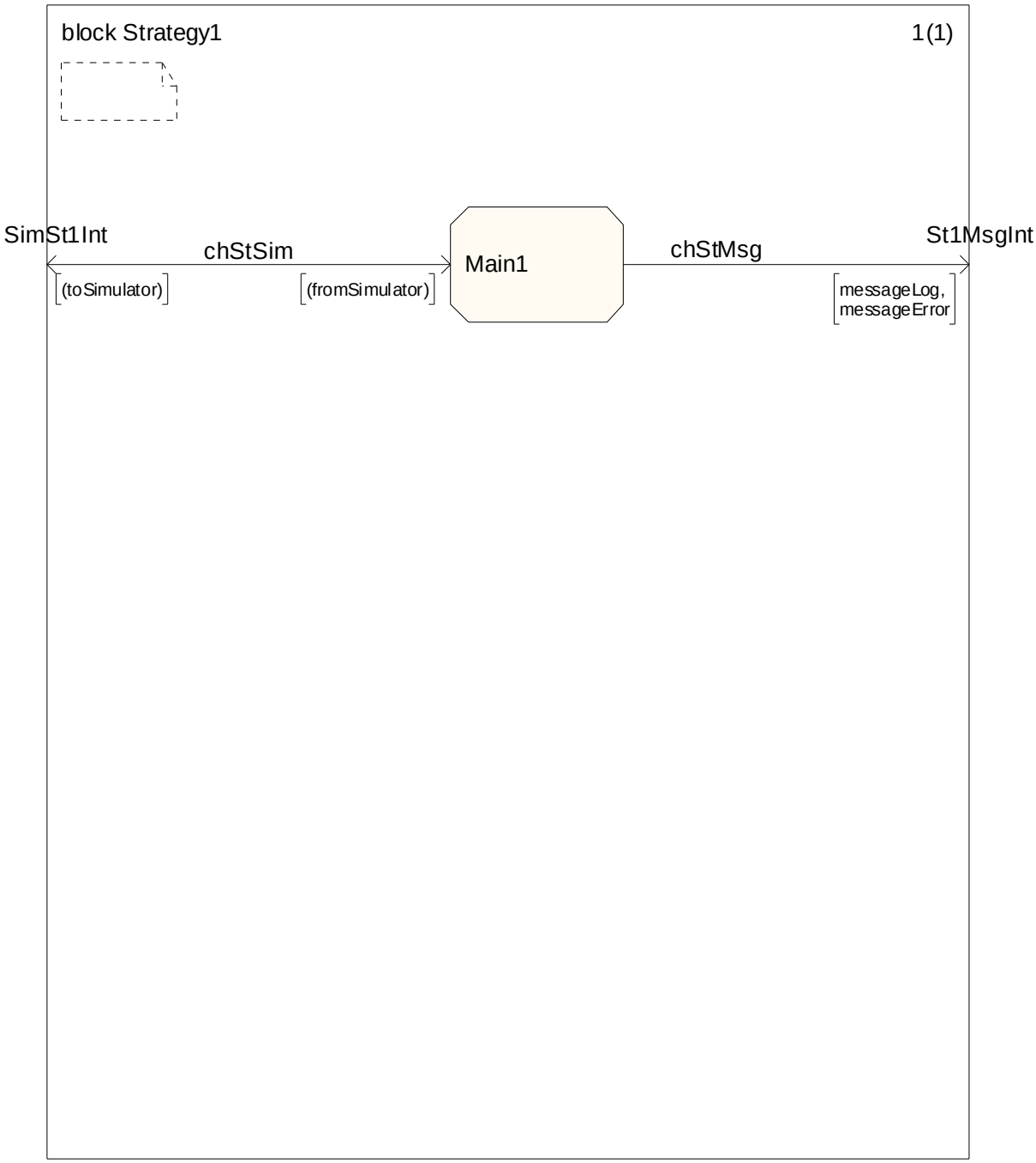

Рисунок 21. Блок стратегии.

Приложение B -- Код процесса Router (блок симулятора).

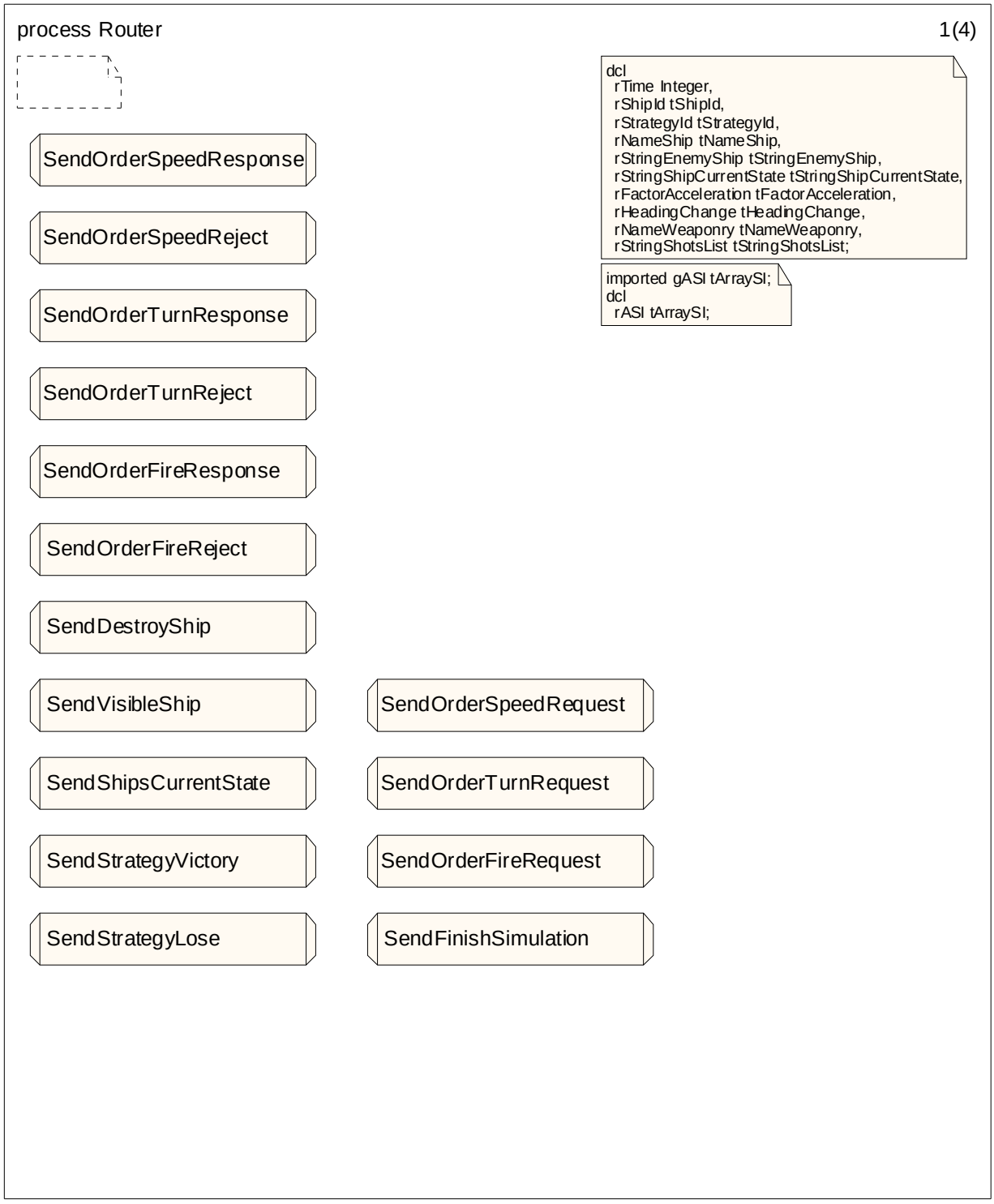

Рисунок 22. Процесс Router (1).

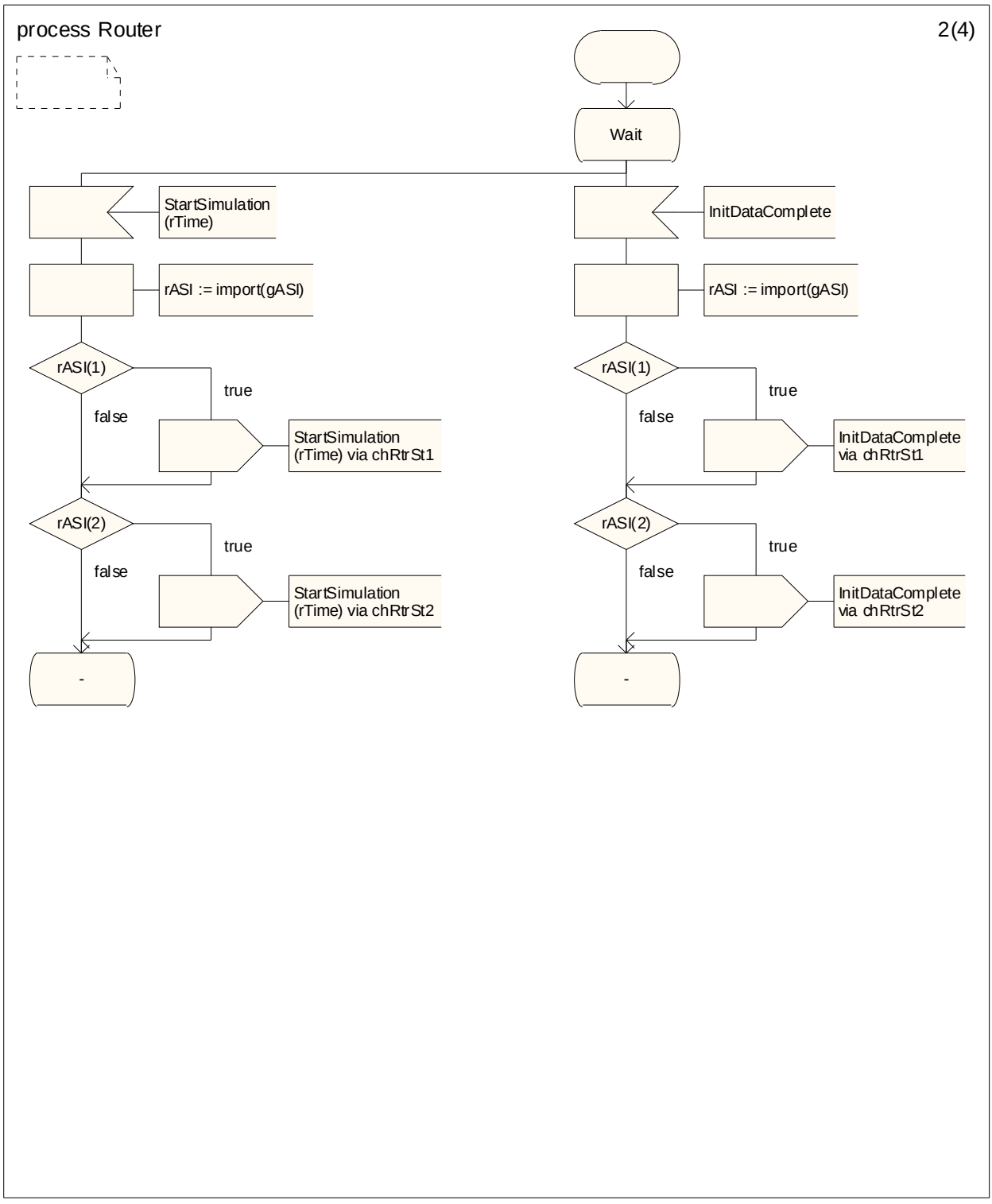

Рисунок 23. Процесс Router (2).

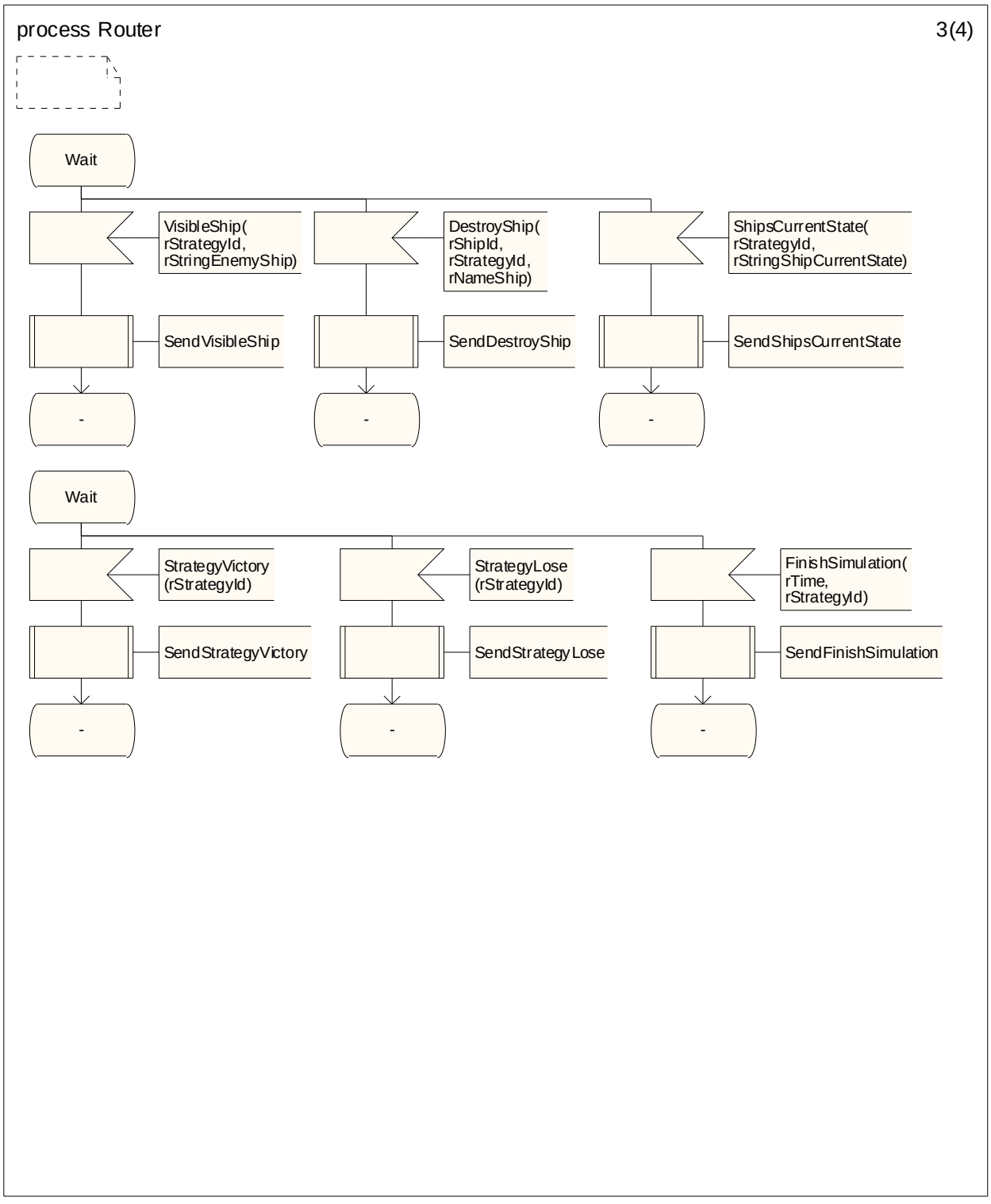

Рисунок 24. Процесс Router (3).

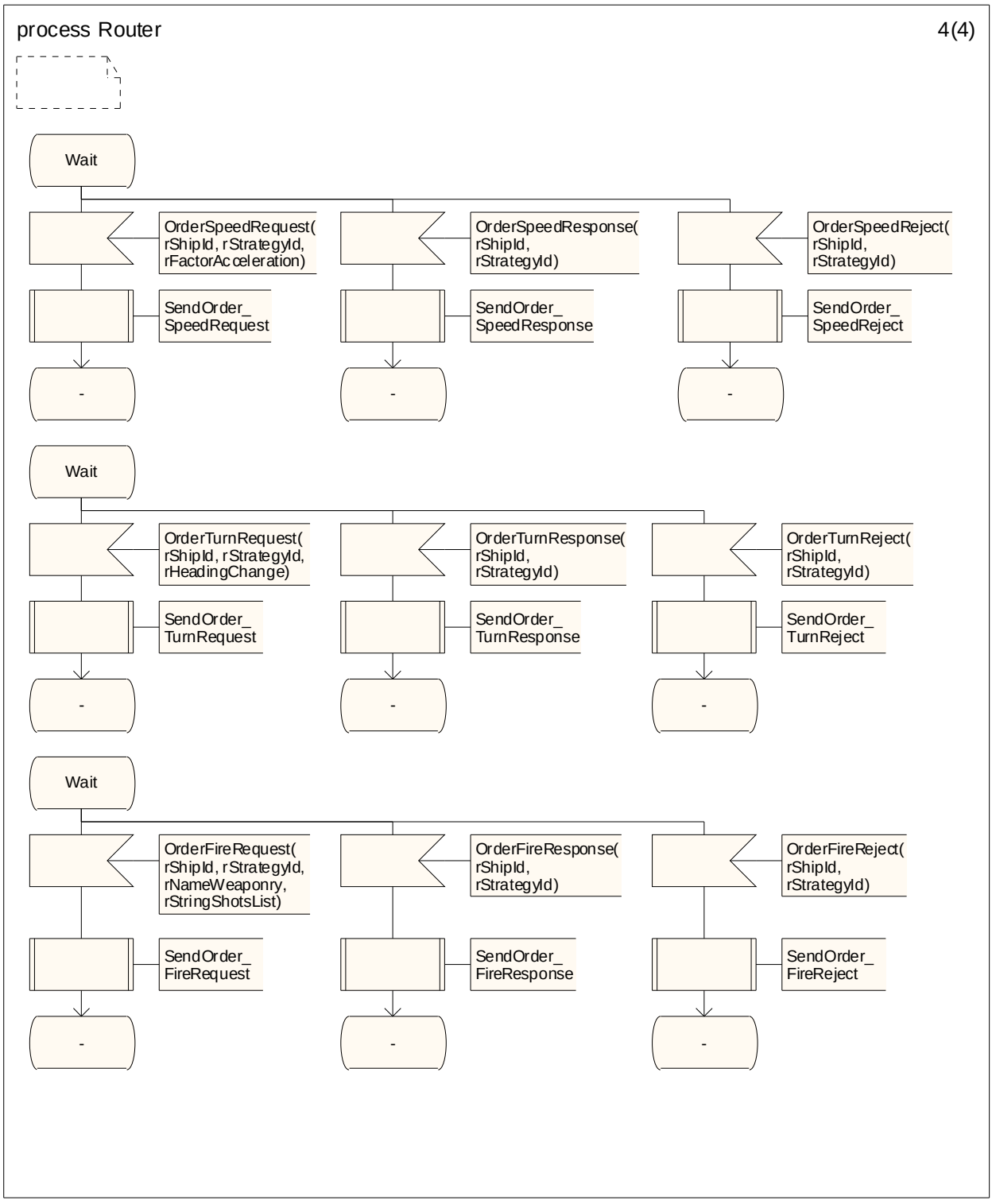

Рисунок 25. Процесс Router (4).

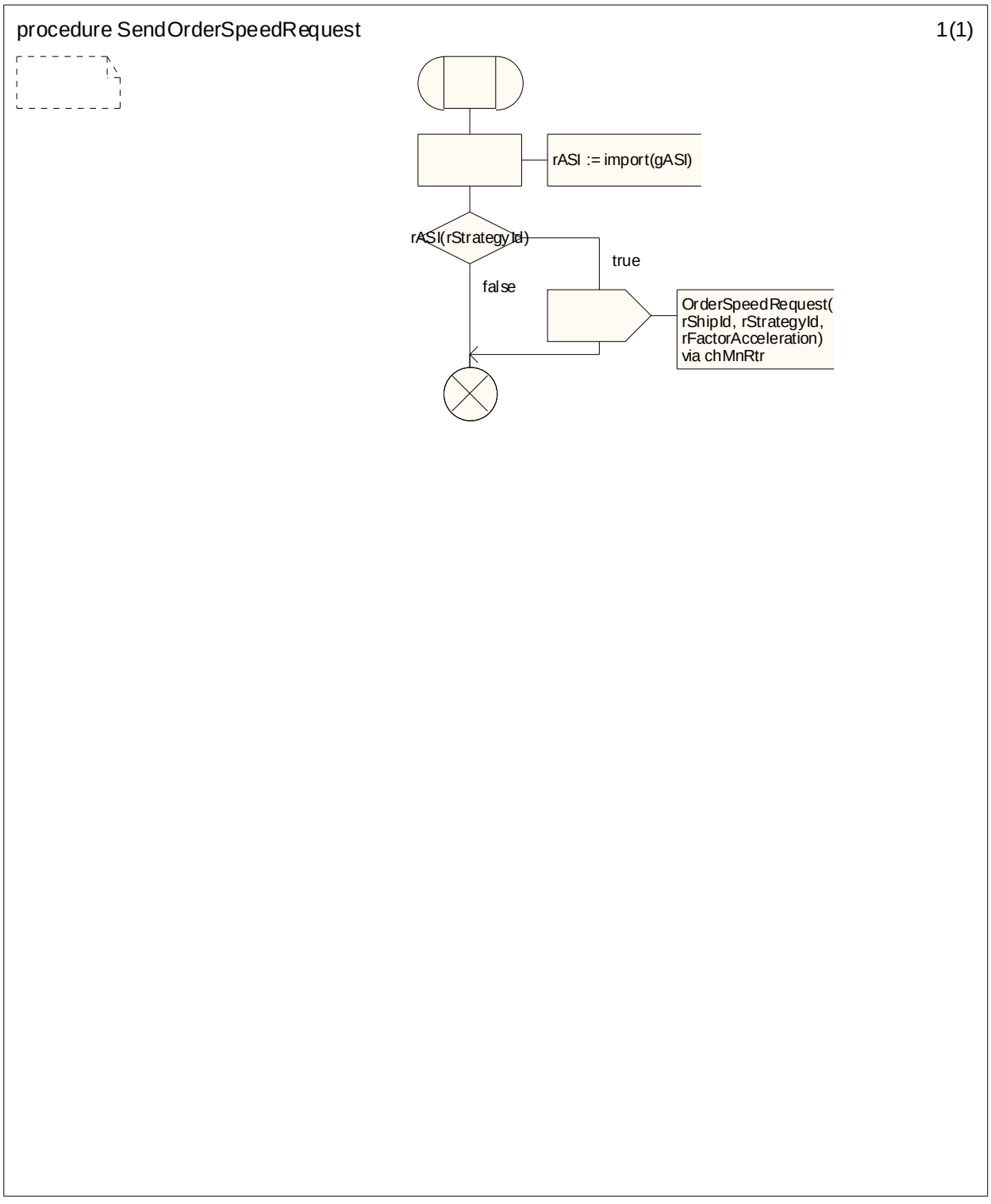

Рисунок 26. Процедура SendOrderSpeedRequest.

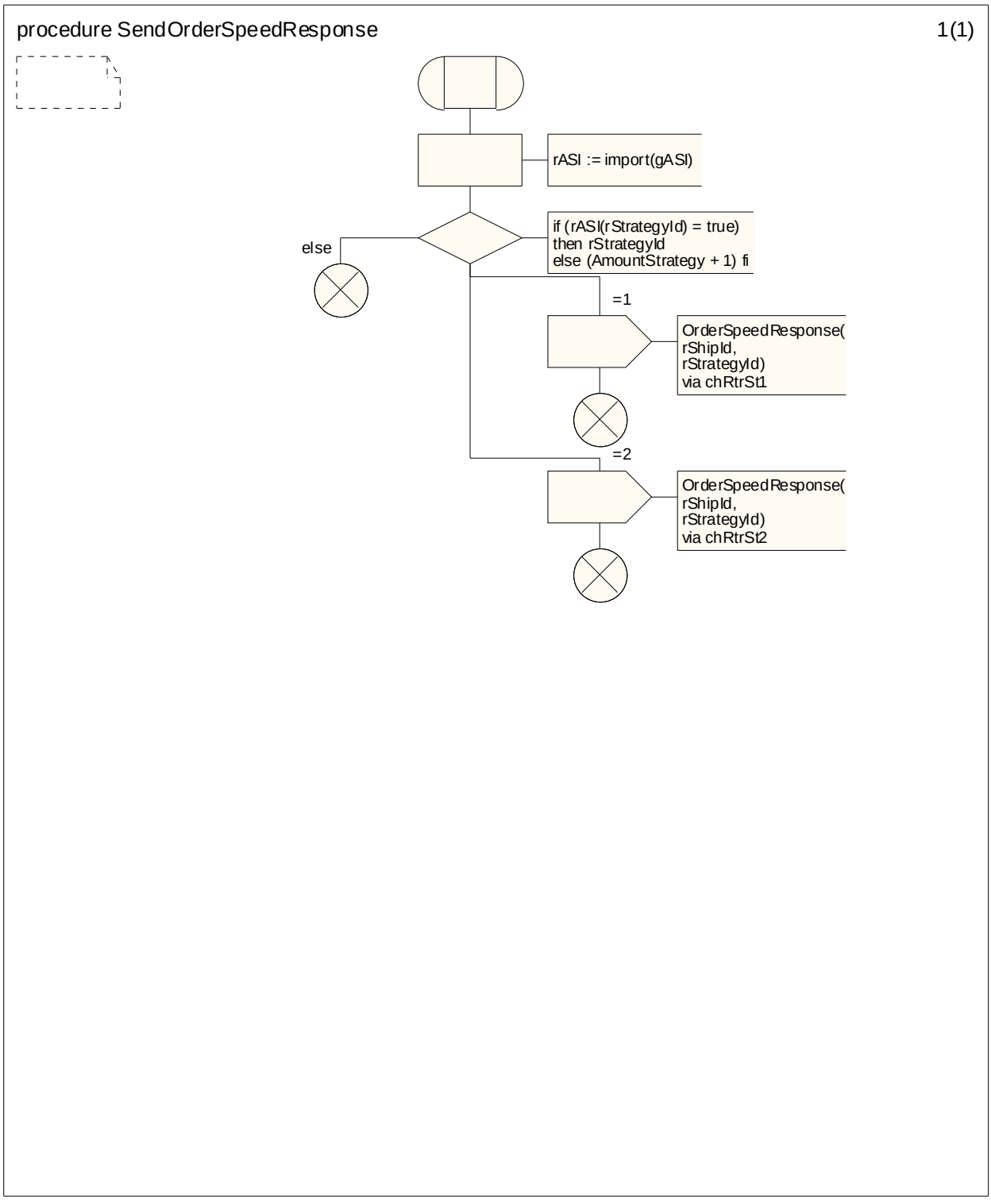

Рисунок 27. Процедура SendOrderSpeedResponse.

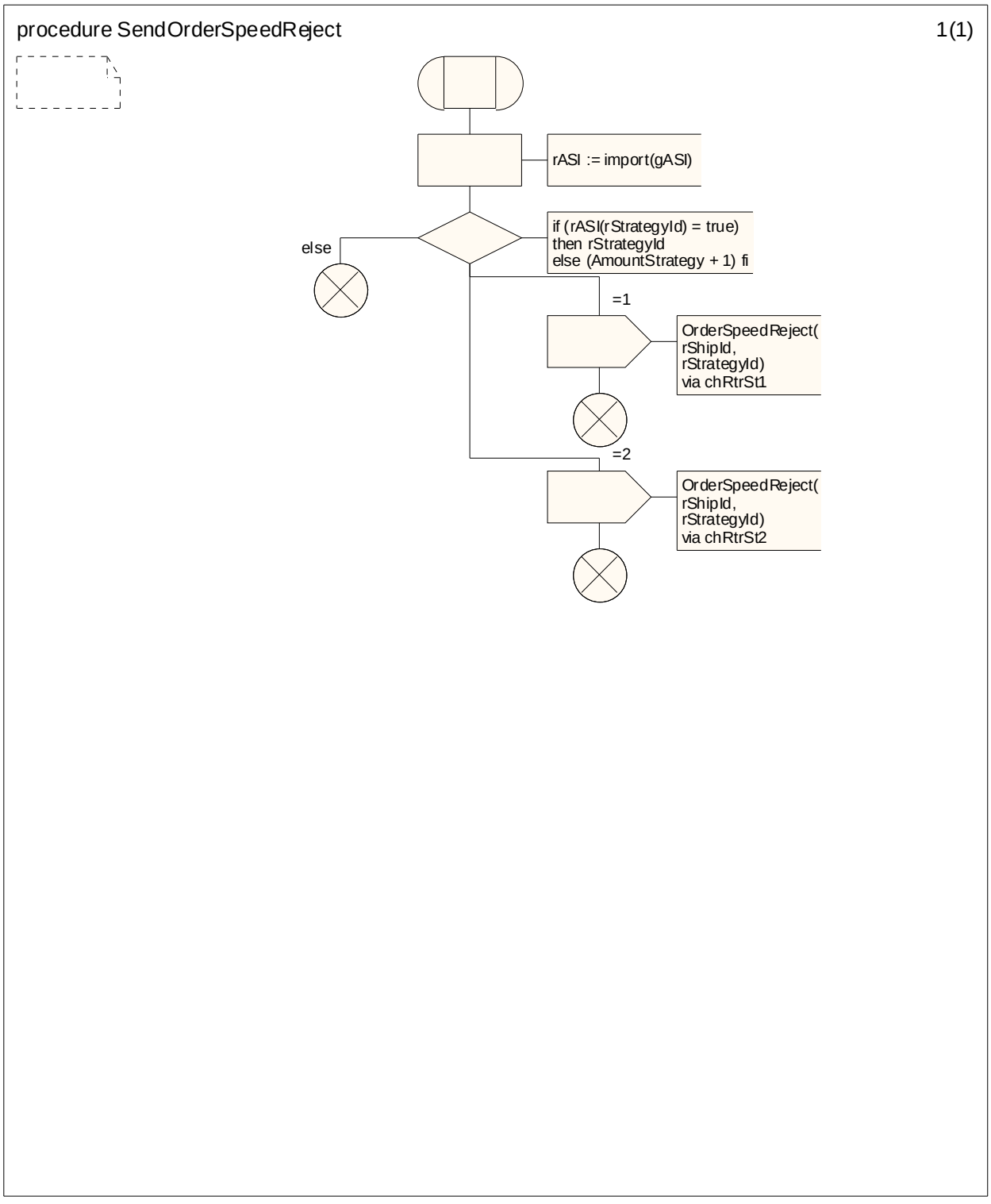

Рисунок 28. Процедура SendOrderSpeedReject.

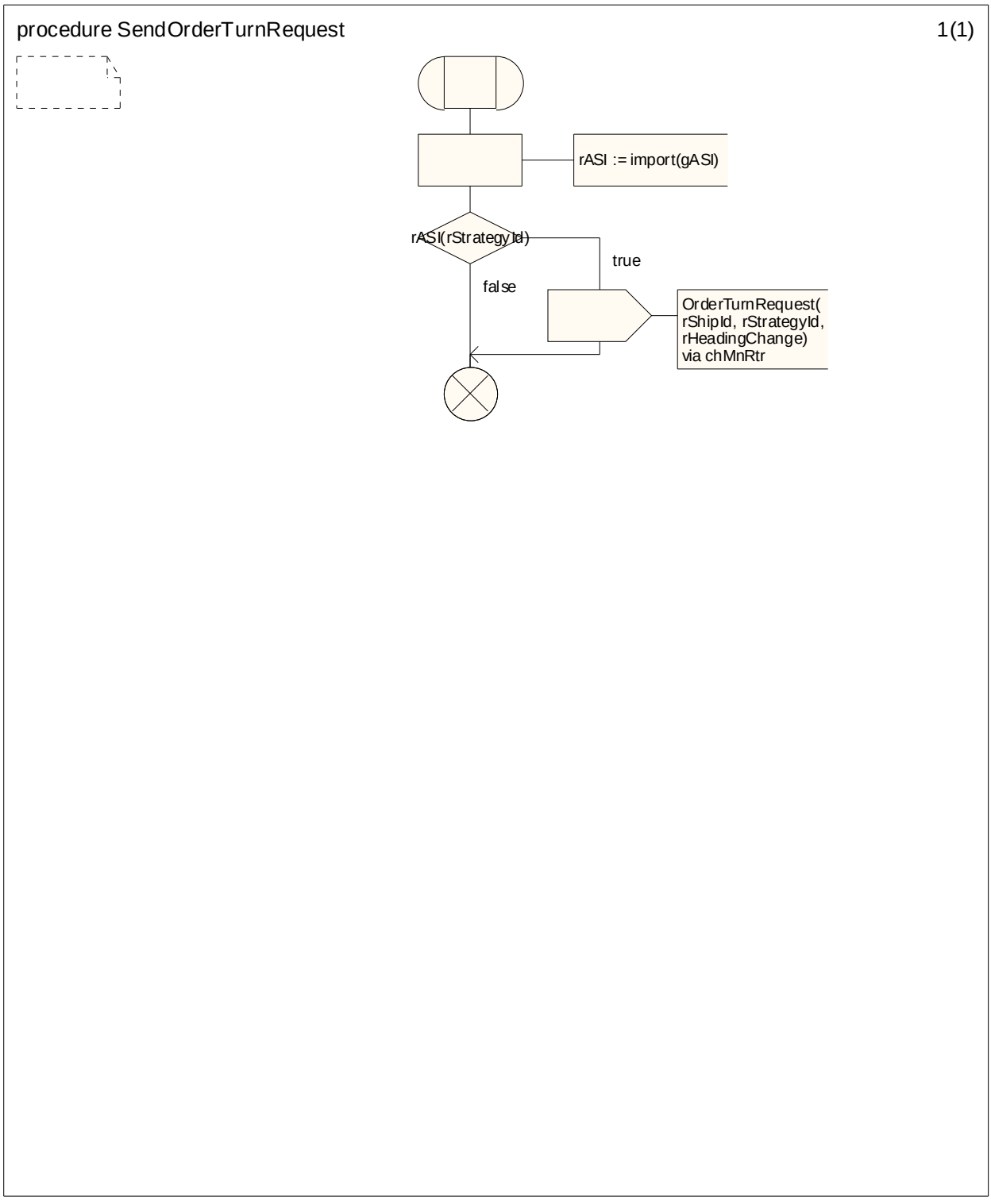

Рисунок 29. Процедура SendOrderTurnRequest.

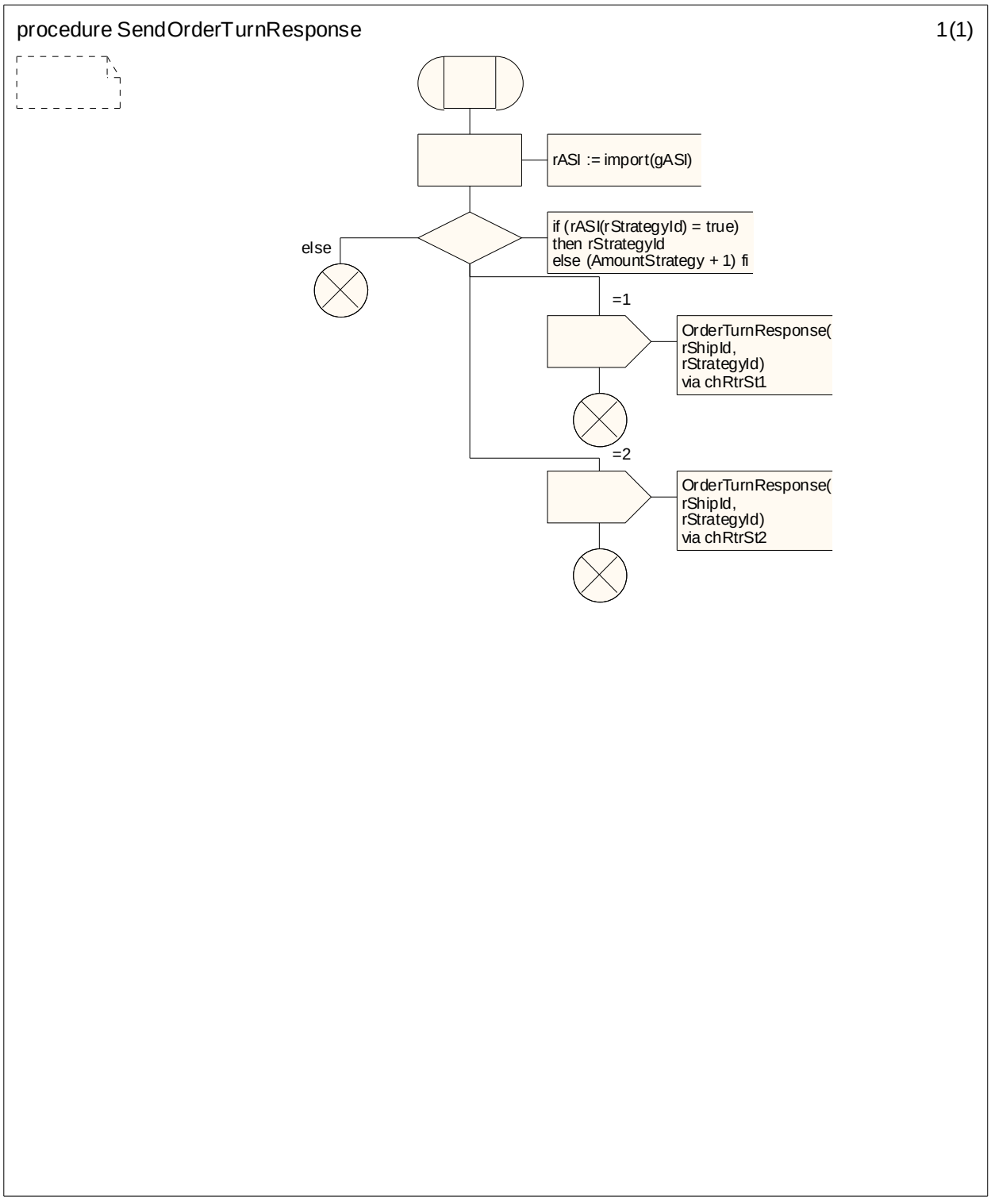

Рисунок 30. Процедура SendOrderTurnResponse.

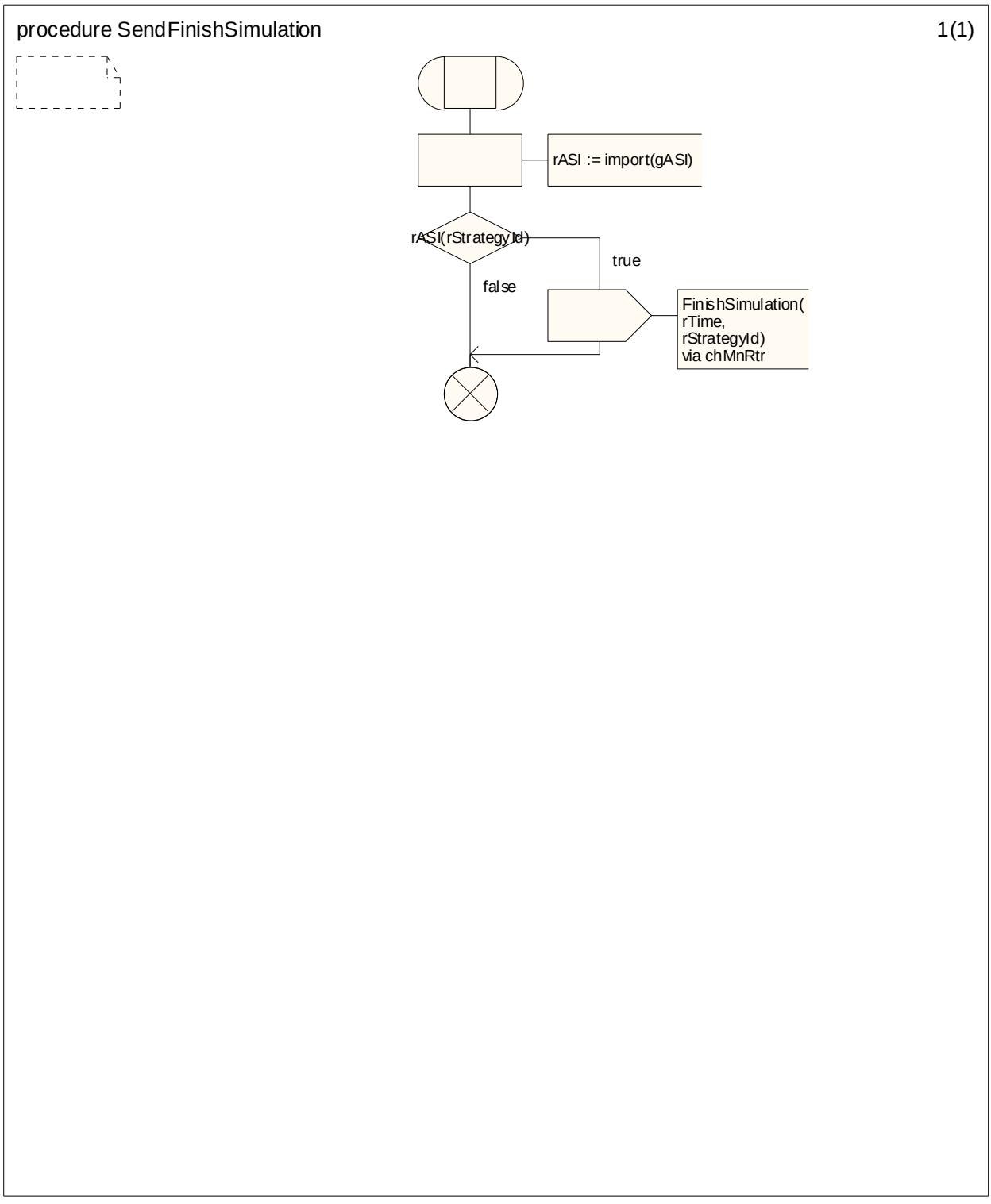

Рисунок 31. Процедура SendFinishSimulation.

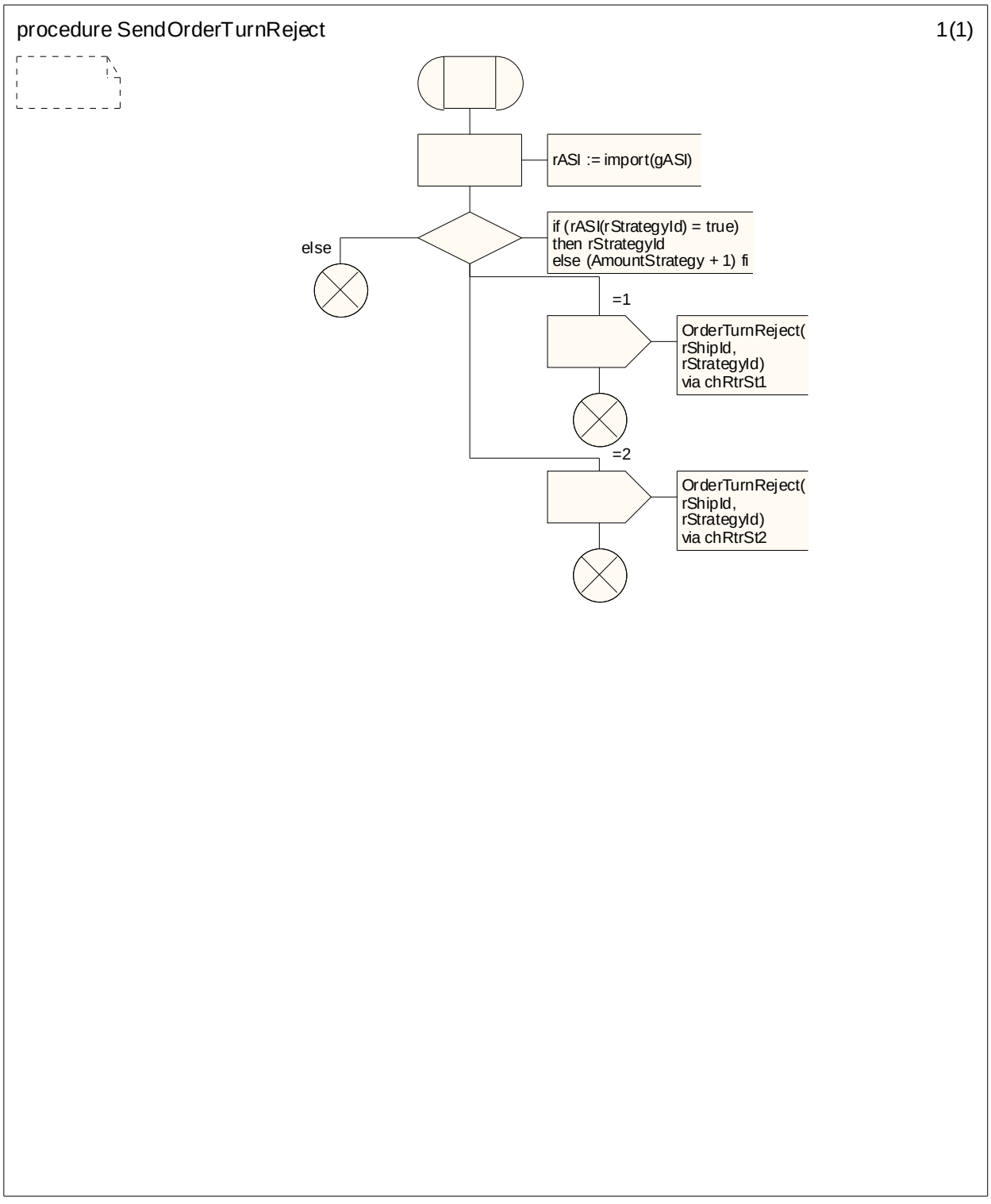

Рисунок 32. Процедура SendOrderTurnReject.

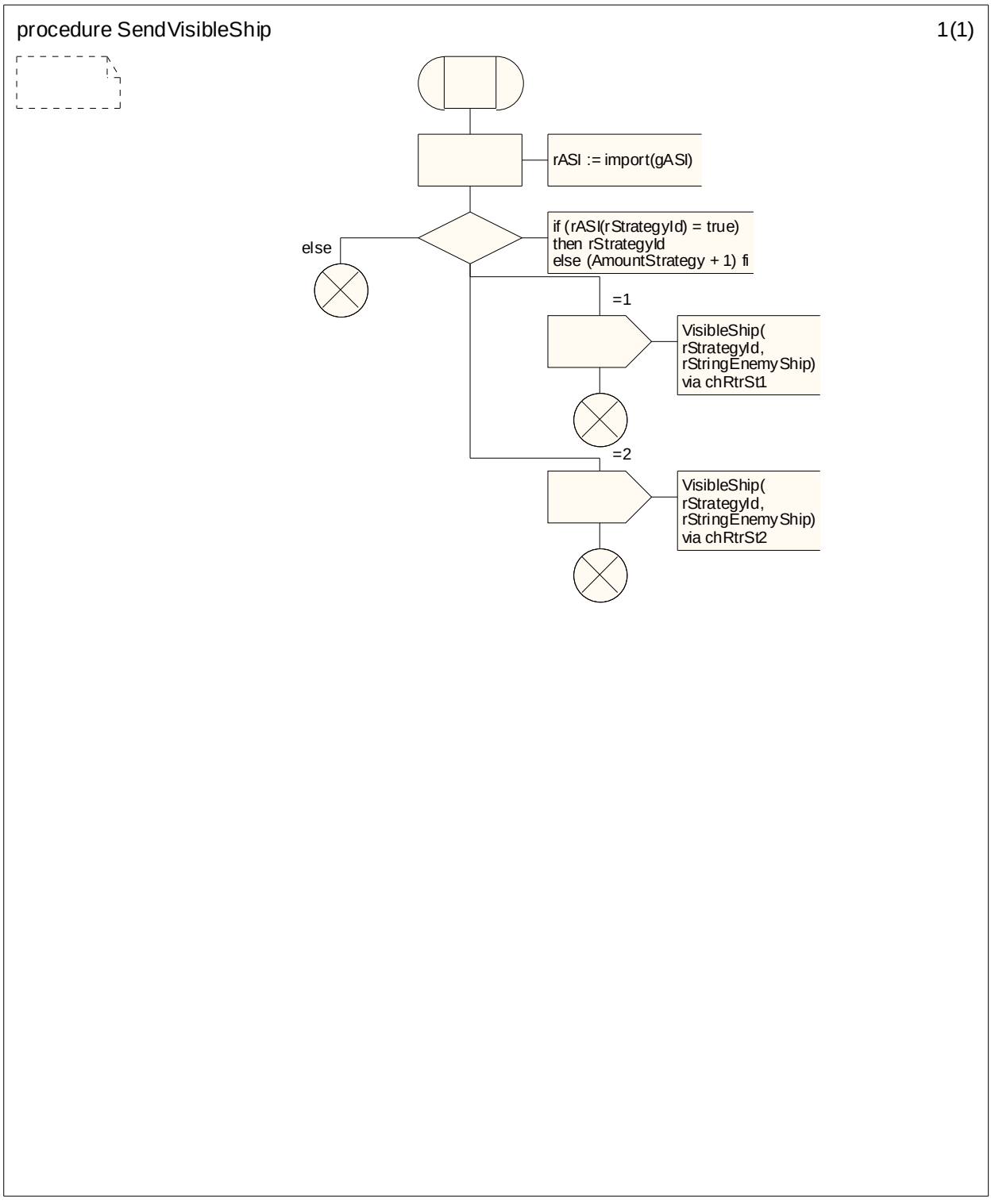

Рисунок 33. Процедура SendVisibleShip.

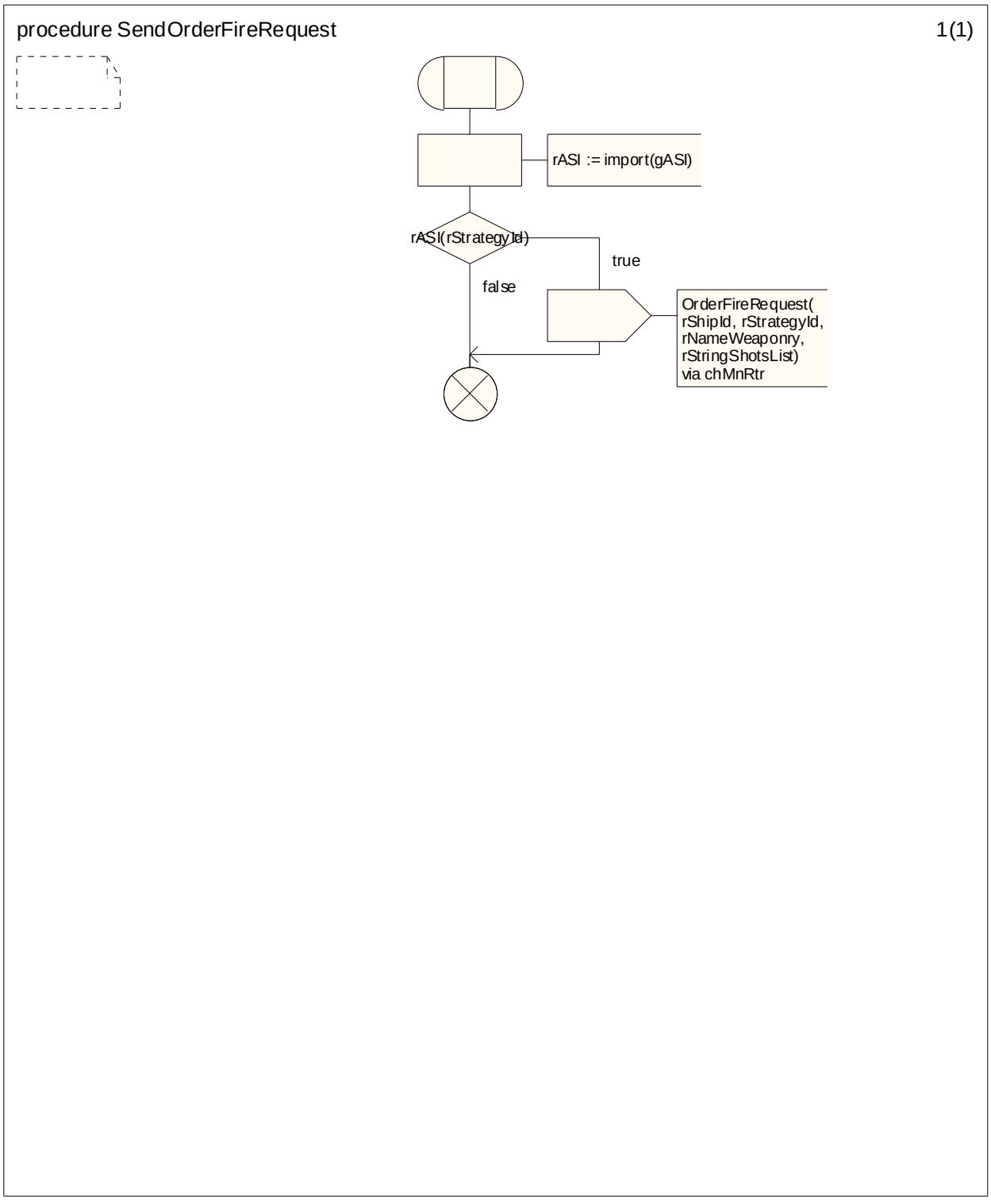

Рисунок 34. Процедура SendOrderFireRequest.

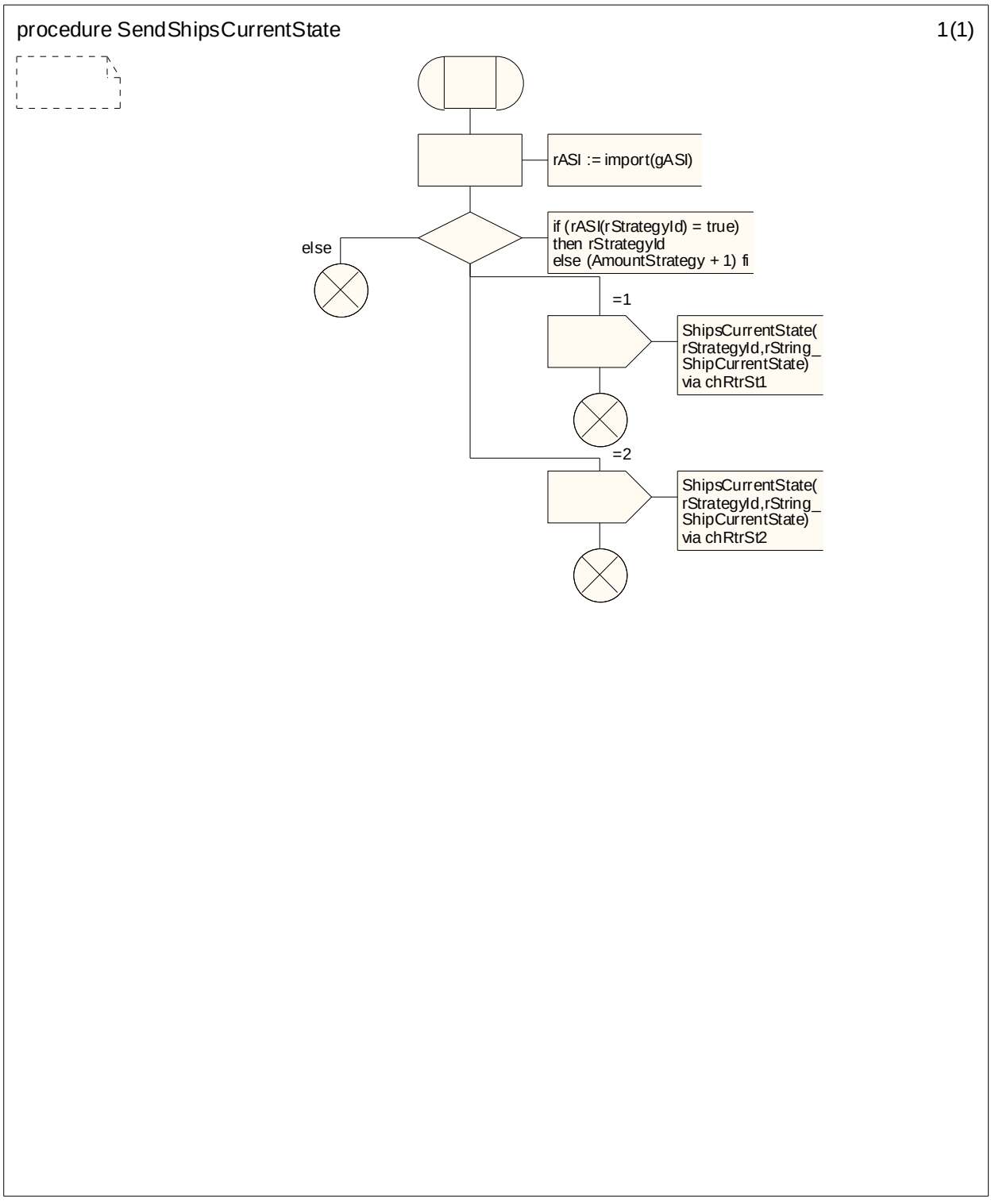

Рисунок 35. Процедура SendShipsCurrentState.

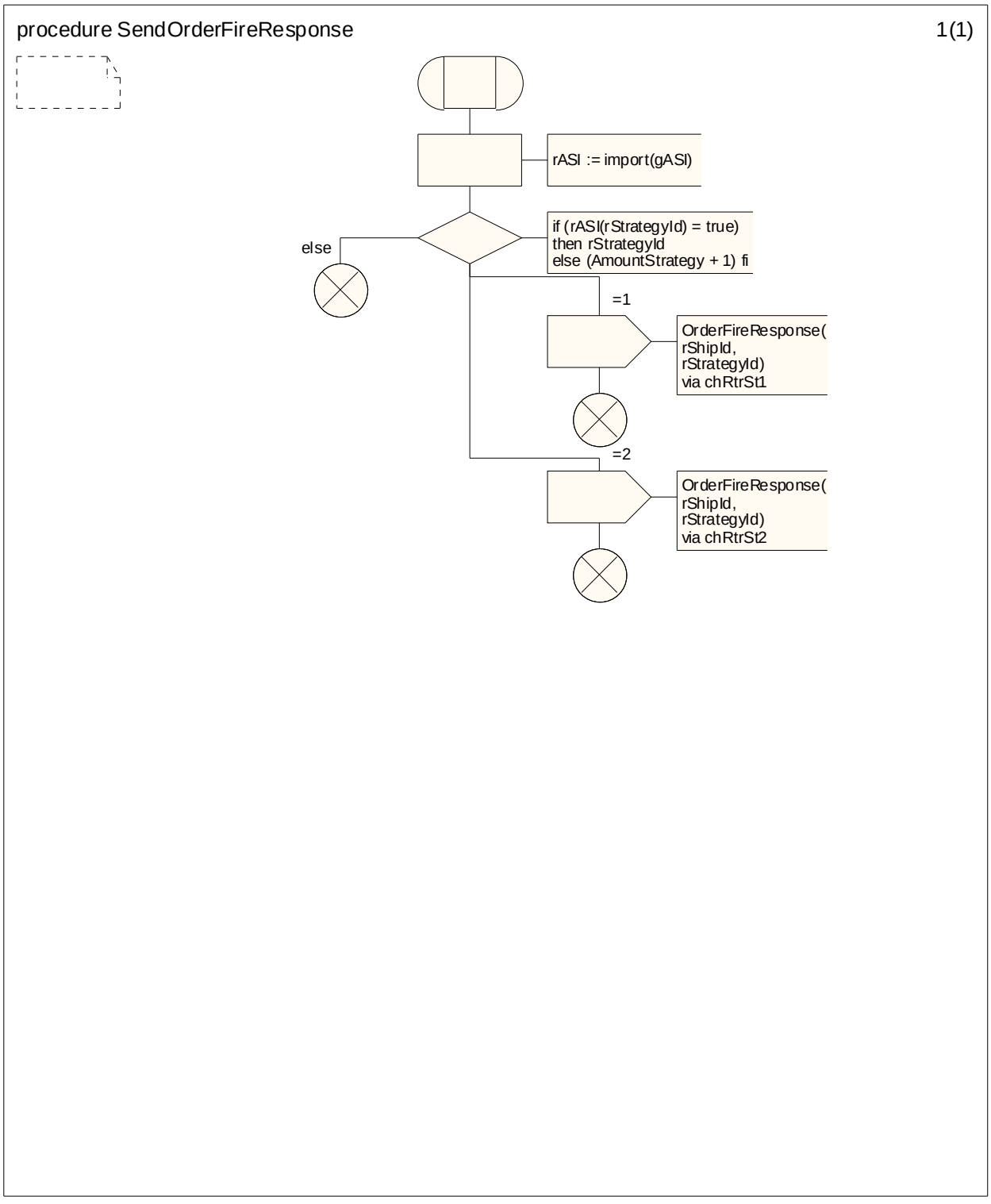

Рисунок 36. Процедура SendOrderFireResponse.

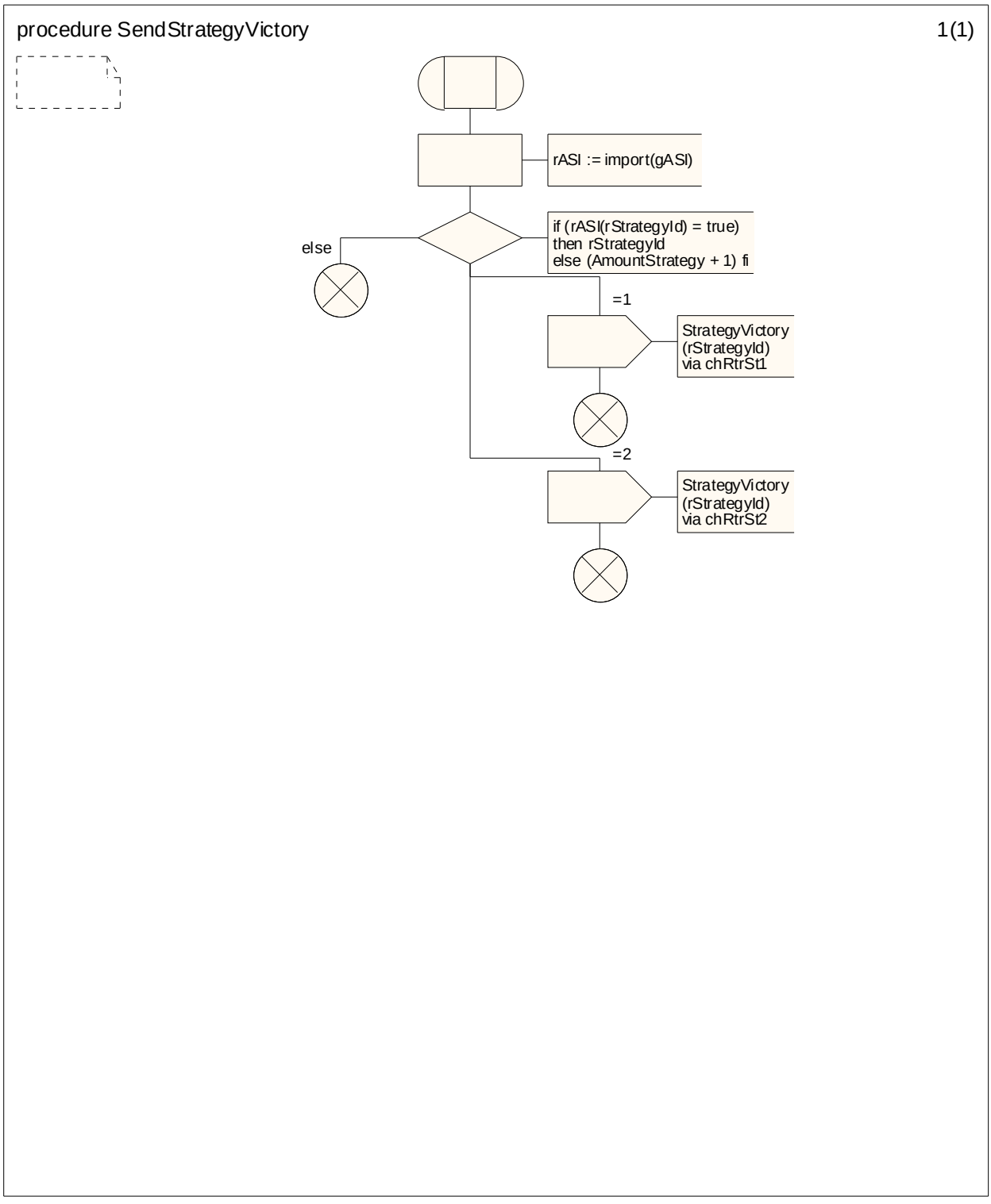

Рисунок 37. Процедура SendStrategyVictory.

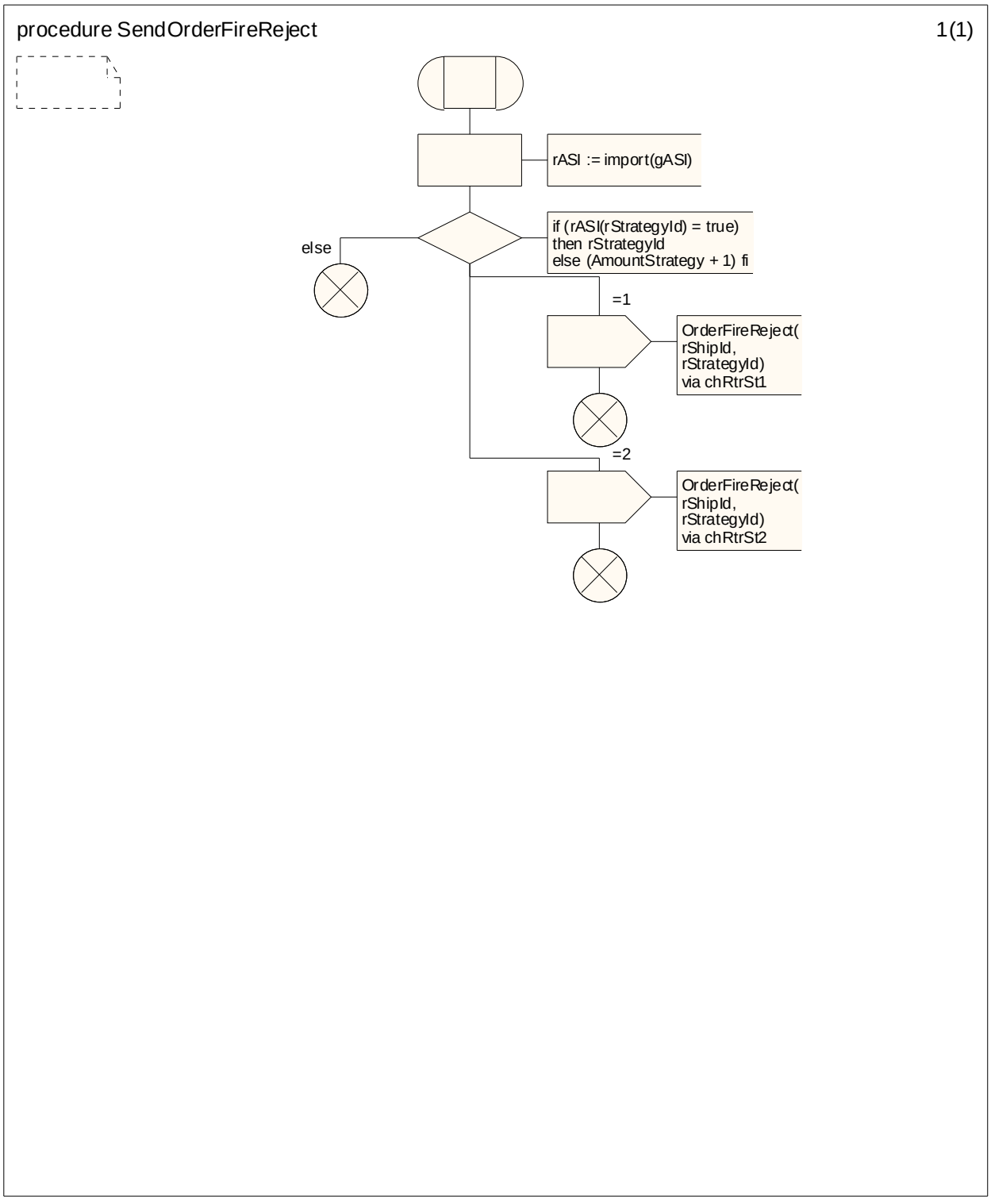

Рисунок 38. Процедура SendOrderFireReject.

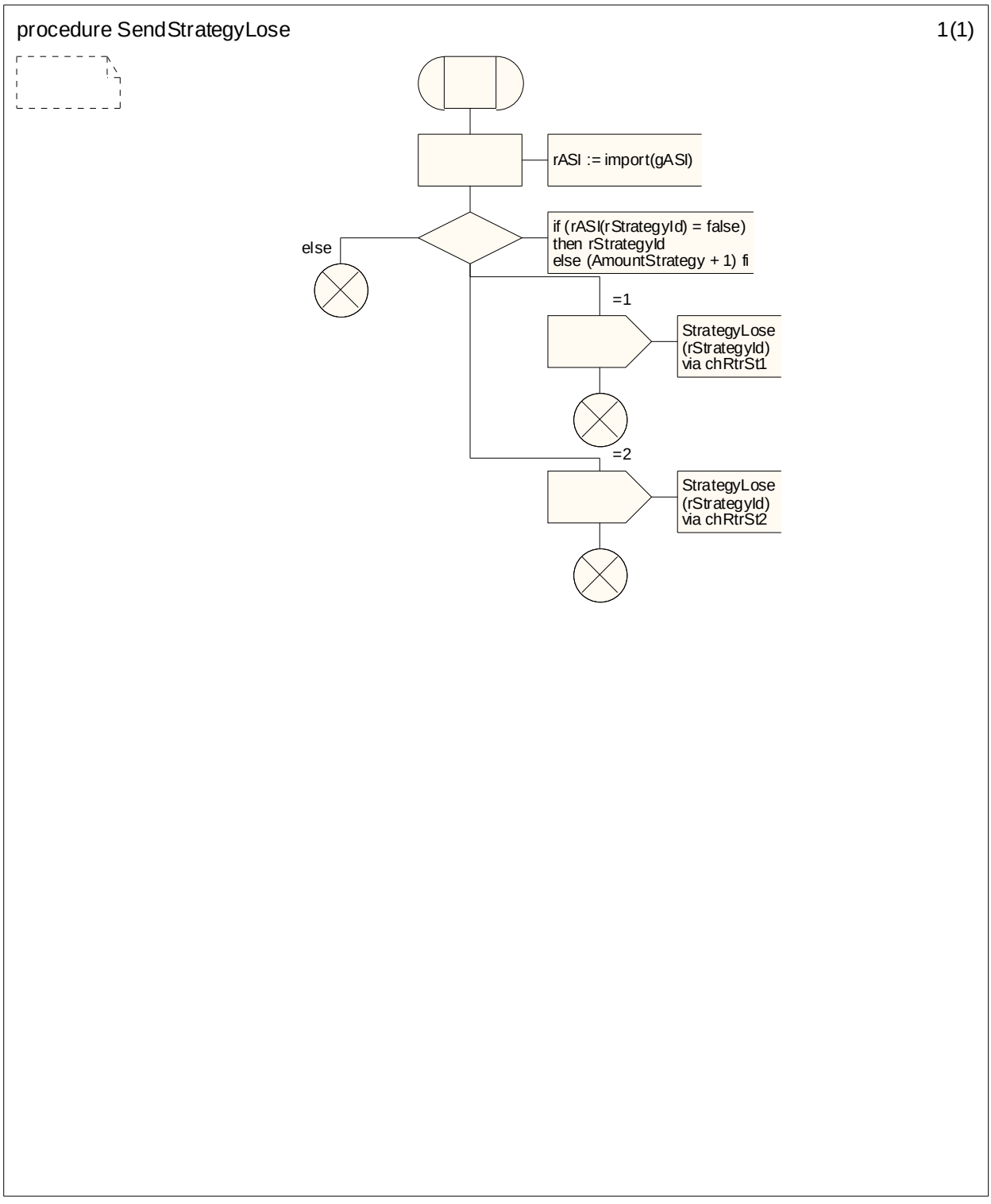

Рисунок 39. Процедура SendStrategyLose.

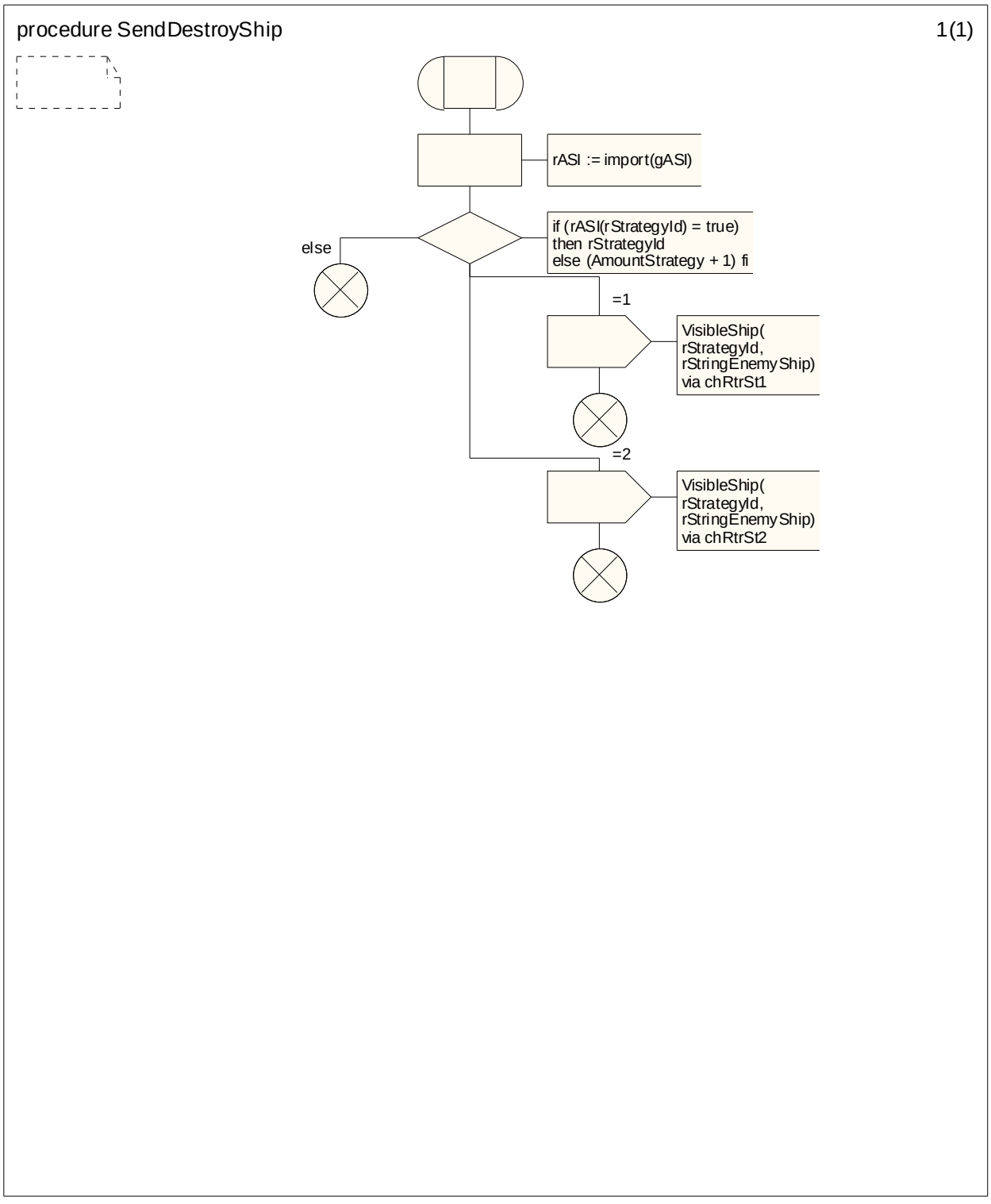

Рисунок 40. Процедура SendDestroyShip.

Приложение C -- Код процесса Main (блок симулятора).

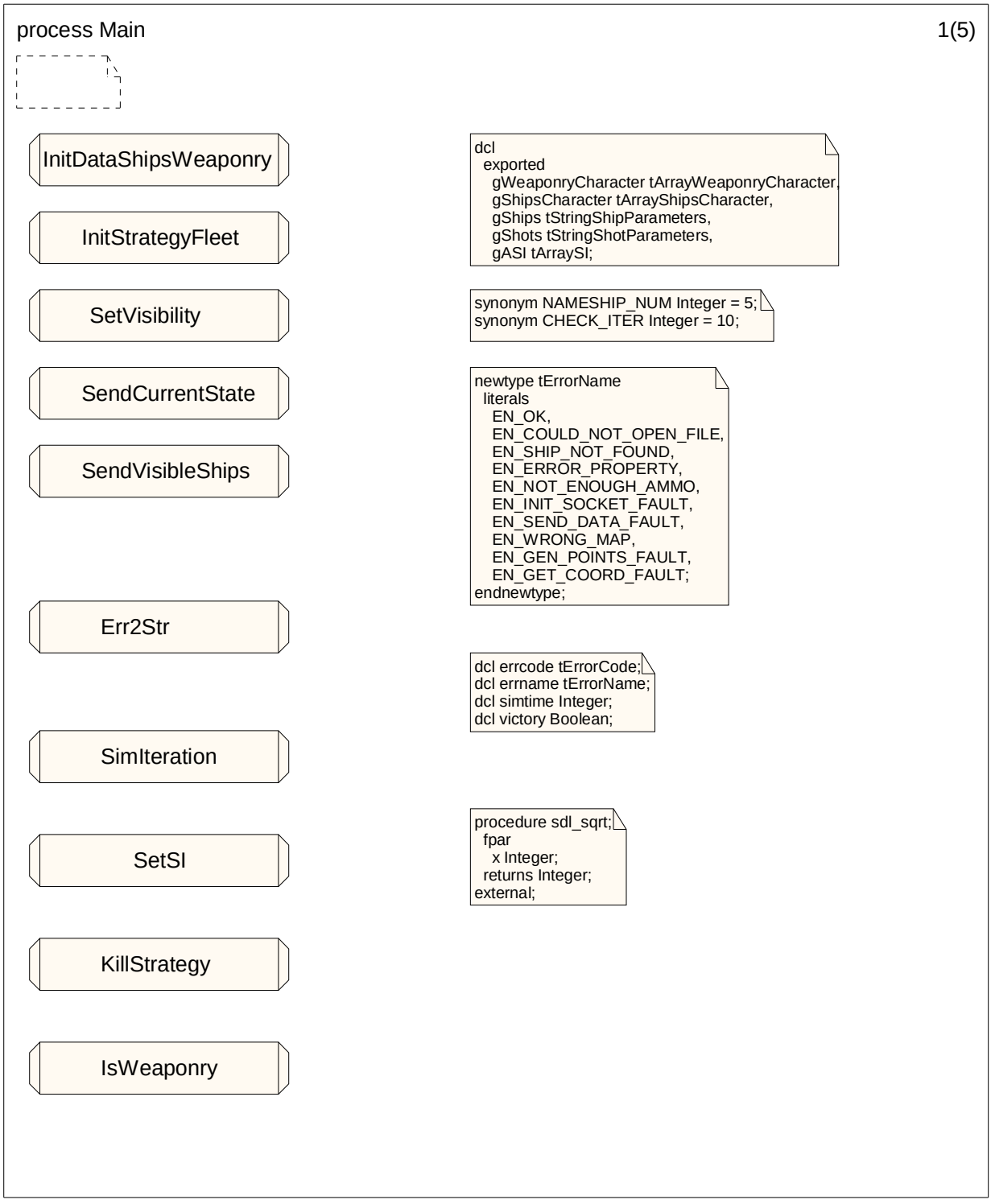

Рисунок 41. Процесс Main (1).

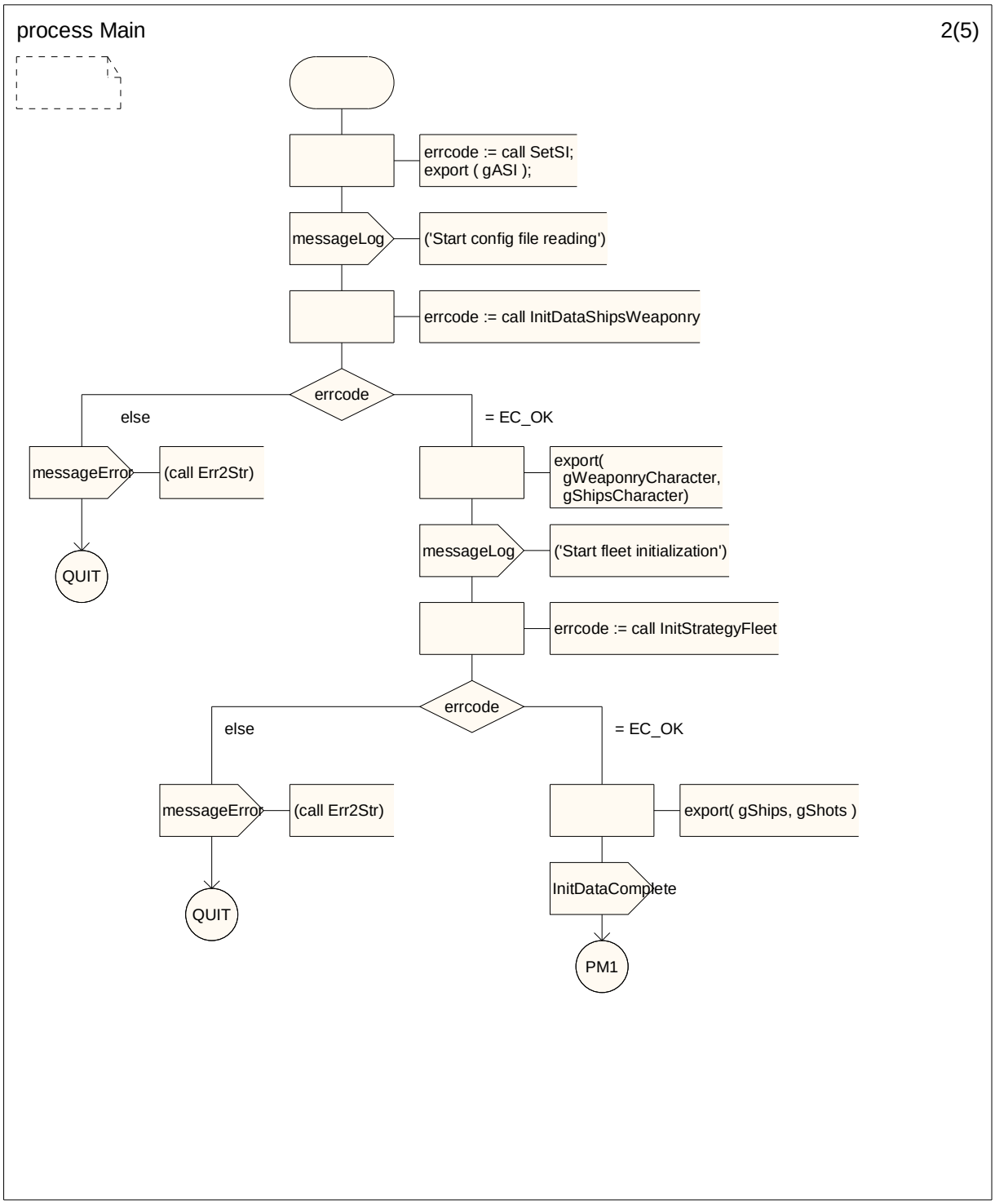

Рисунок 42. Процесс Main (2).

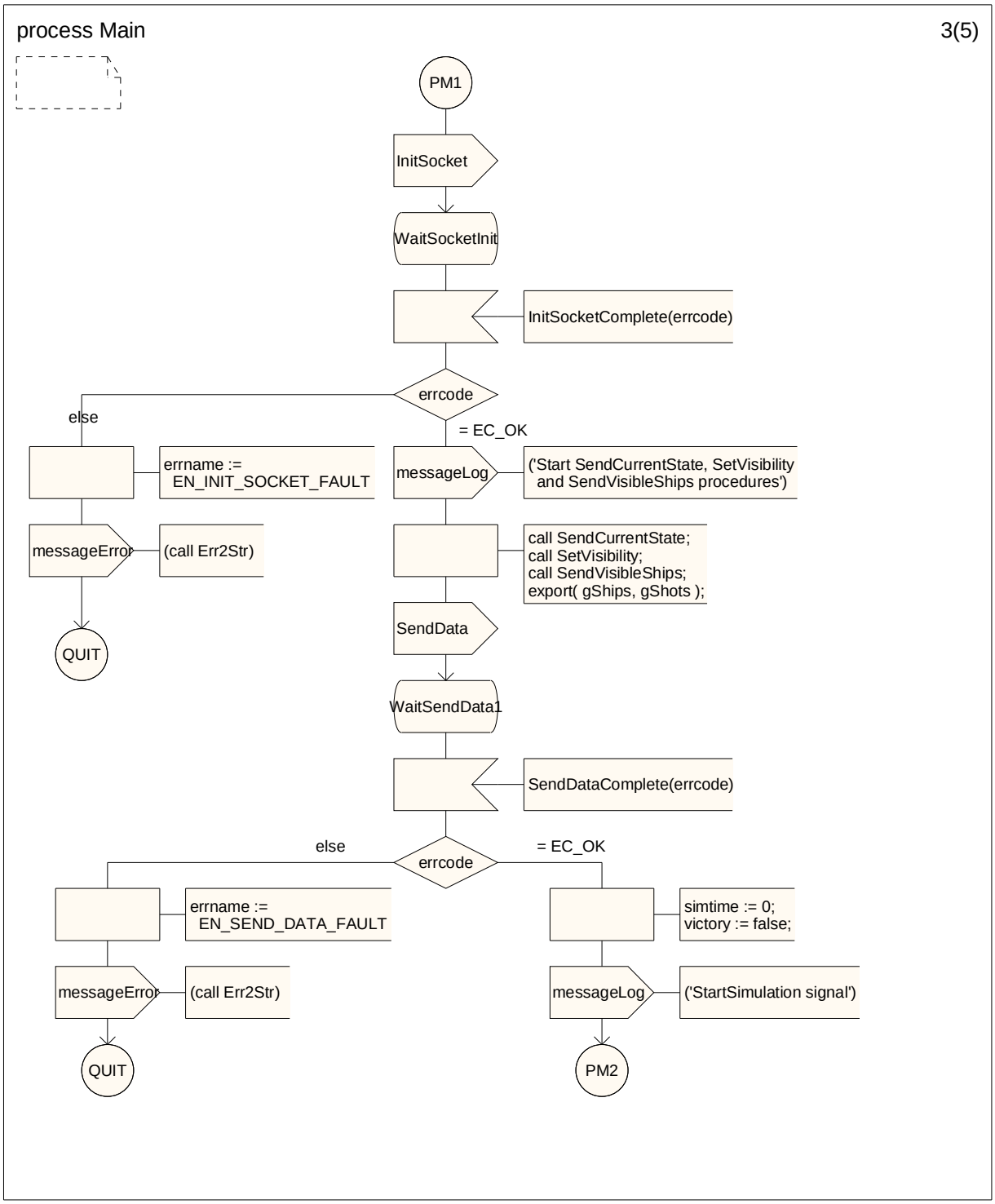

Рисунок 43. Процесс Main (3).

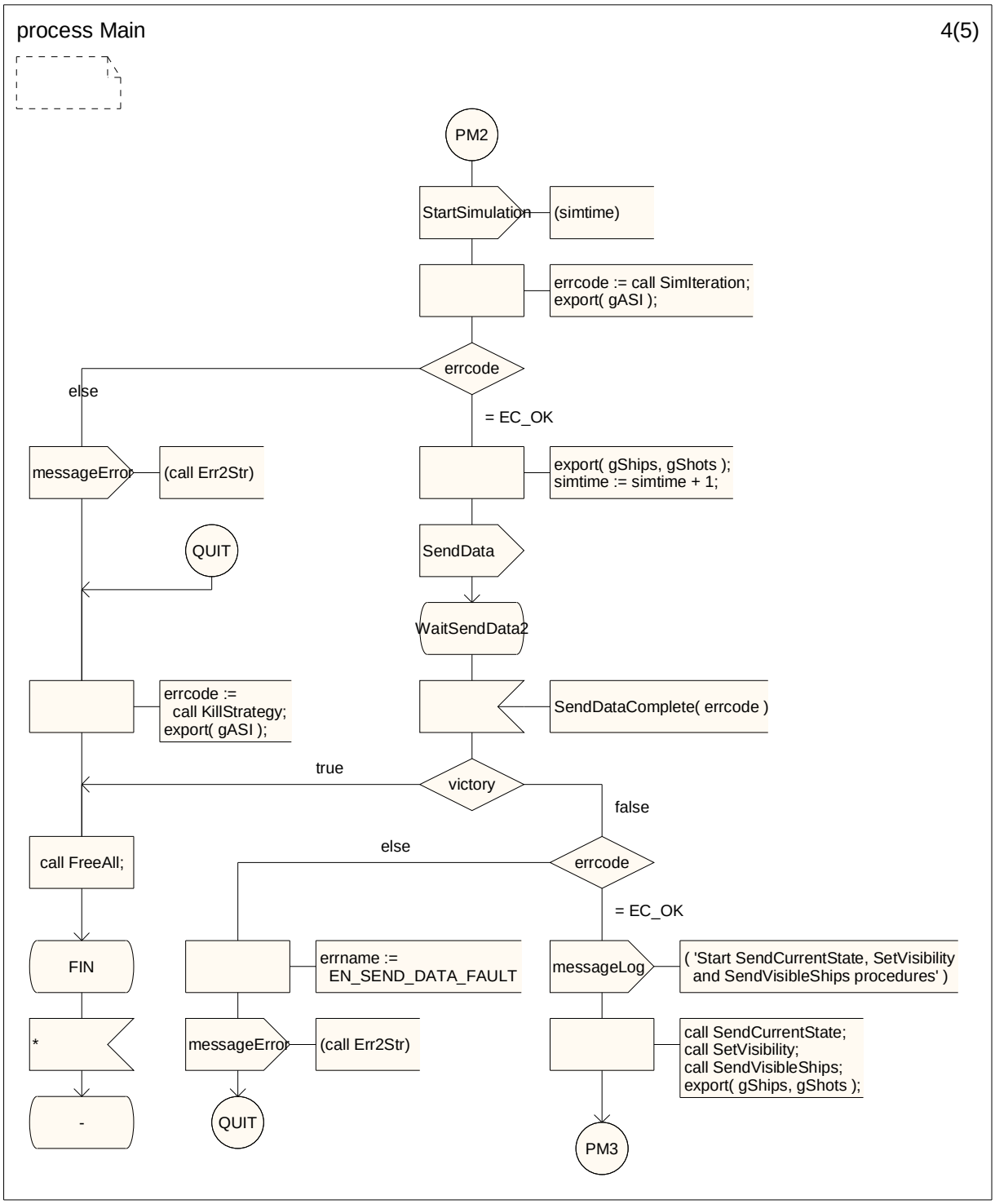

Рисунок 44. Процесс Main (4).

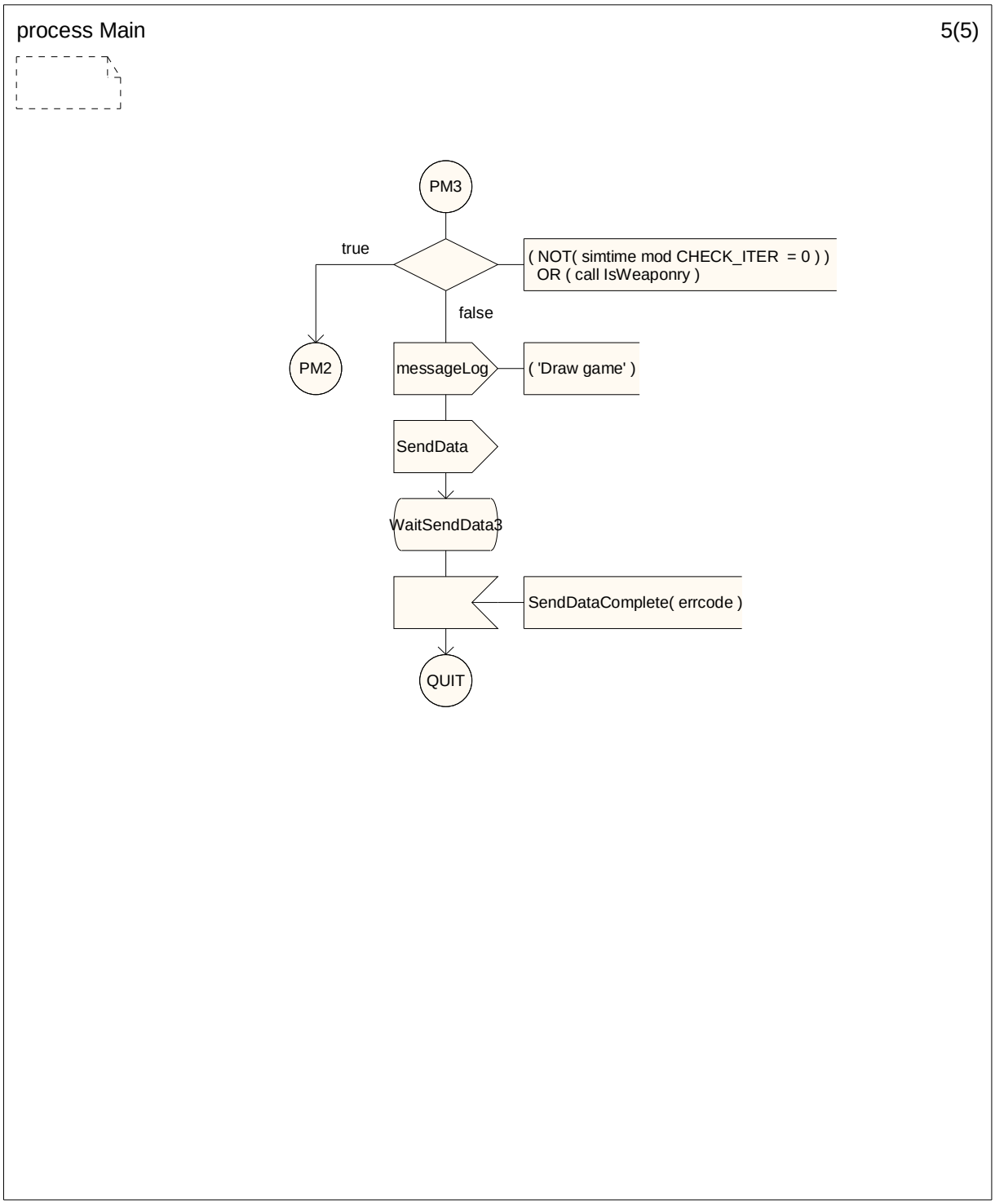

Рисунок 45. Процесс Main (5).

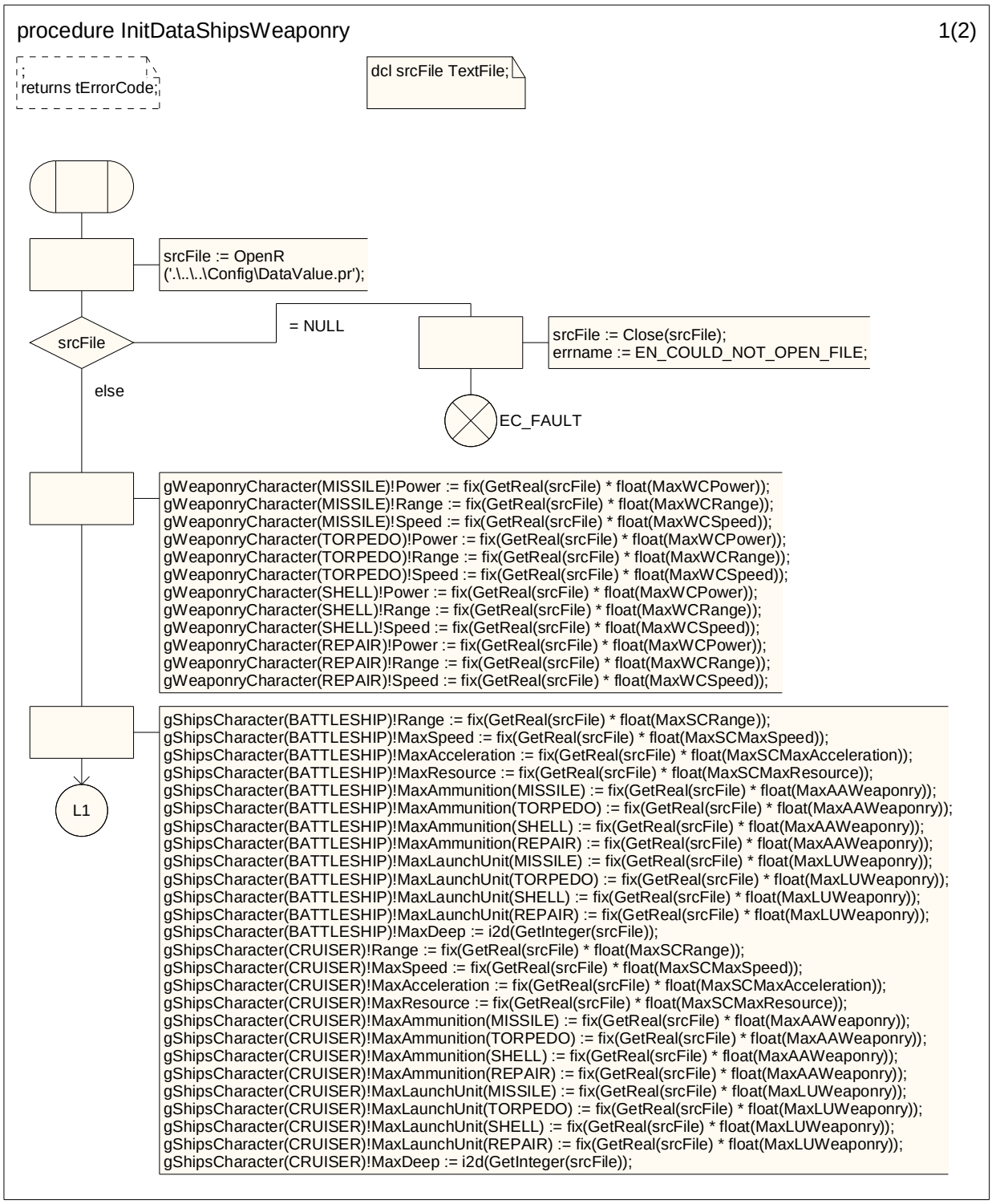

Рисунок 46. Процедура InitDataShipsWeaponry (1).

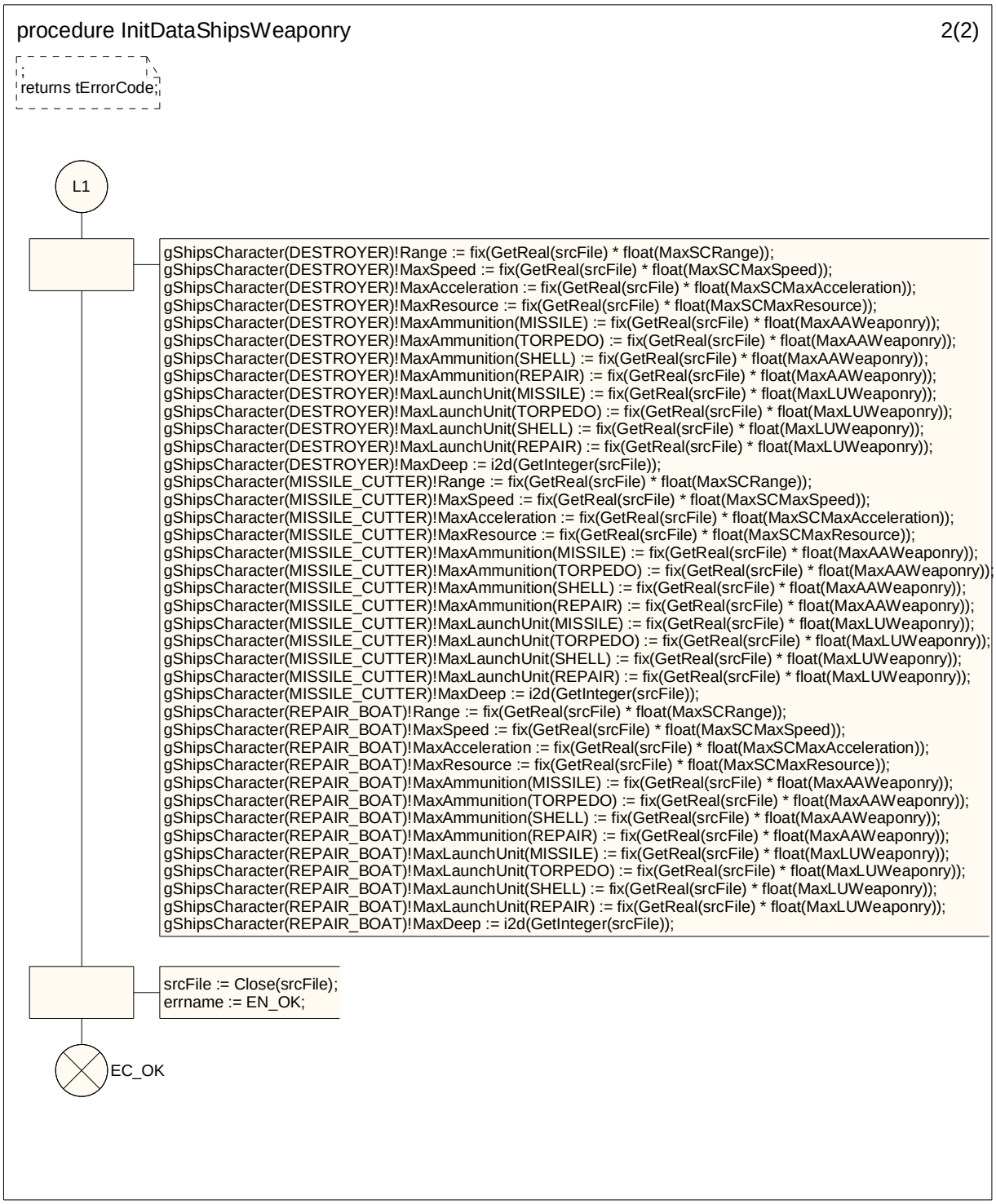

Рисунок 47. Процедура InitDataShipsWeaponry (2).

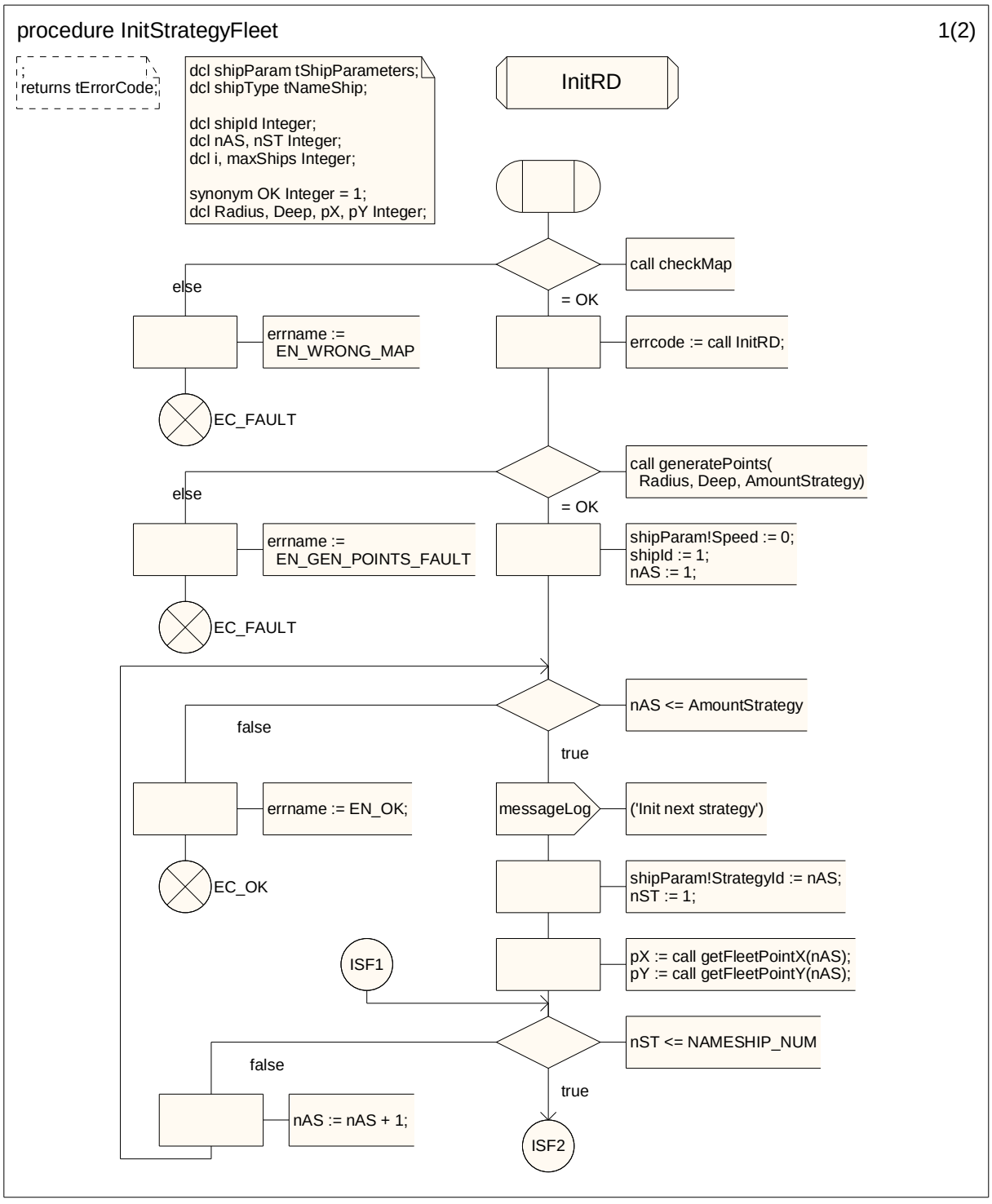

Рисунок 48. Процедура InitStrategyFleet (1).

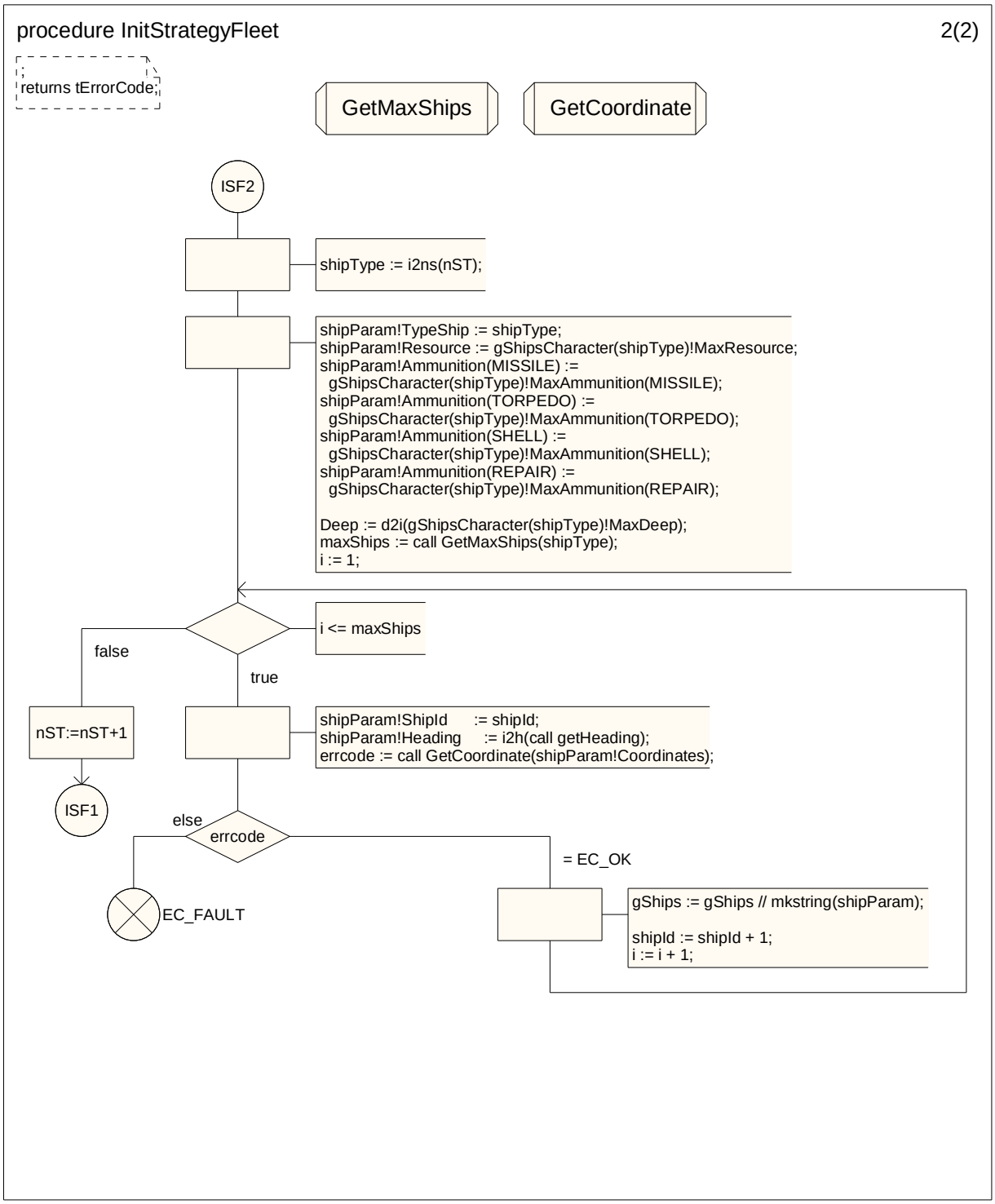

Рисунок 49. Процедура InitStrategyFleet (2).

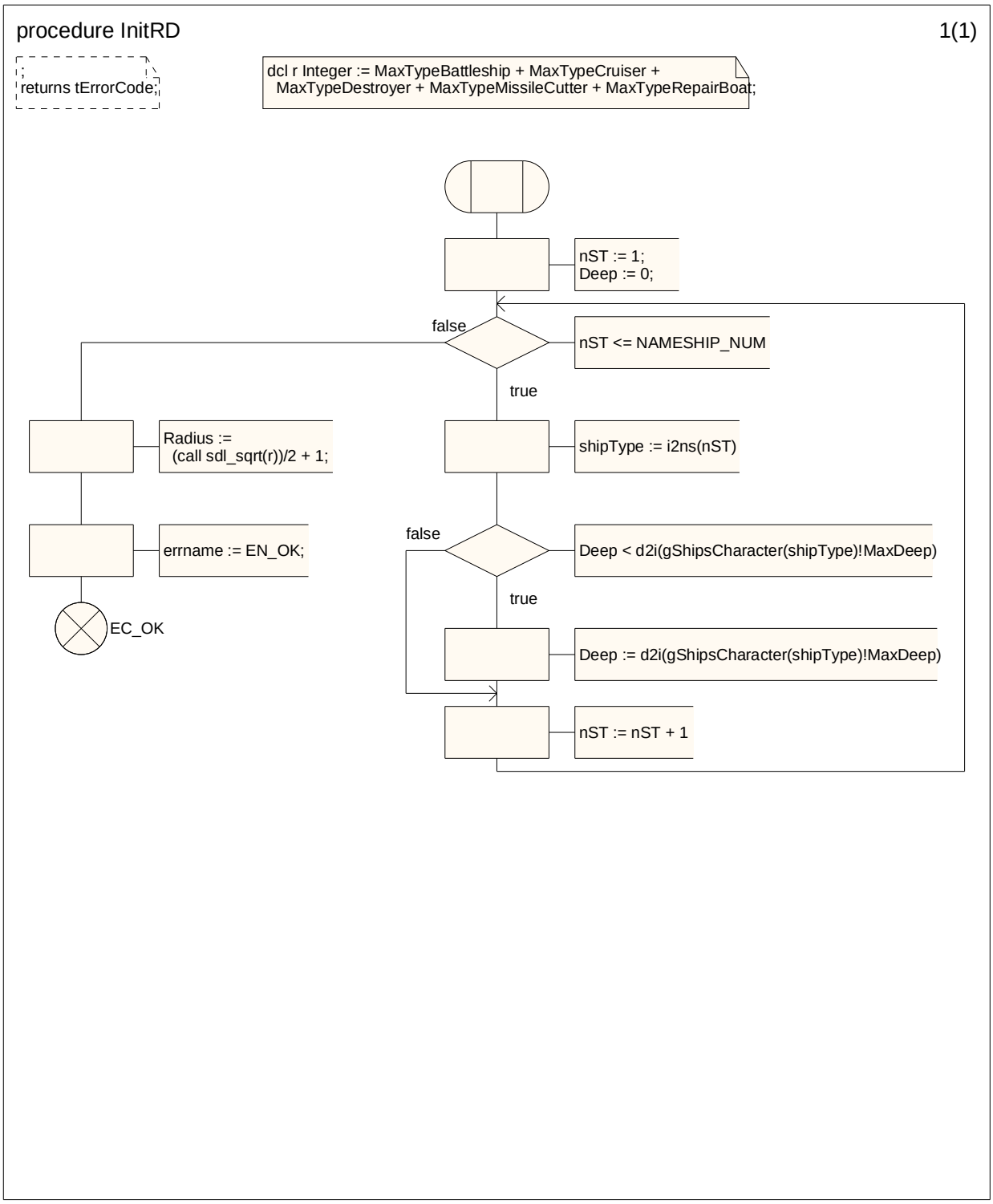

Рисунок 50. Процедура InitRD.

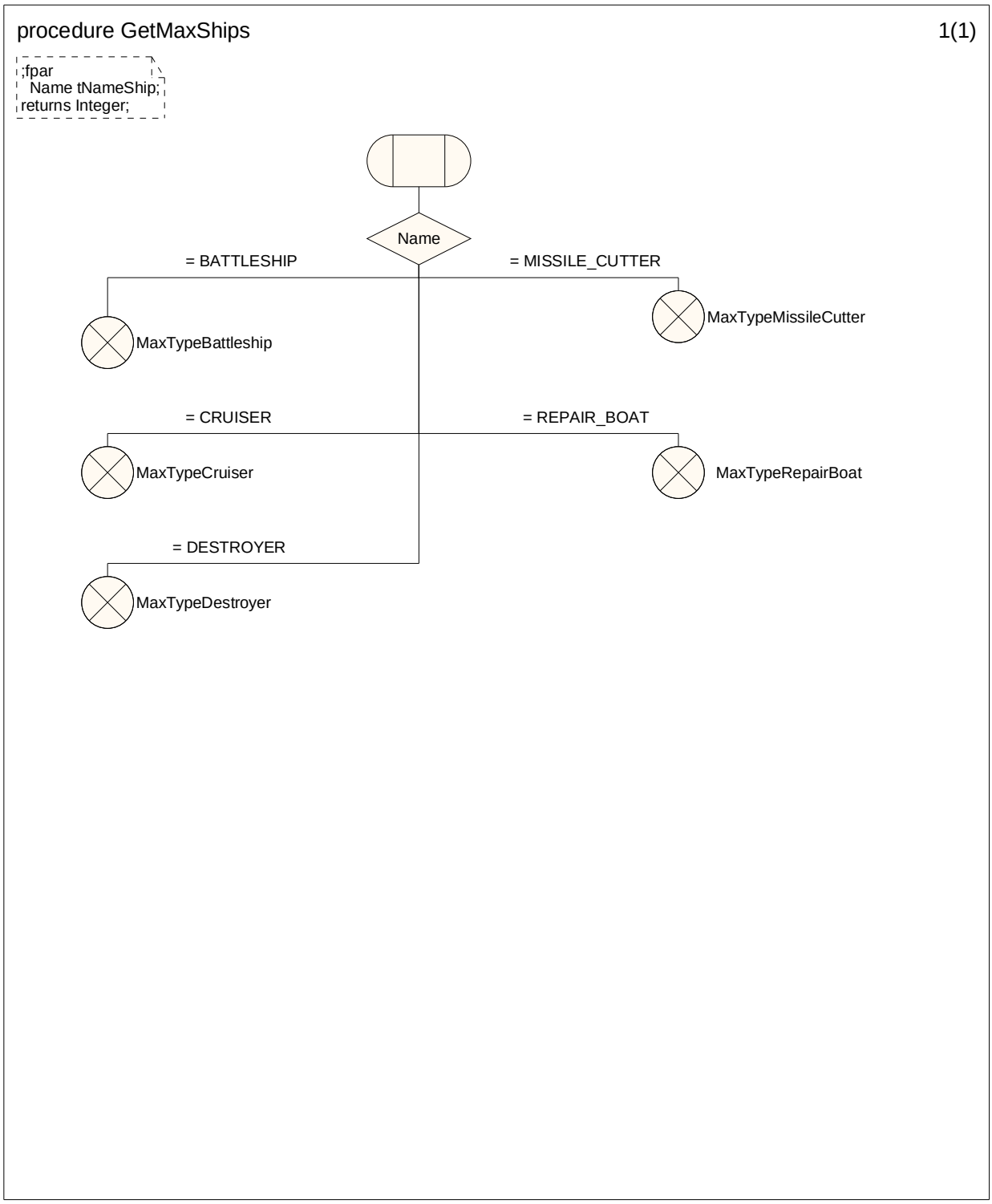

Рисунок 51. Процедура GetMaxShips.

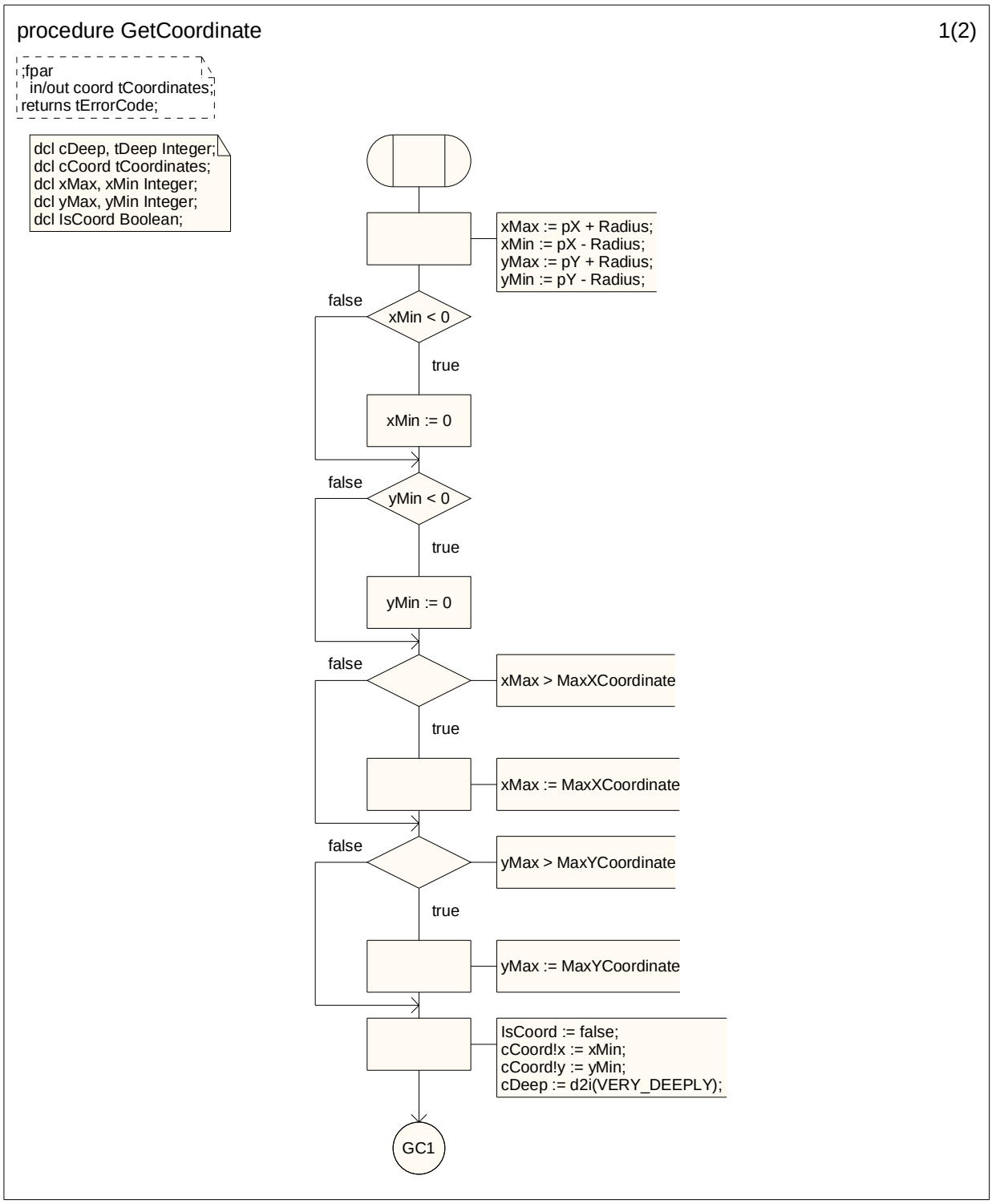

Рисунок 52. Процедура GetCoordinate (1).

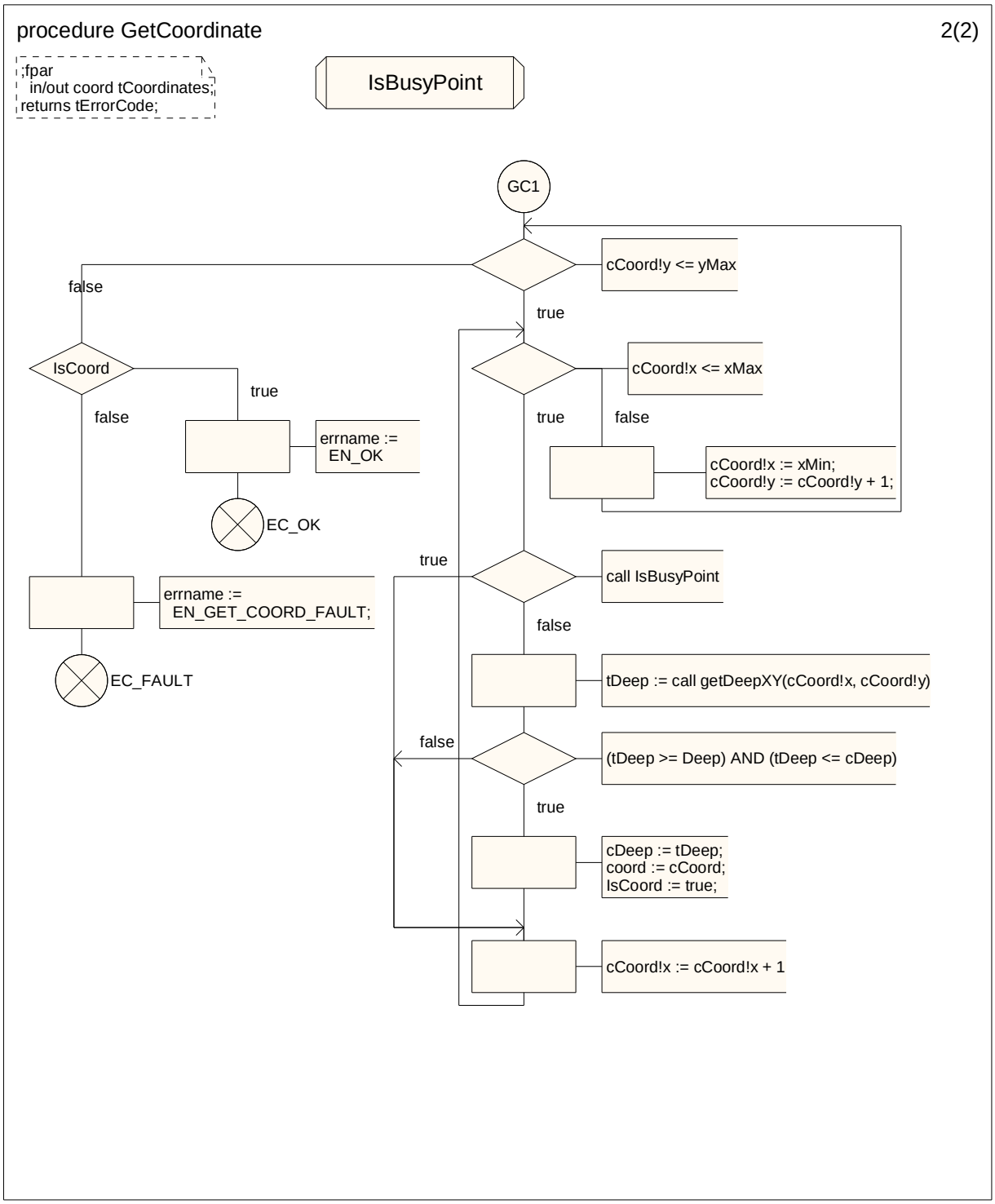

Рисунок 53. Процедура GetCoordinate (2).

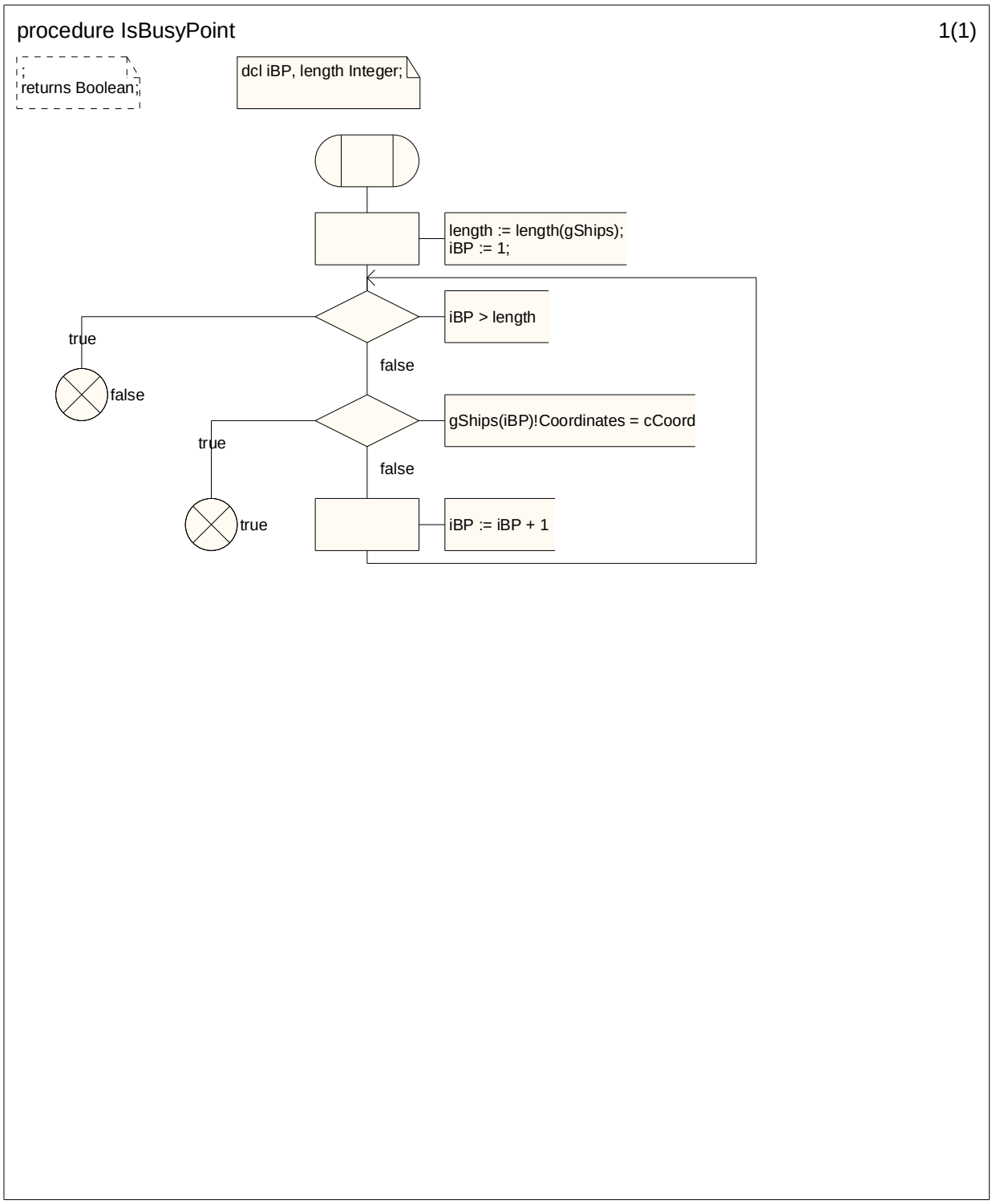

Рисунок 54. Процедура IsBusyPoint.

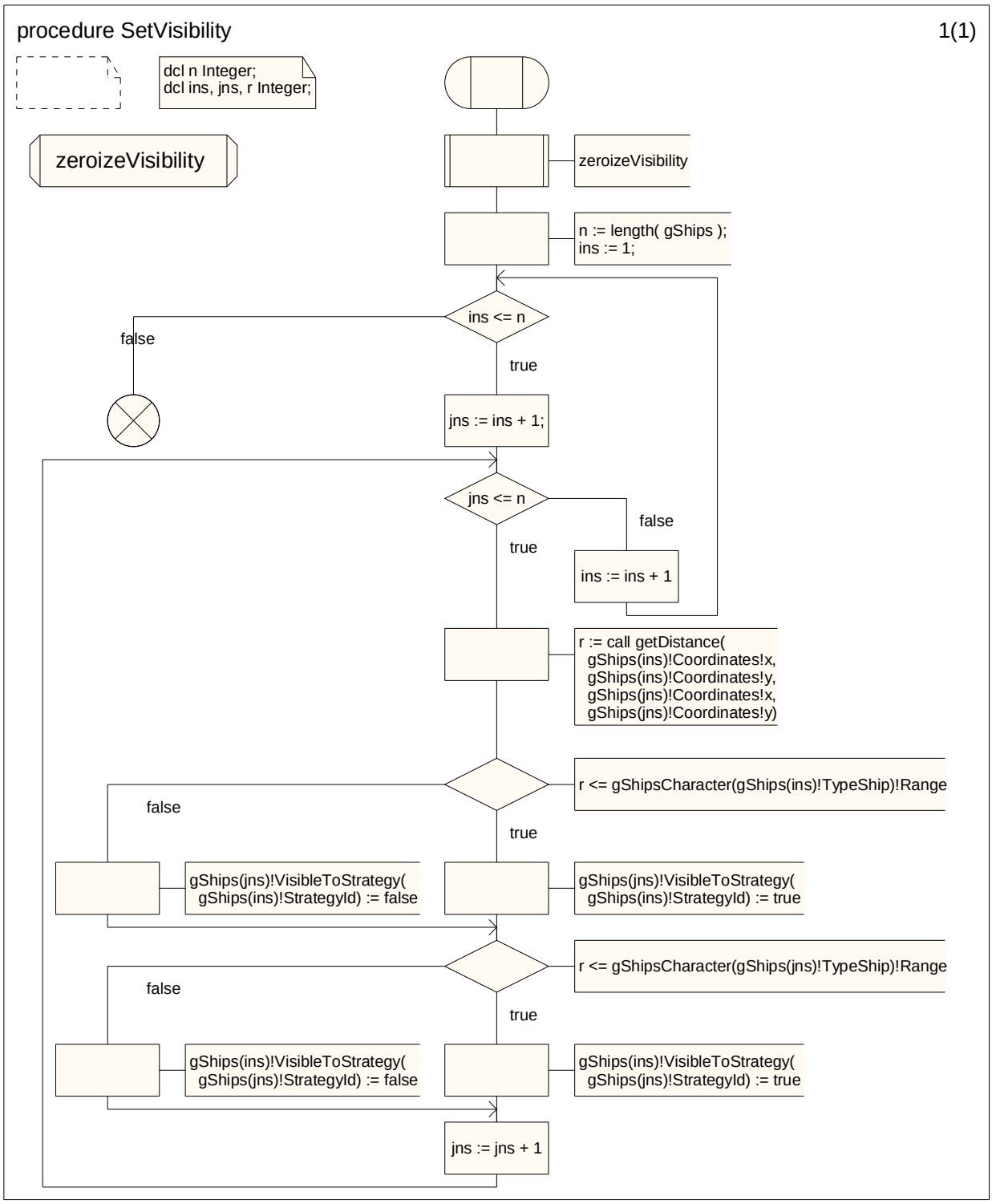

Рисунок 55. Процедура SetVisibility.

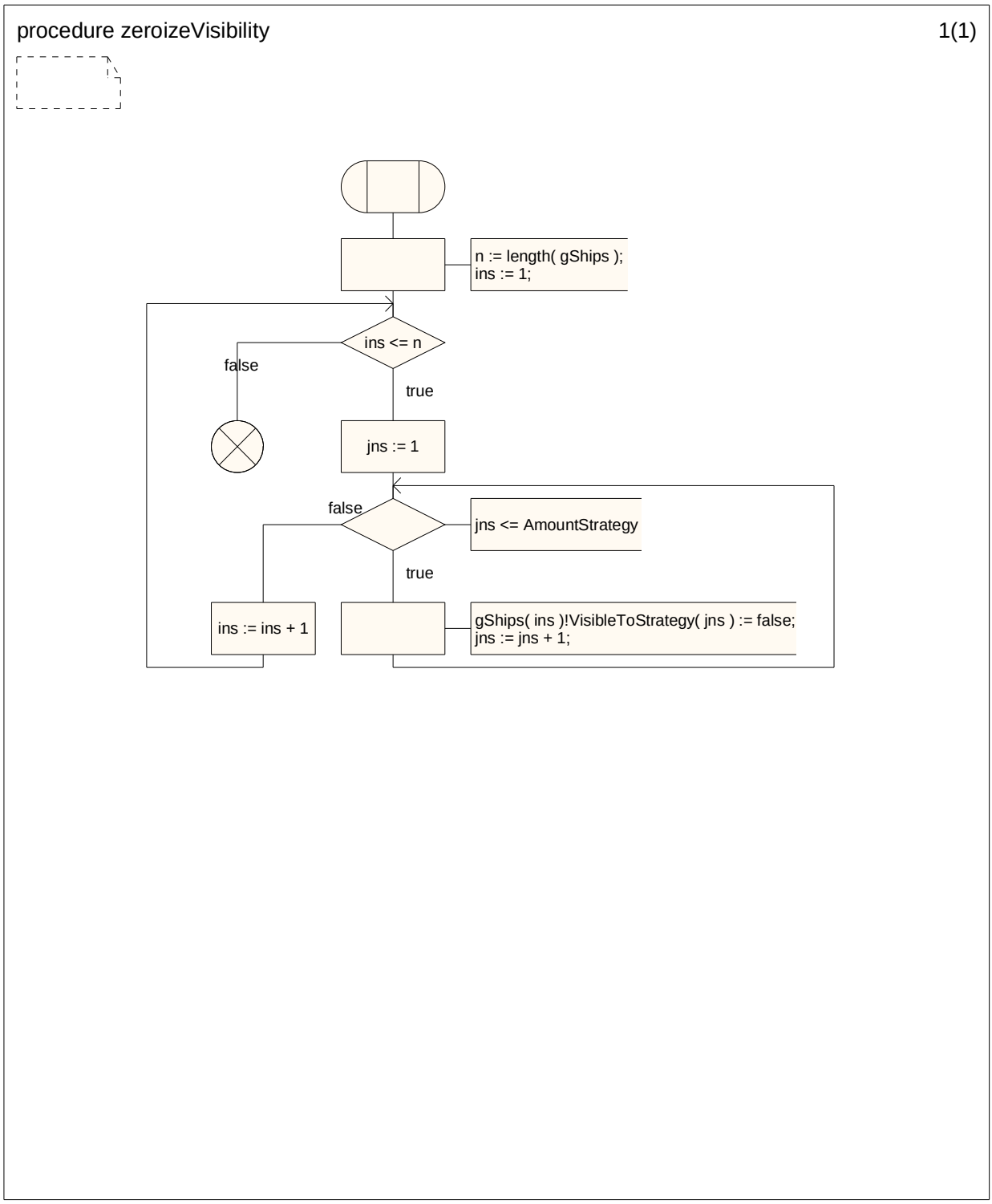

Рисунок 56. Процедура zeroizeVisibility.

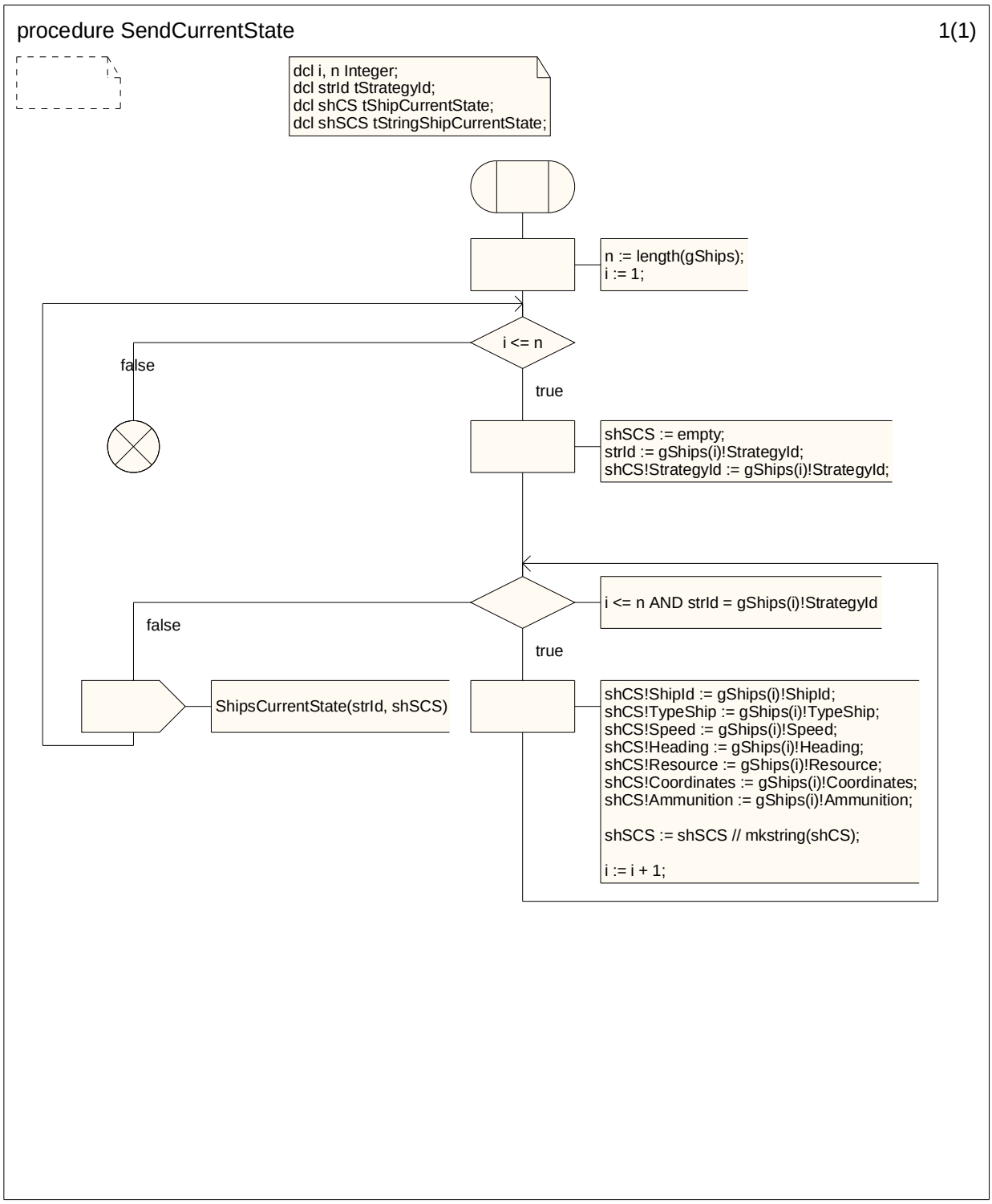

Рисунок 57. Процедура SendCurrentState.

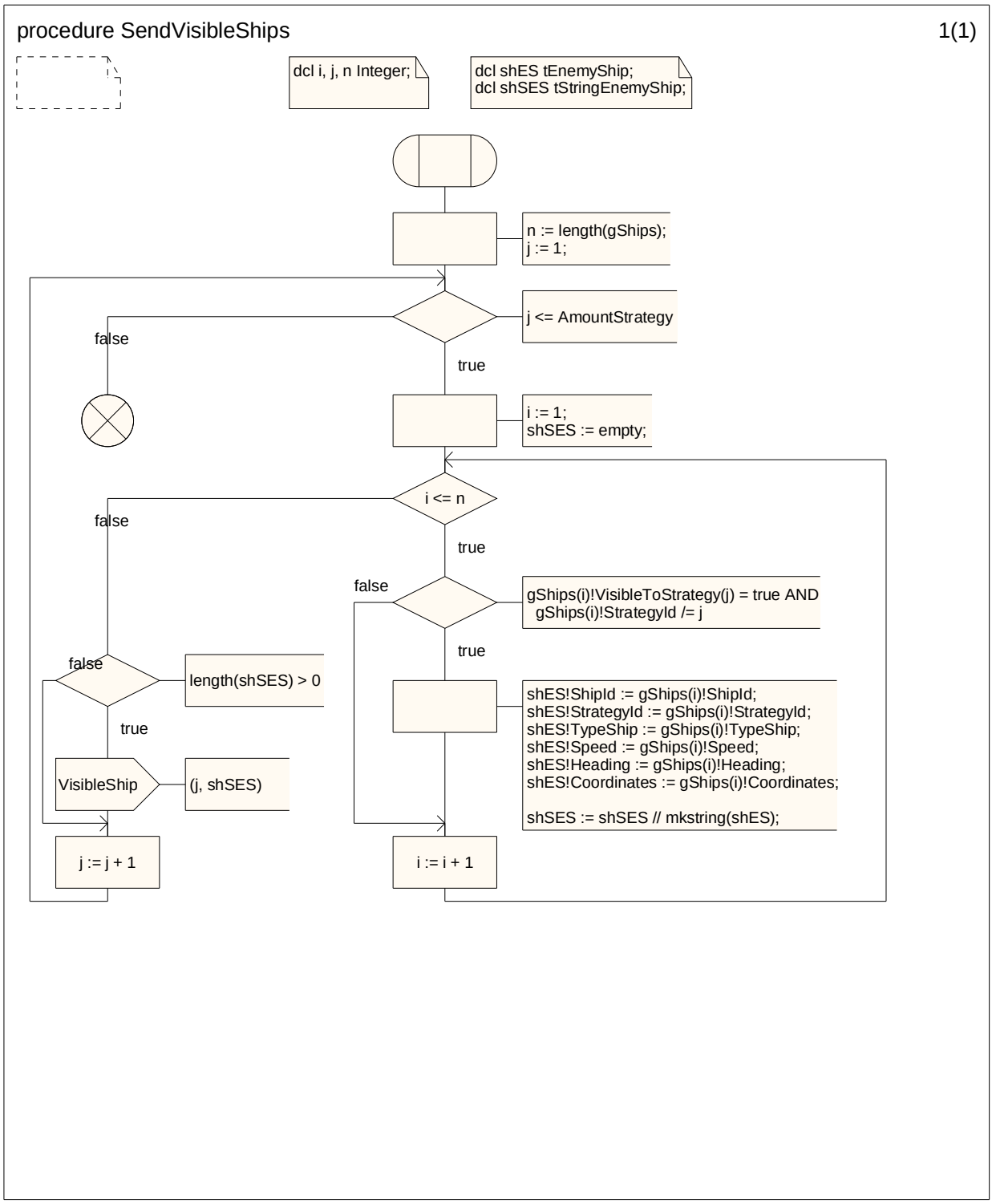

Рисунок 58. Процедура SendVisibleShips.

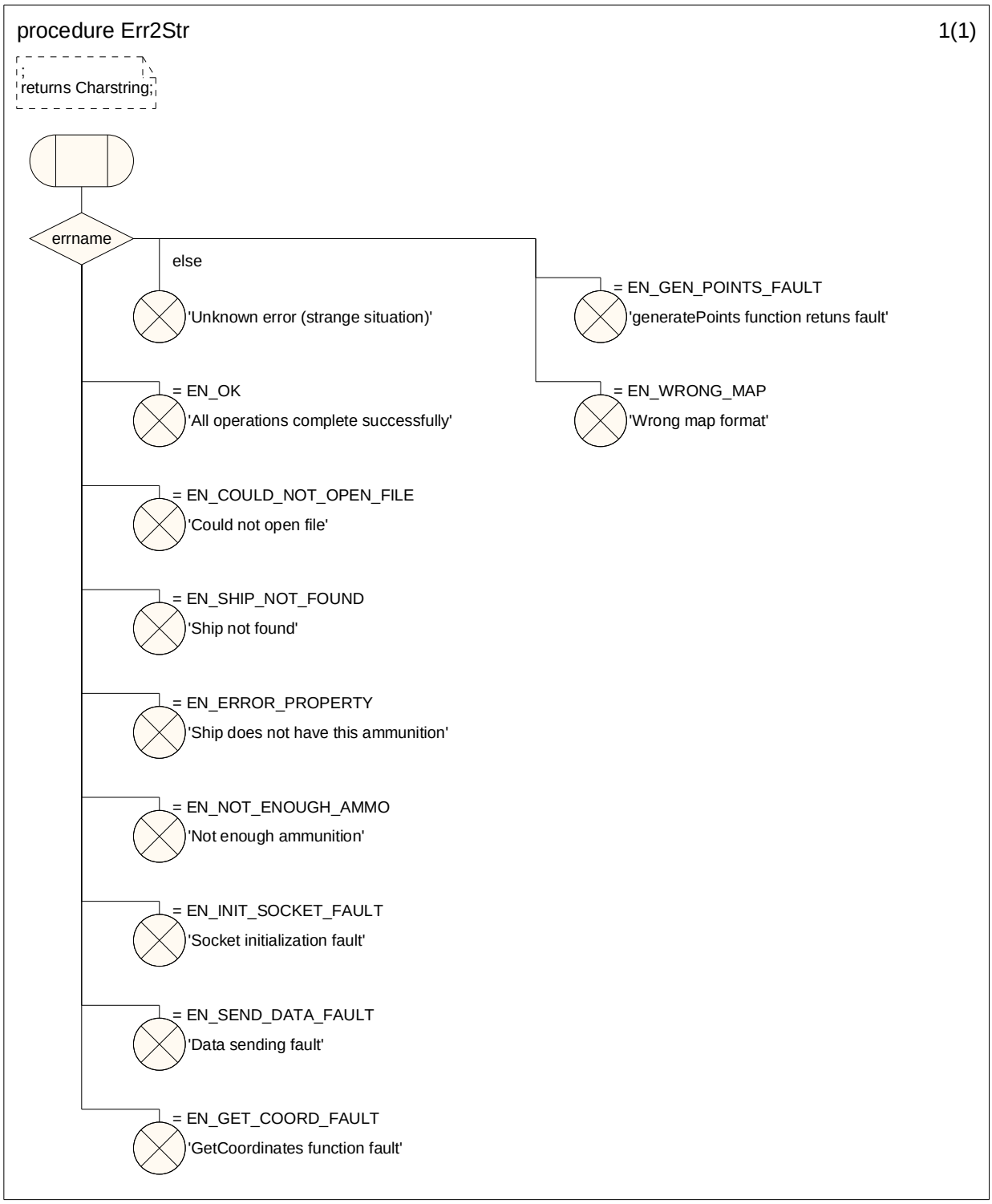

Рисунок 59. Процедура Err2Str.

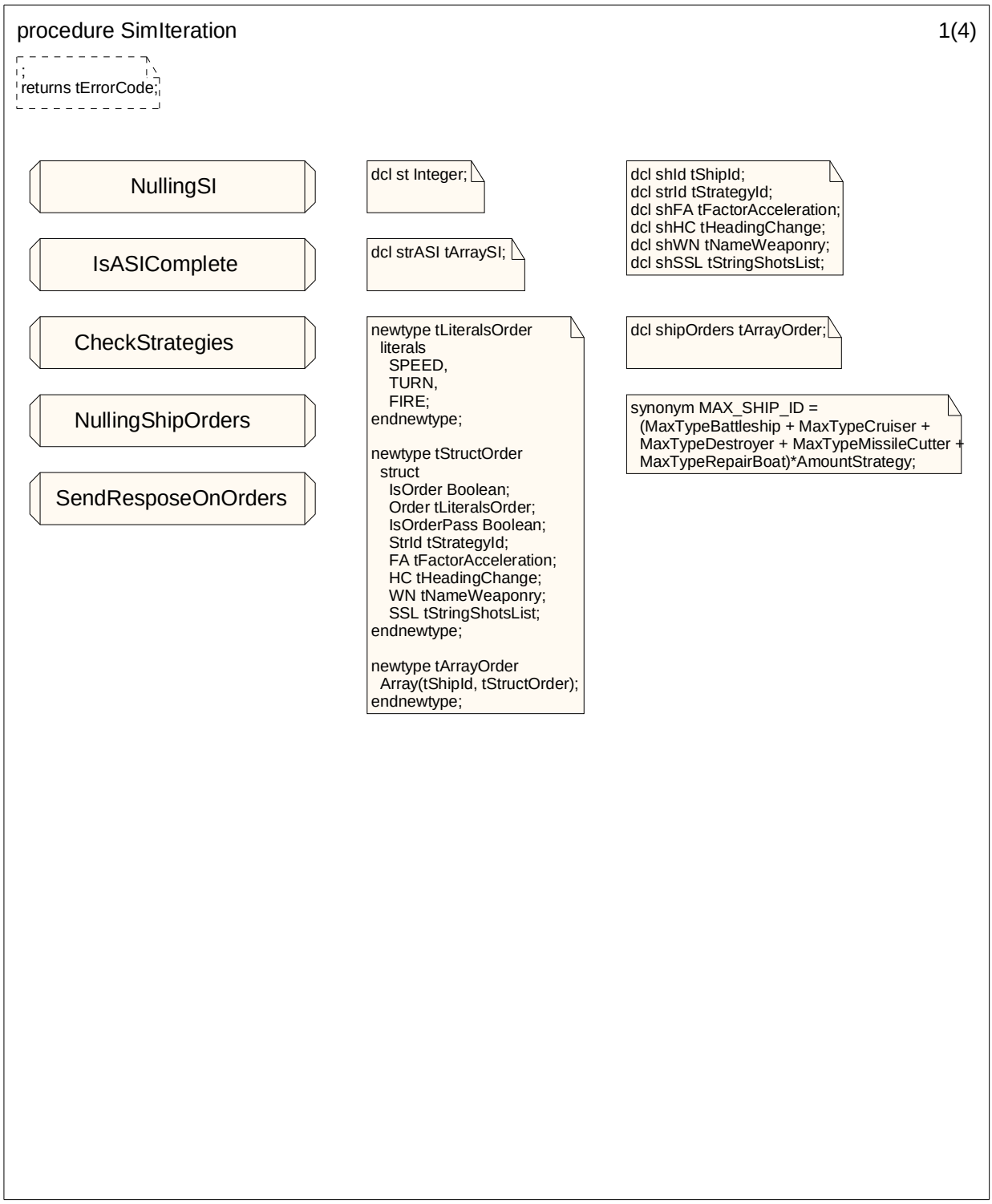

Рисунок 60. Процедура SimIteration (1).

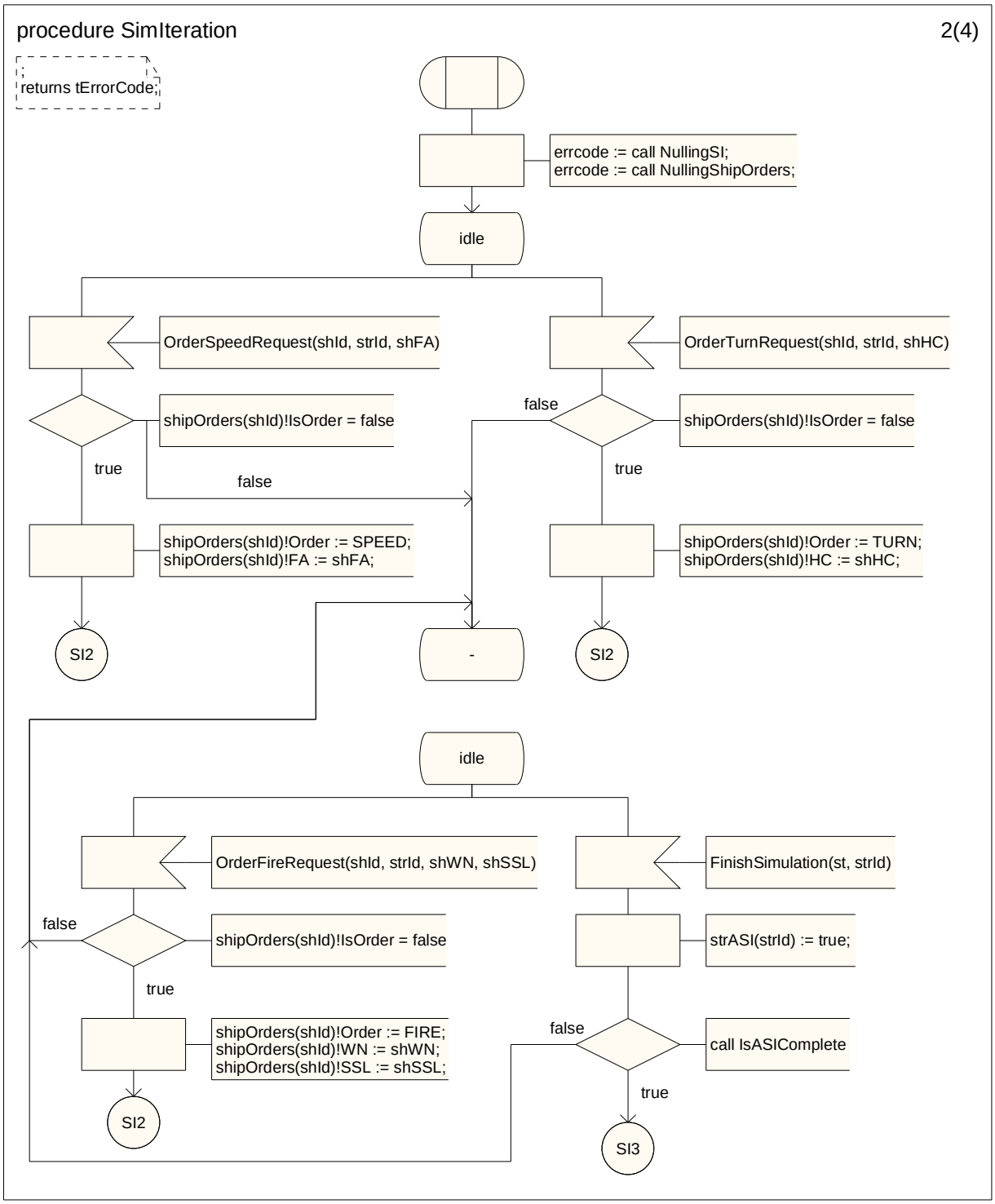

Рисунок 61. Процедура SimIteration (2).

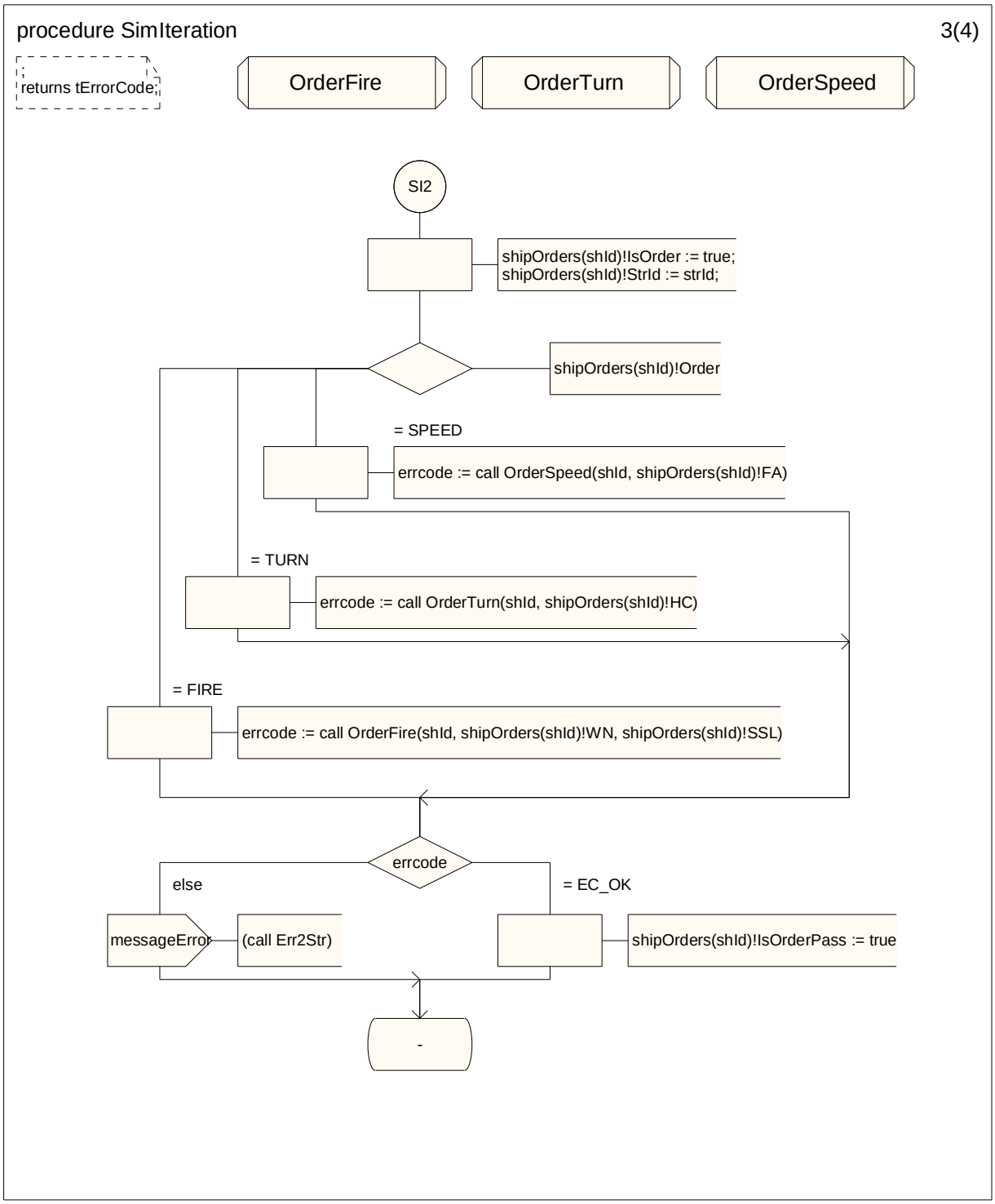

Рисунок 62. Процедура SimIteration (3).

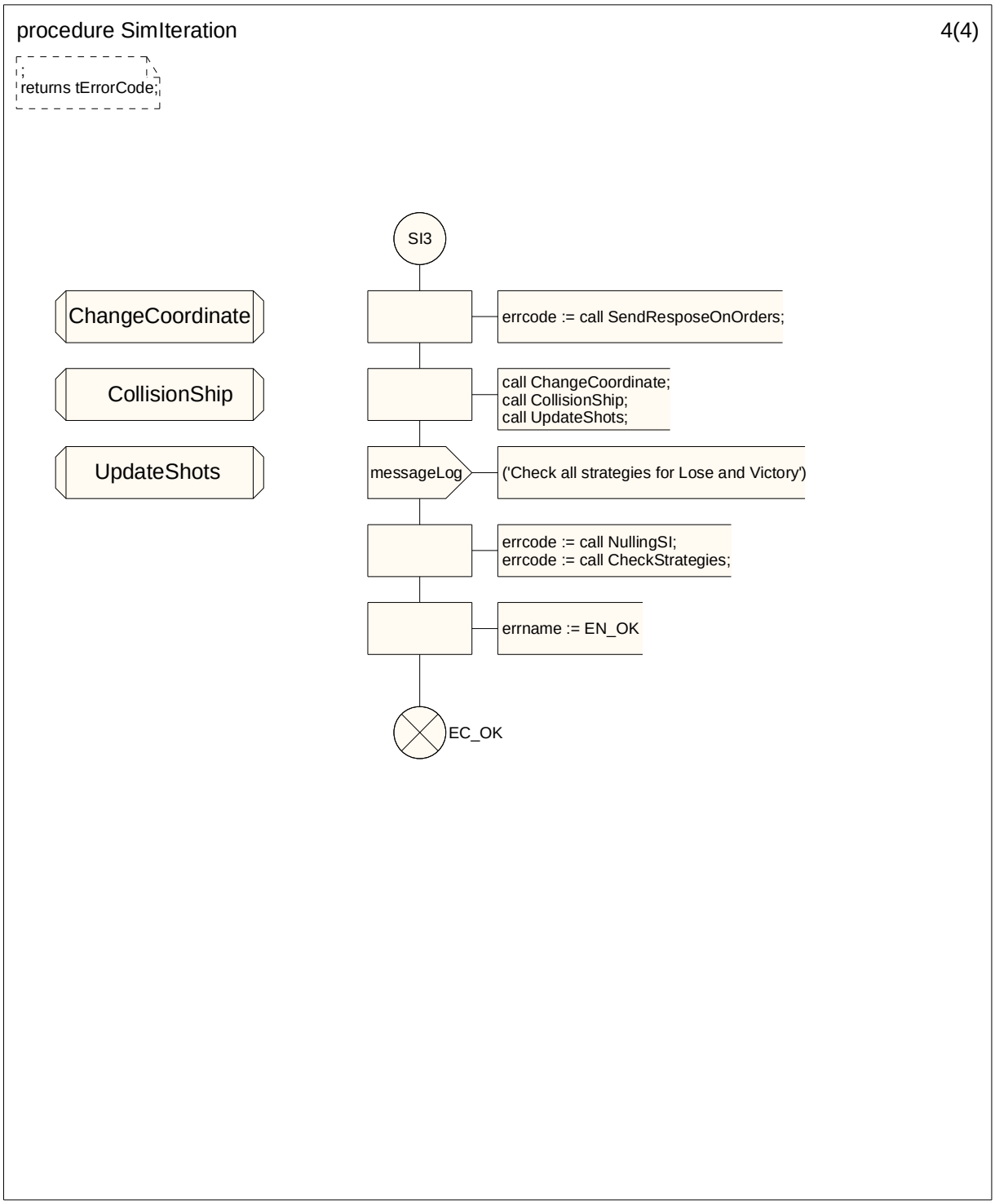

Рисунок 63. Процедура SimIteration (4).

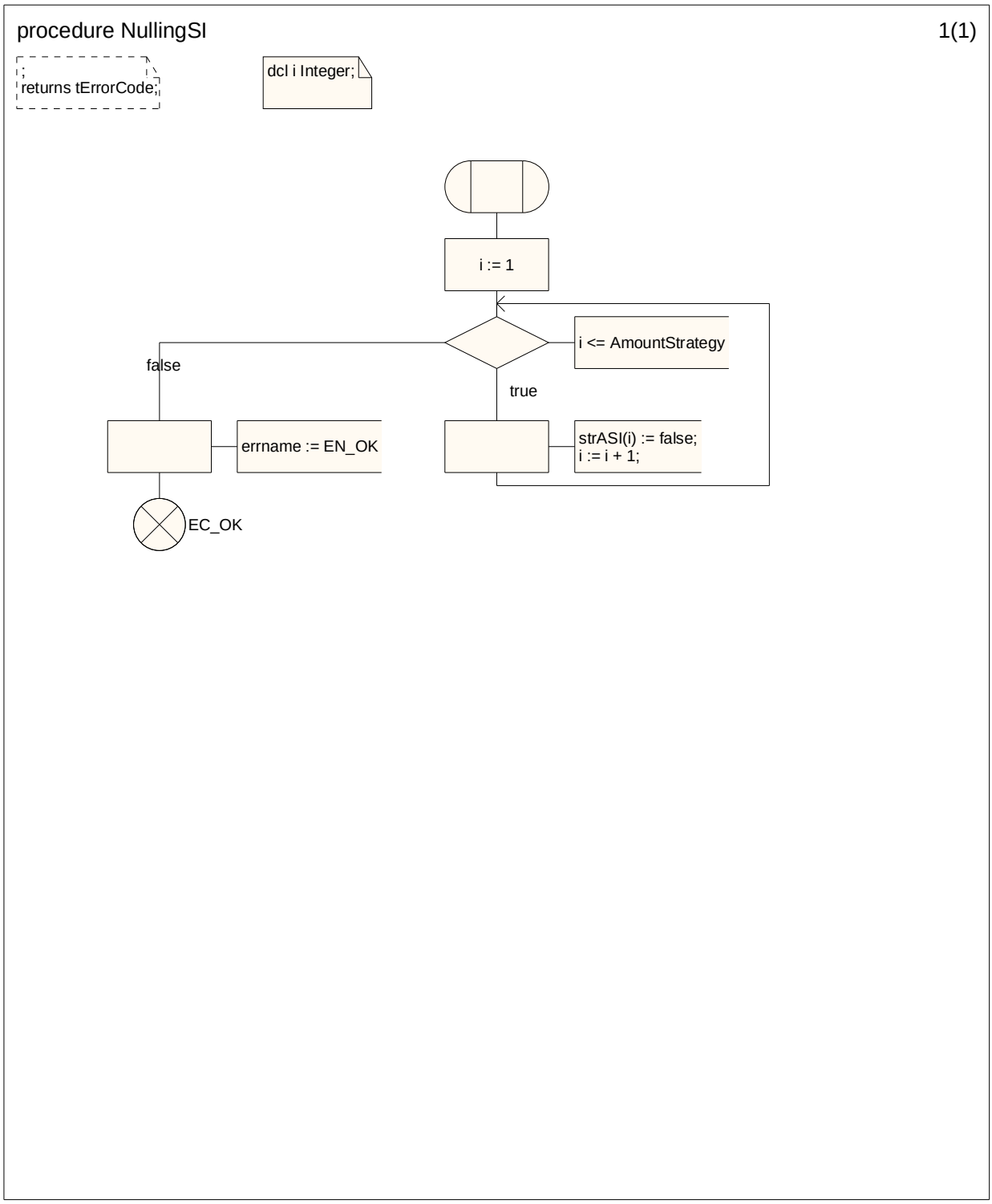

Рисунок 64. Процедура NullingSI.

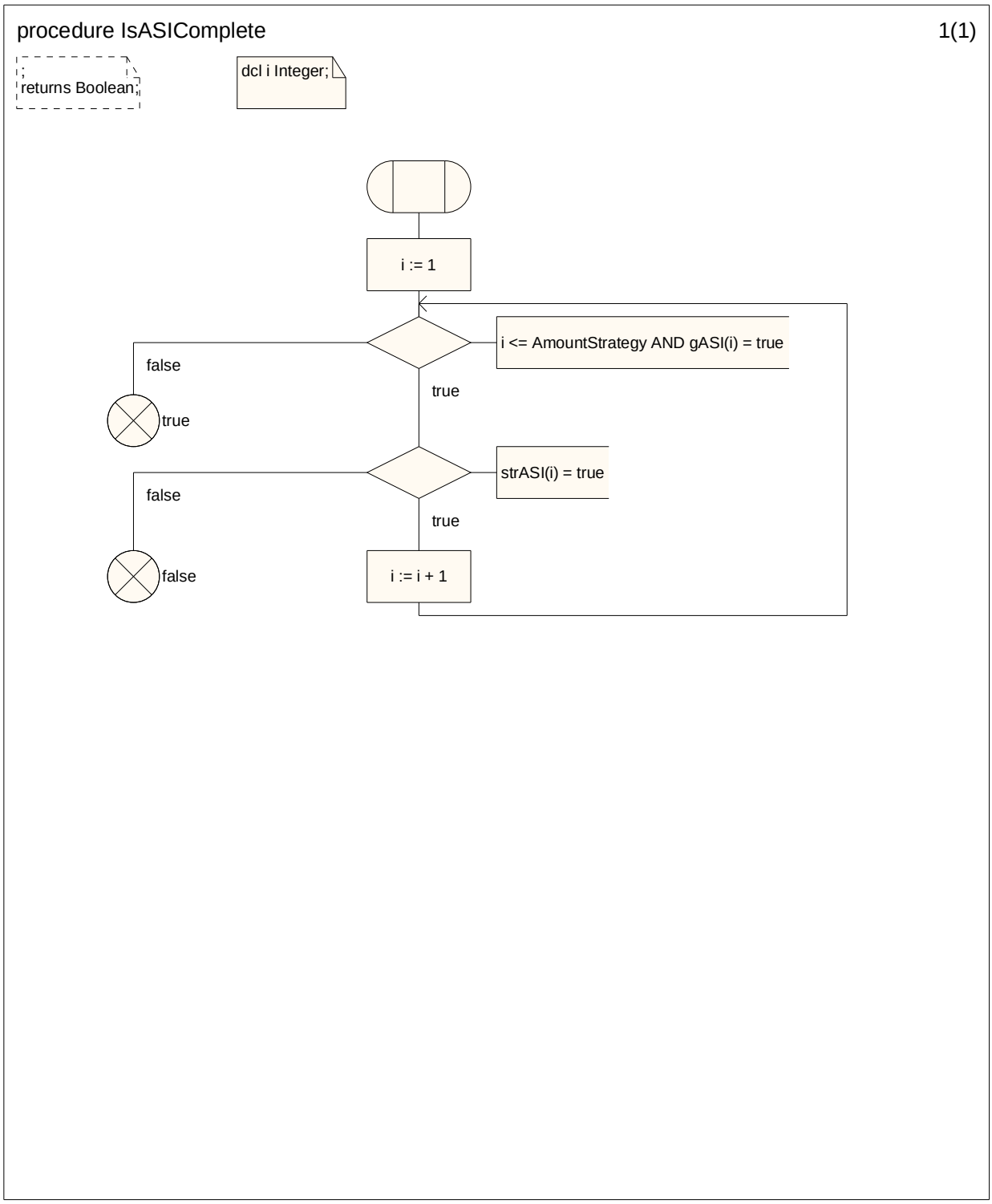

Рисунок 65. Процедура IsASIComplete.

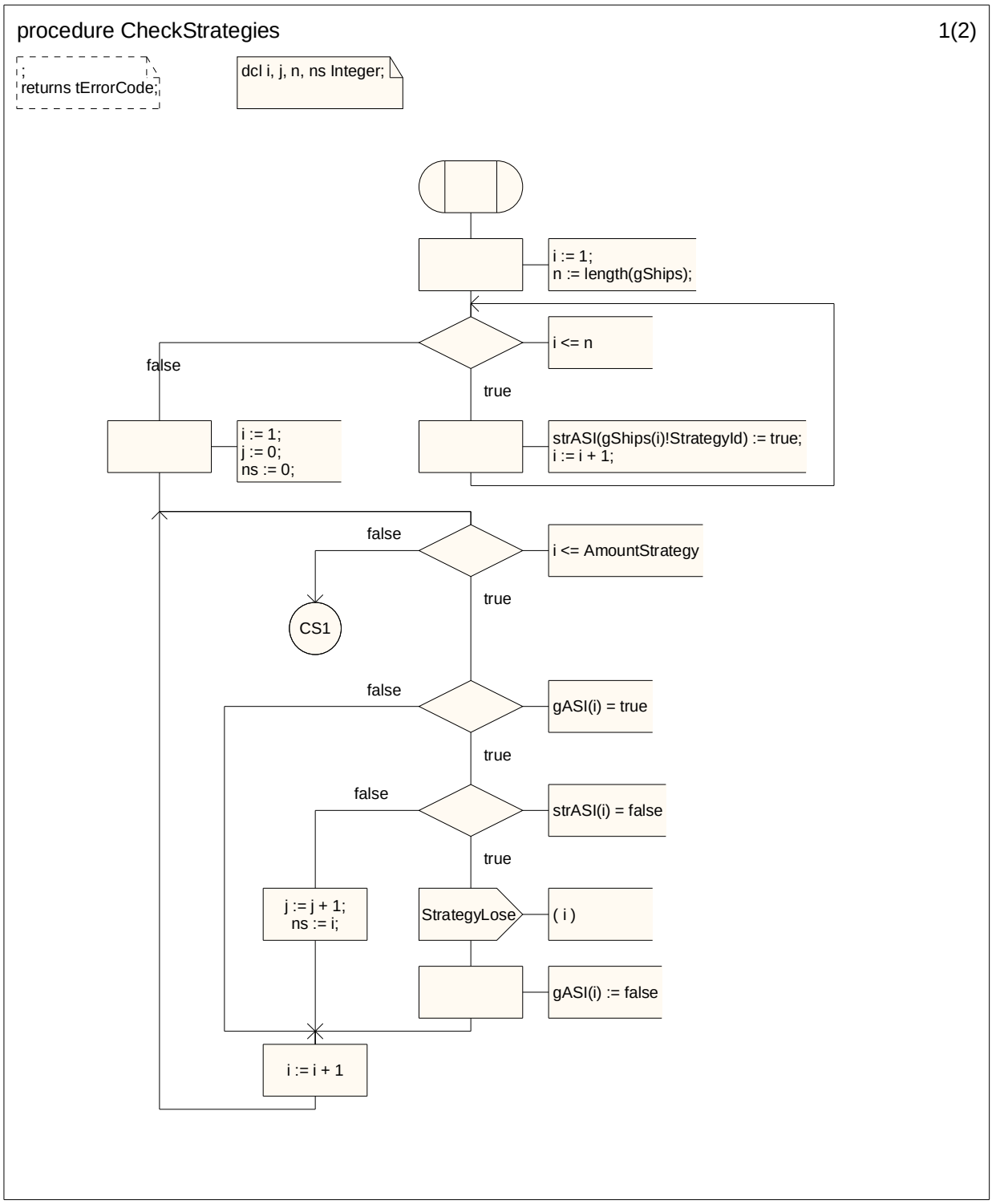

Рисунок 66. Процедура CheckStrategies (1).

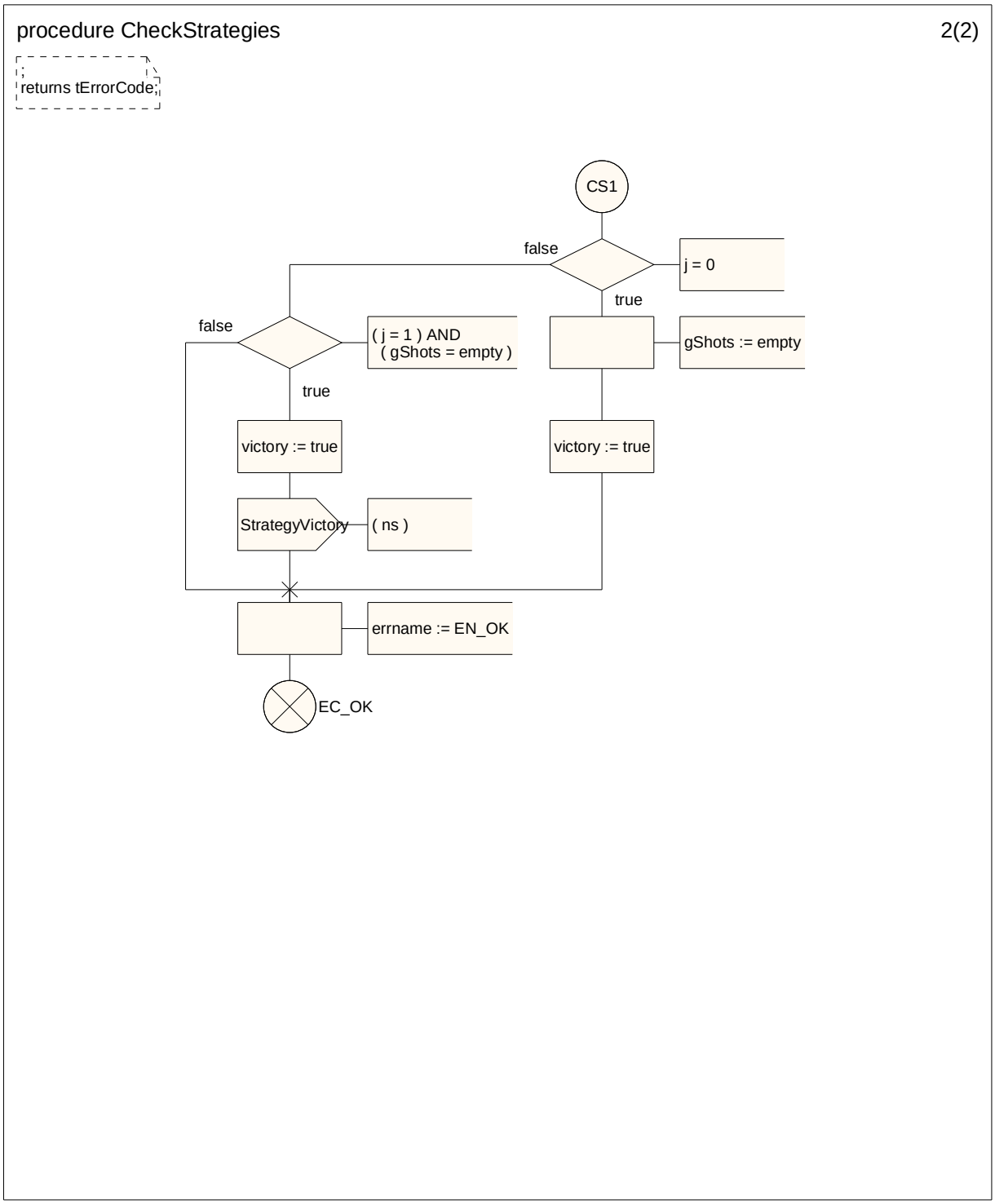

Рисунок 67. Процедура CheckStrategies (2).

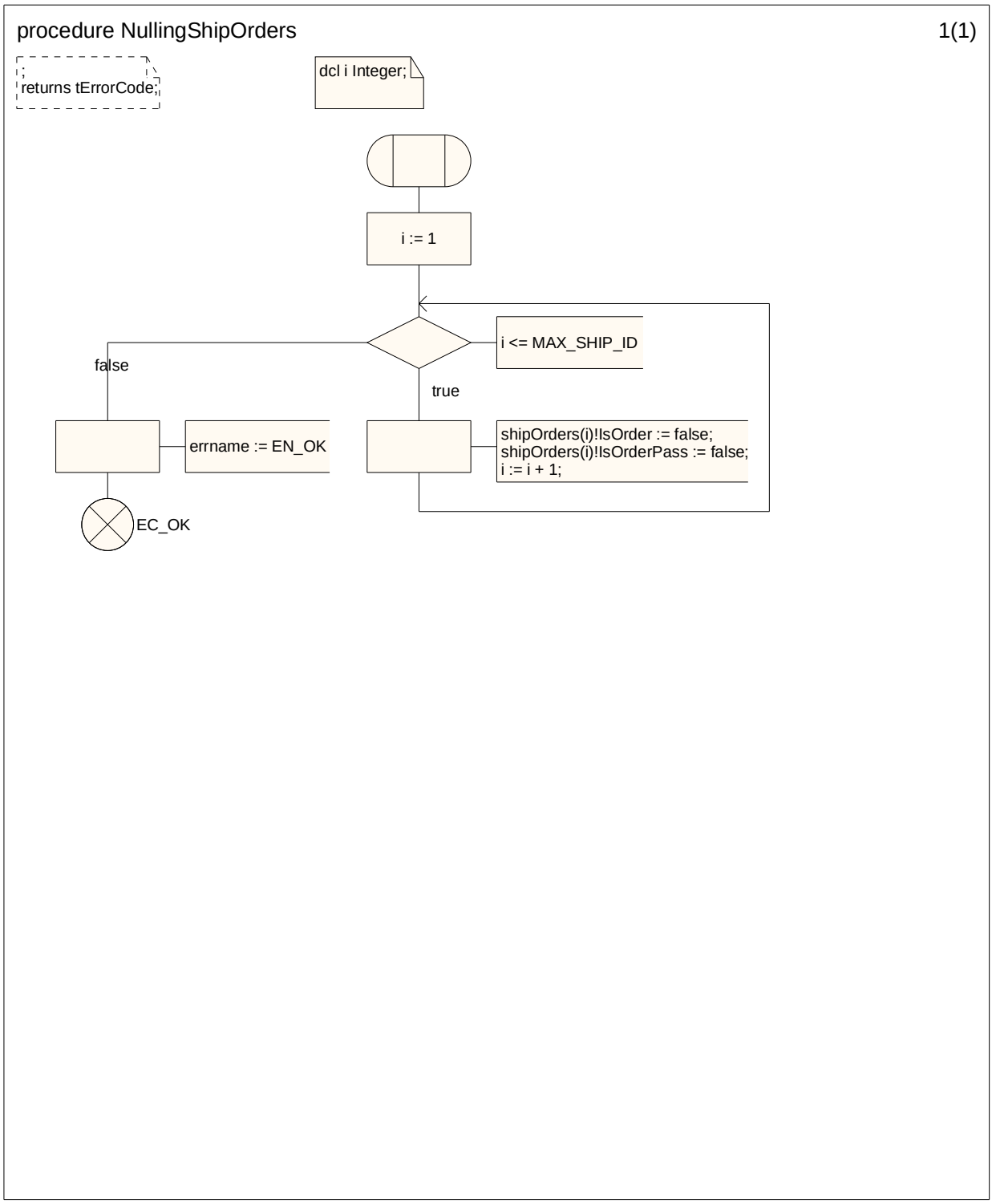

Рисунок 68. Процедура NullingShipOrders.

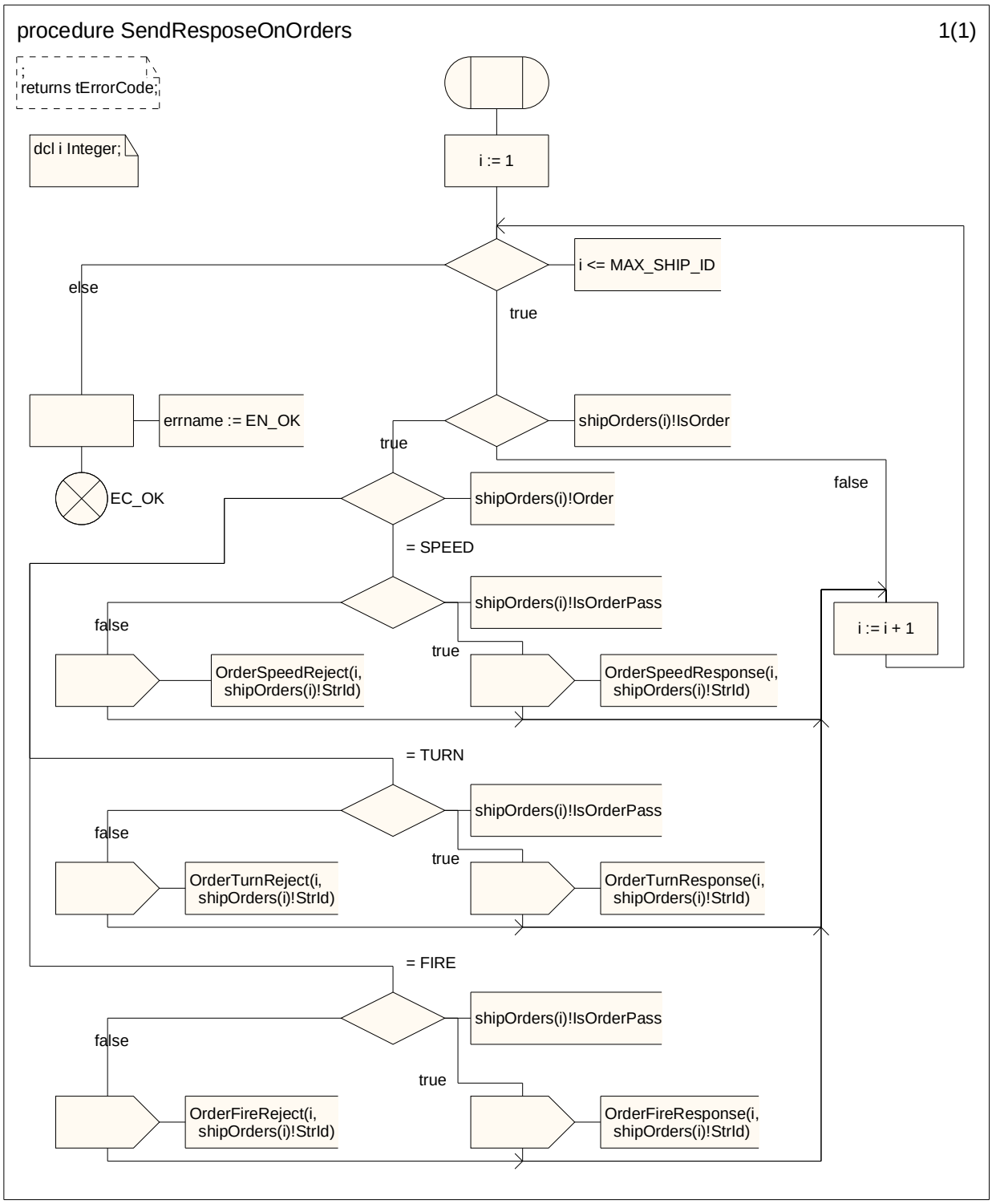

Рисунок 69. Процедура SendResposeOnOrders.

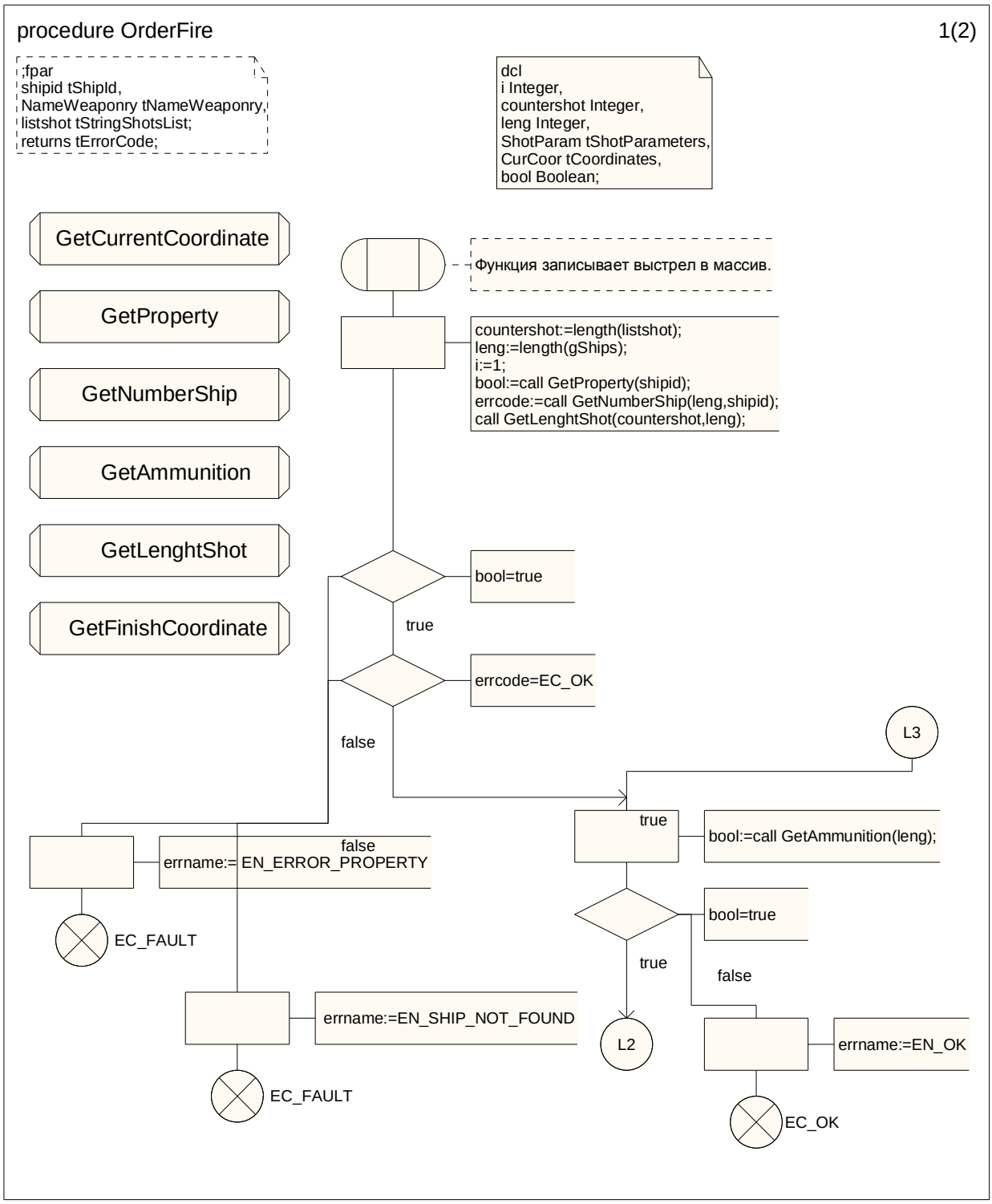

Рисунок 70. Процедура OrderFire (1).

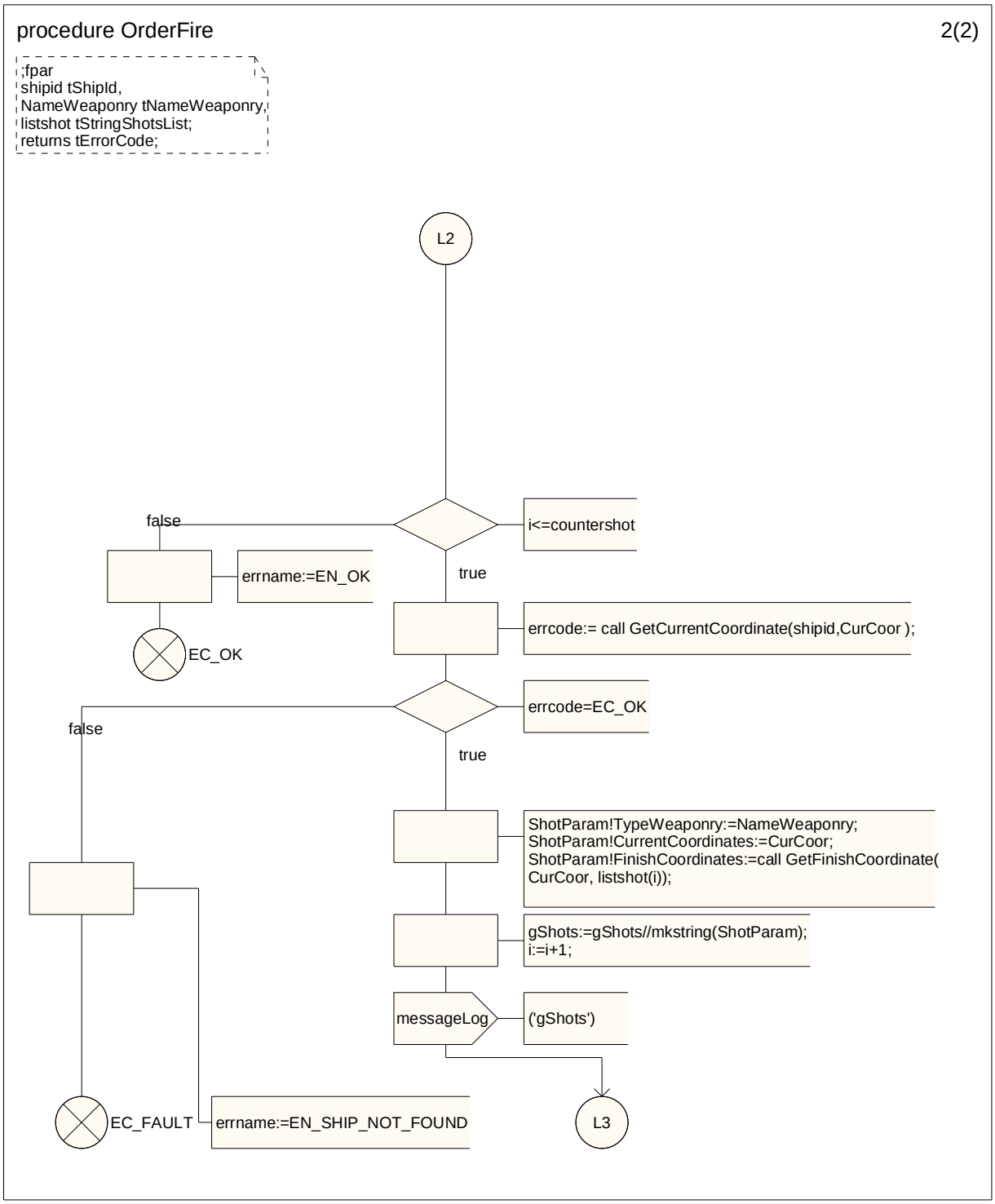

Рисунок 71. Процедура OrderFire (2).

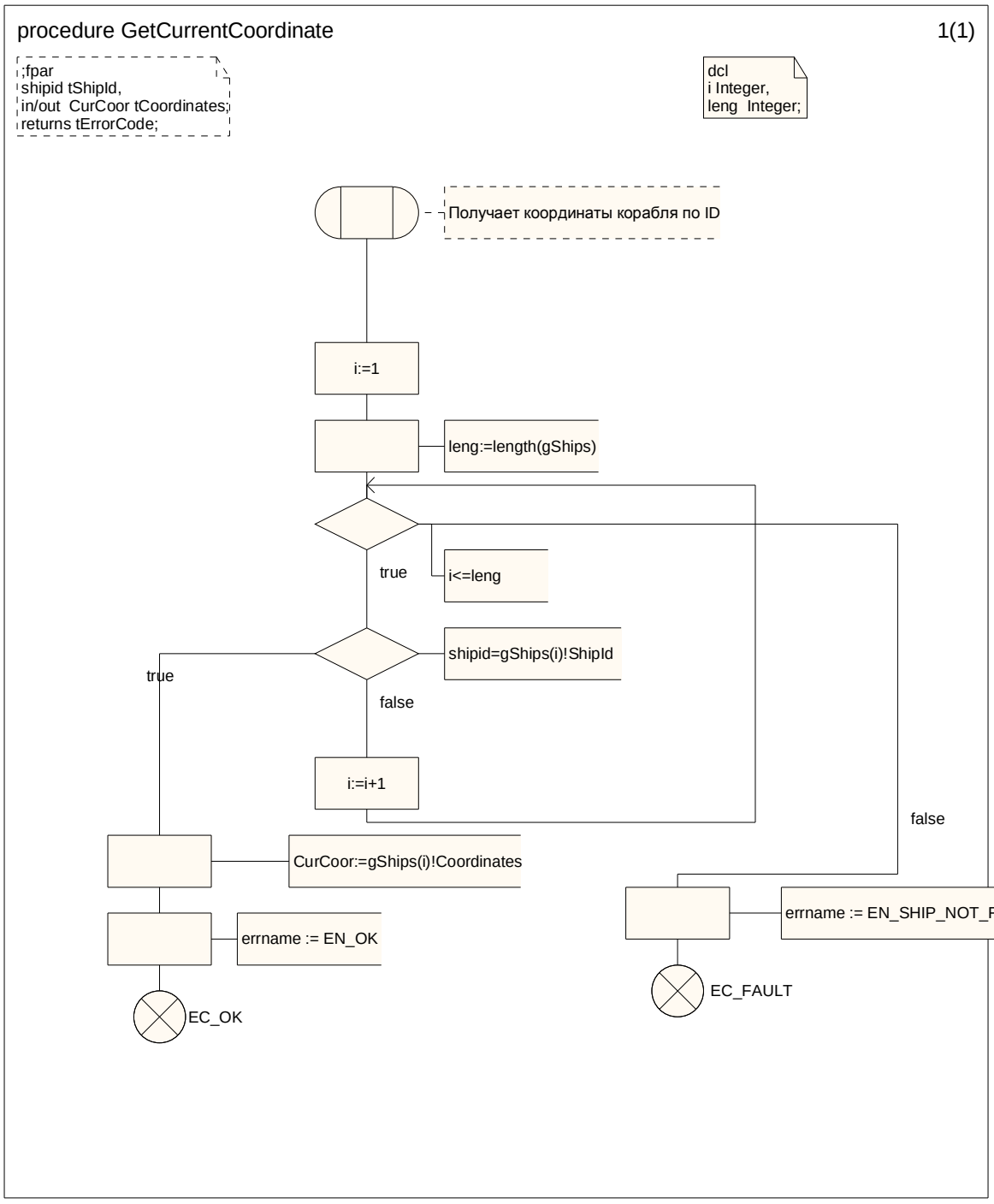

Рисунок 72. Процедура GetCurrentCoordinate.

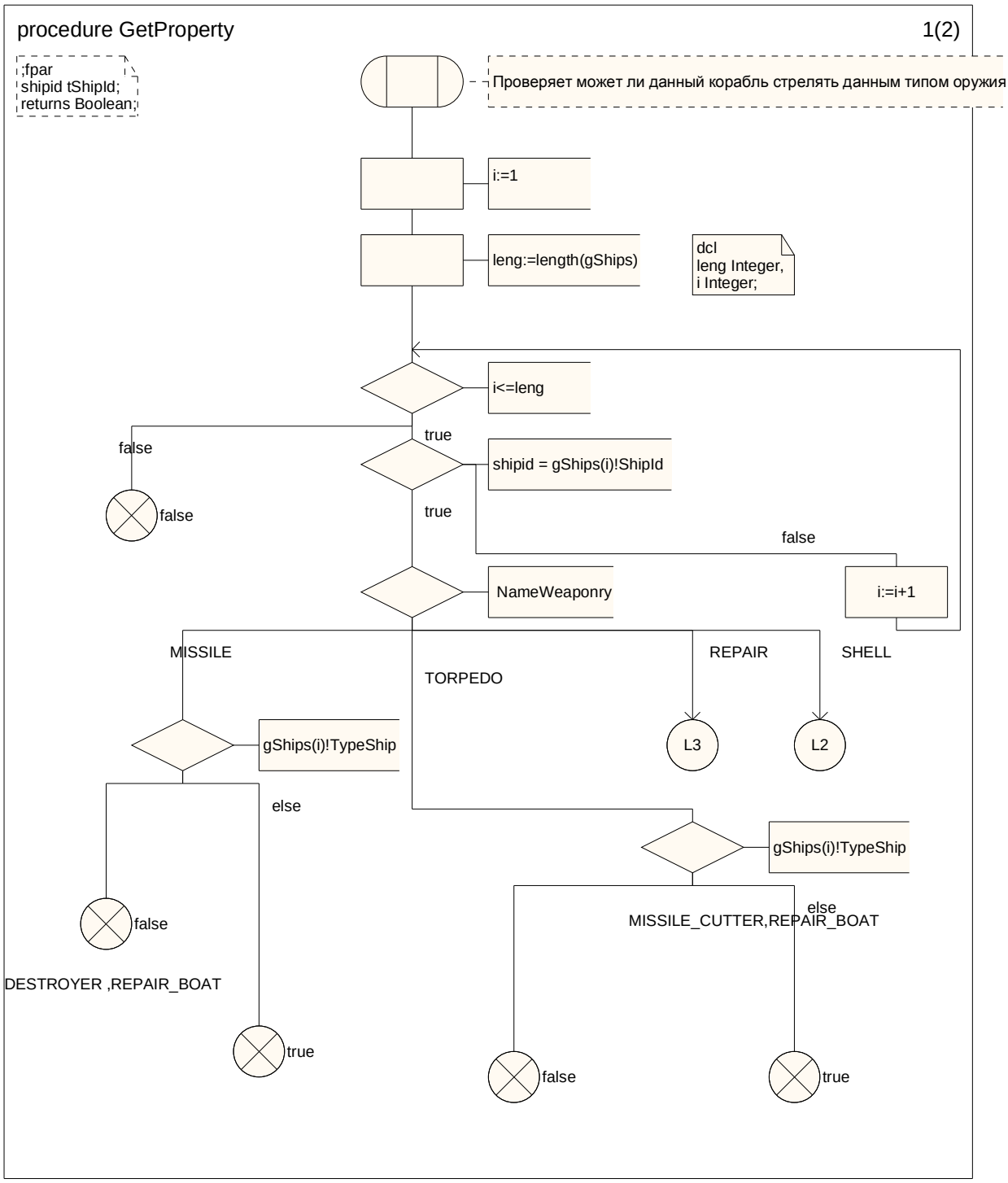

Рисунок 73. Процедура GetProperty (1).

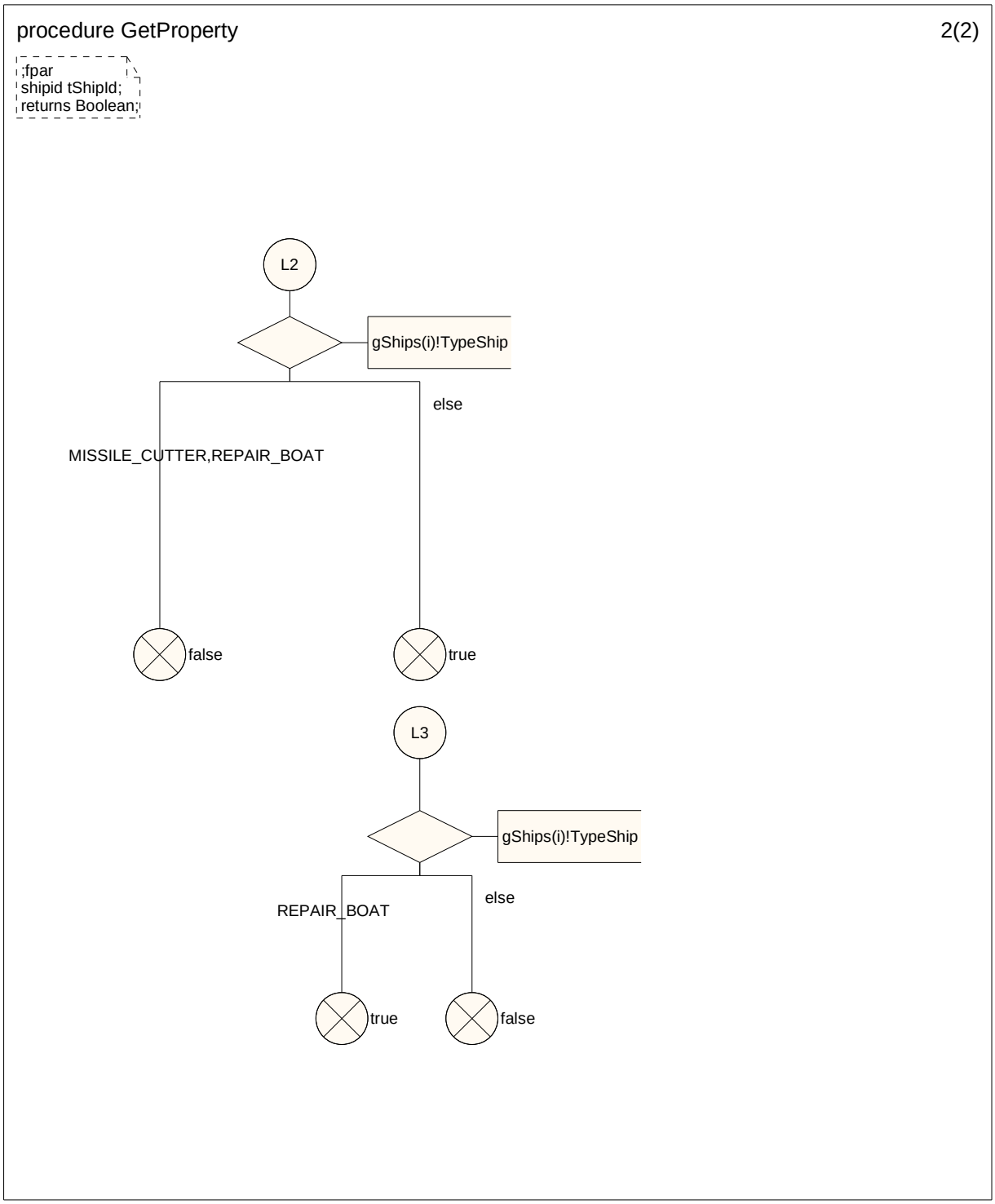

Рисунок 74. Процедура GetProperty (2).

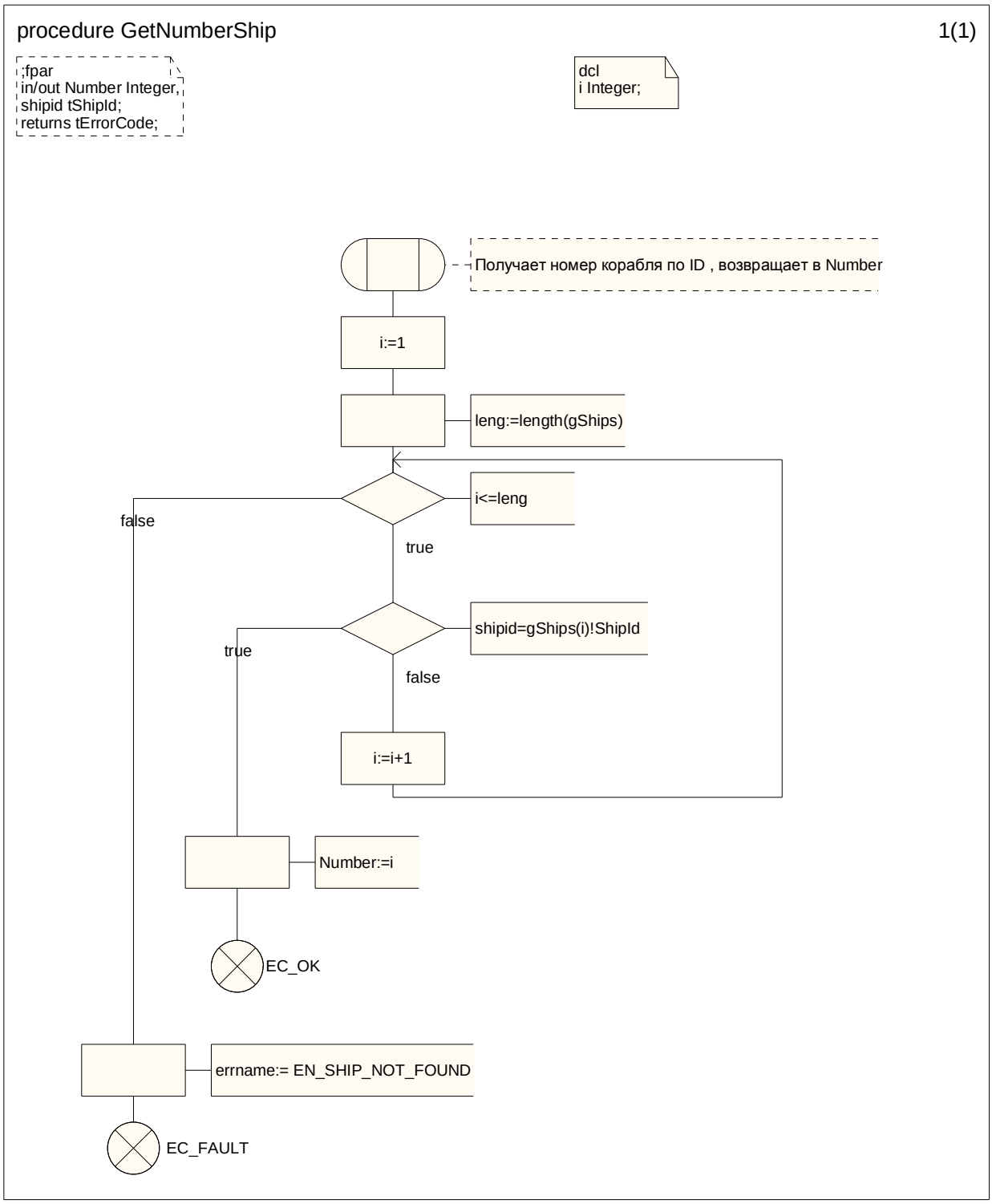

Рисунок 75. Процедура GetNumberShip.

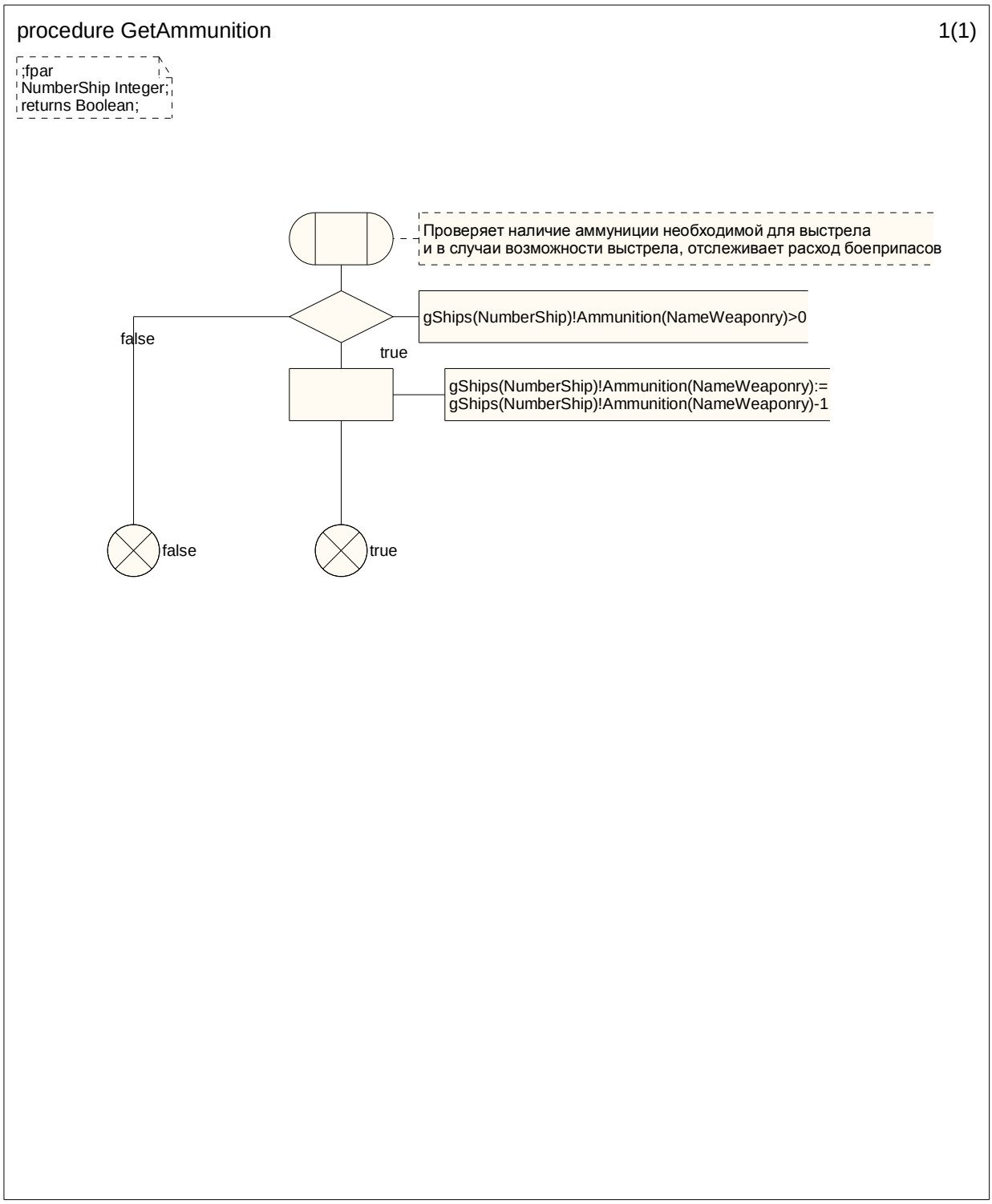

Рисунок 76. Процедура GetAmmunition.

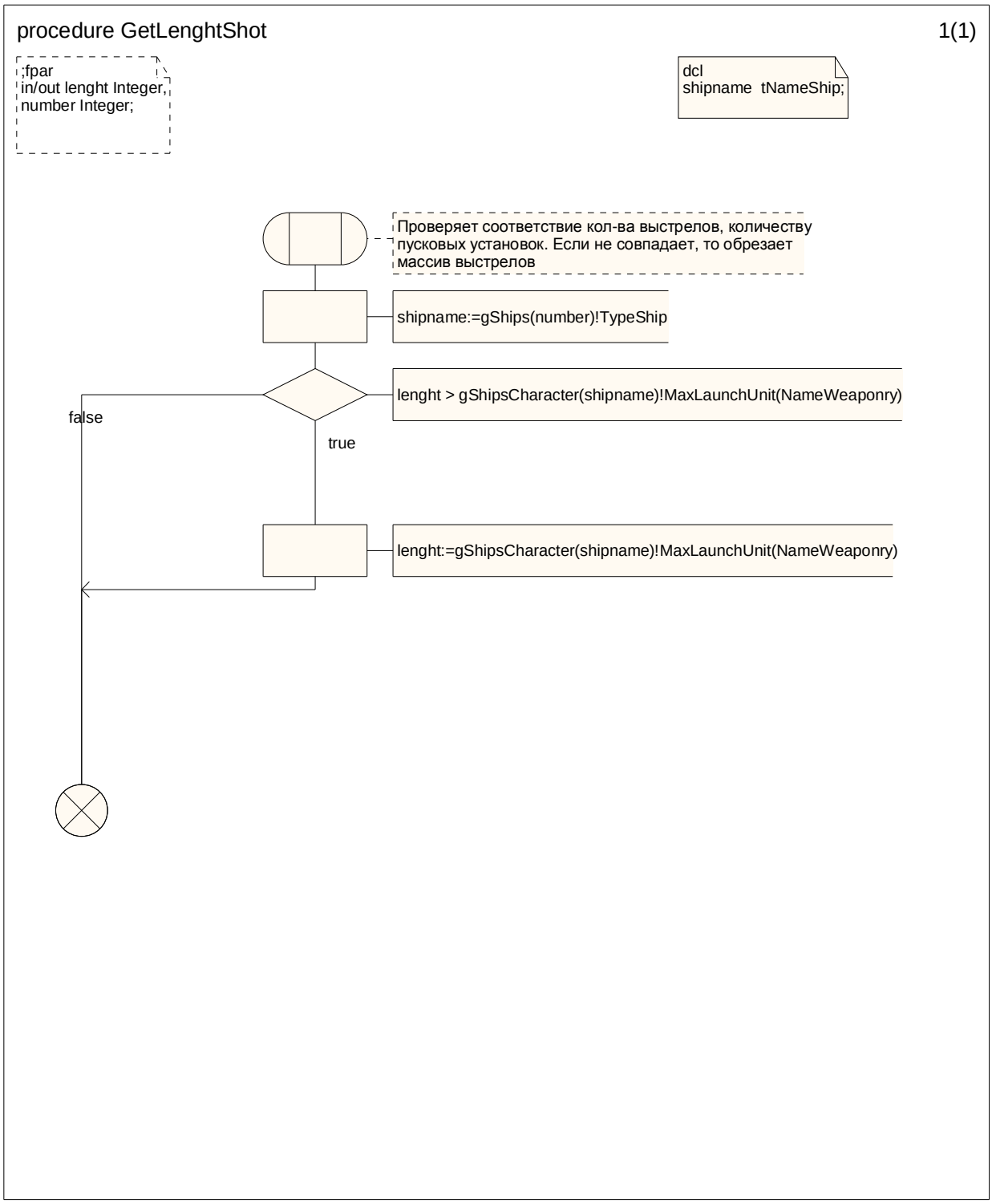

Рисунок 77. Процедура GetLenghtShot.

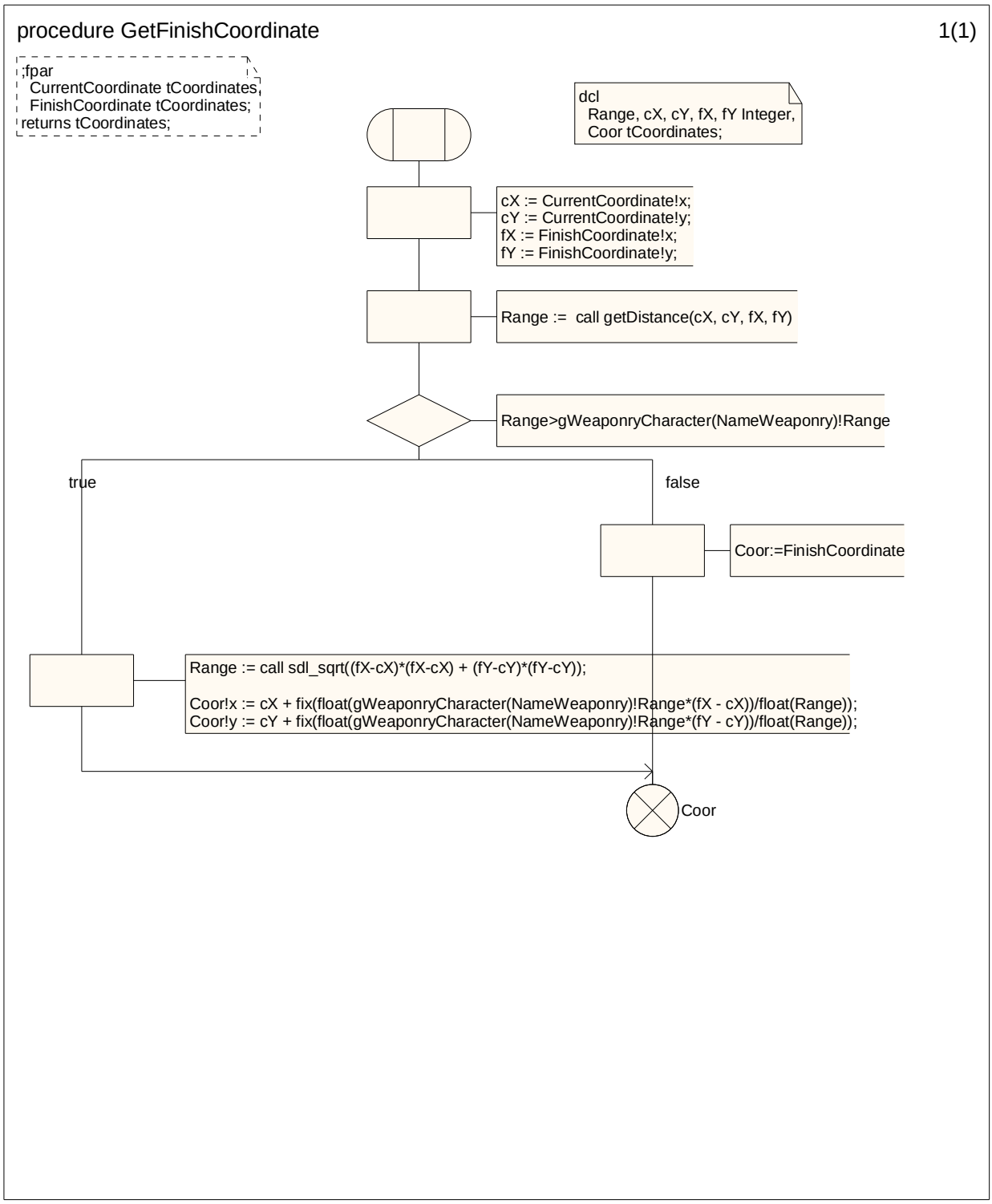

Рисунок 78. Процедура GetFinishCoordinate.

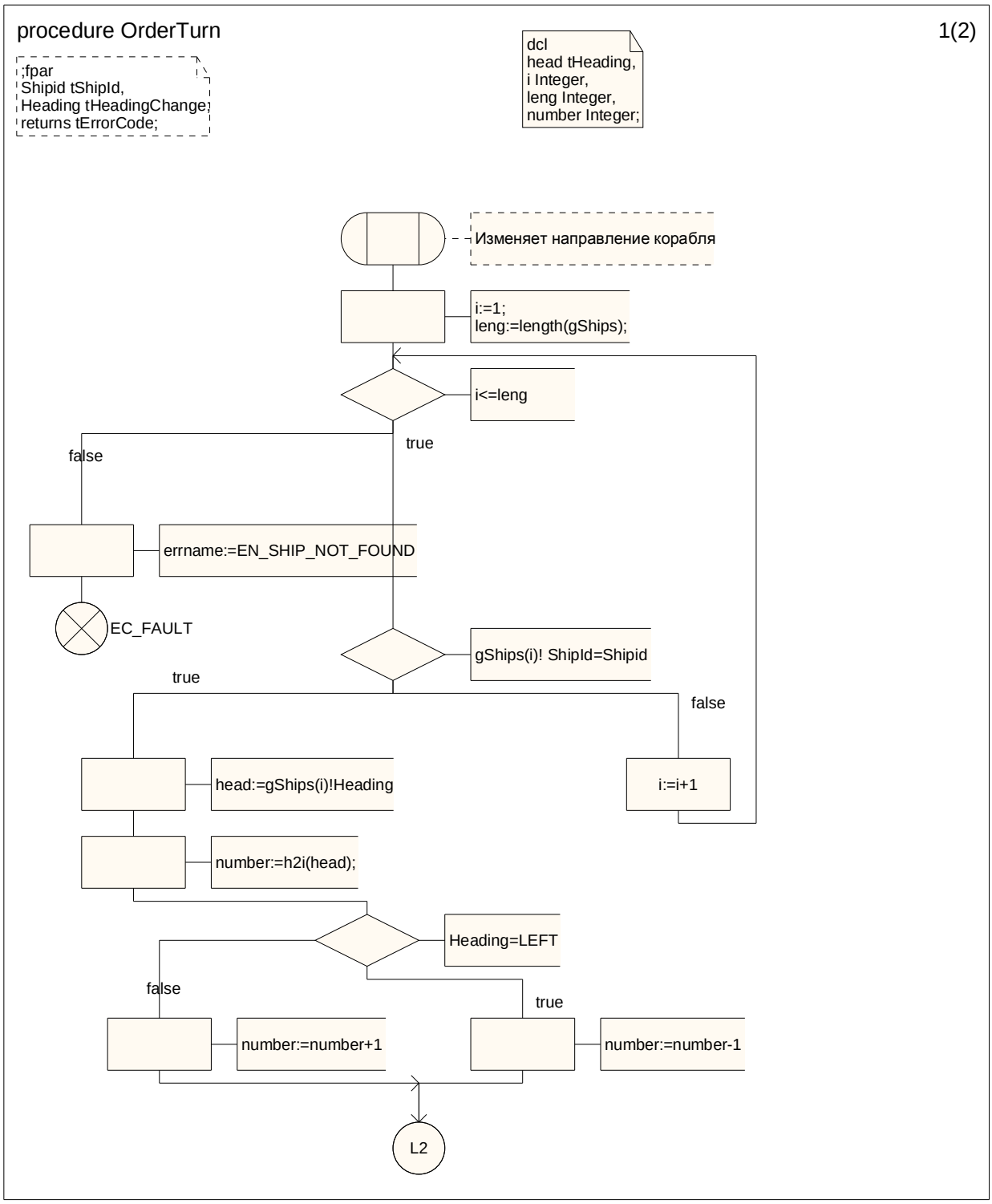

Рисунок 79. Процедура OrderTurn (1).

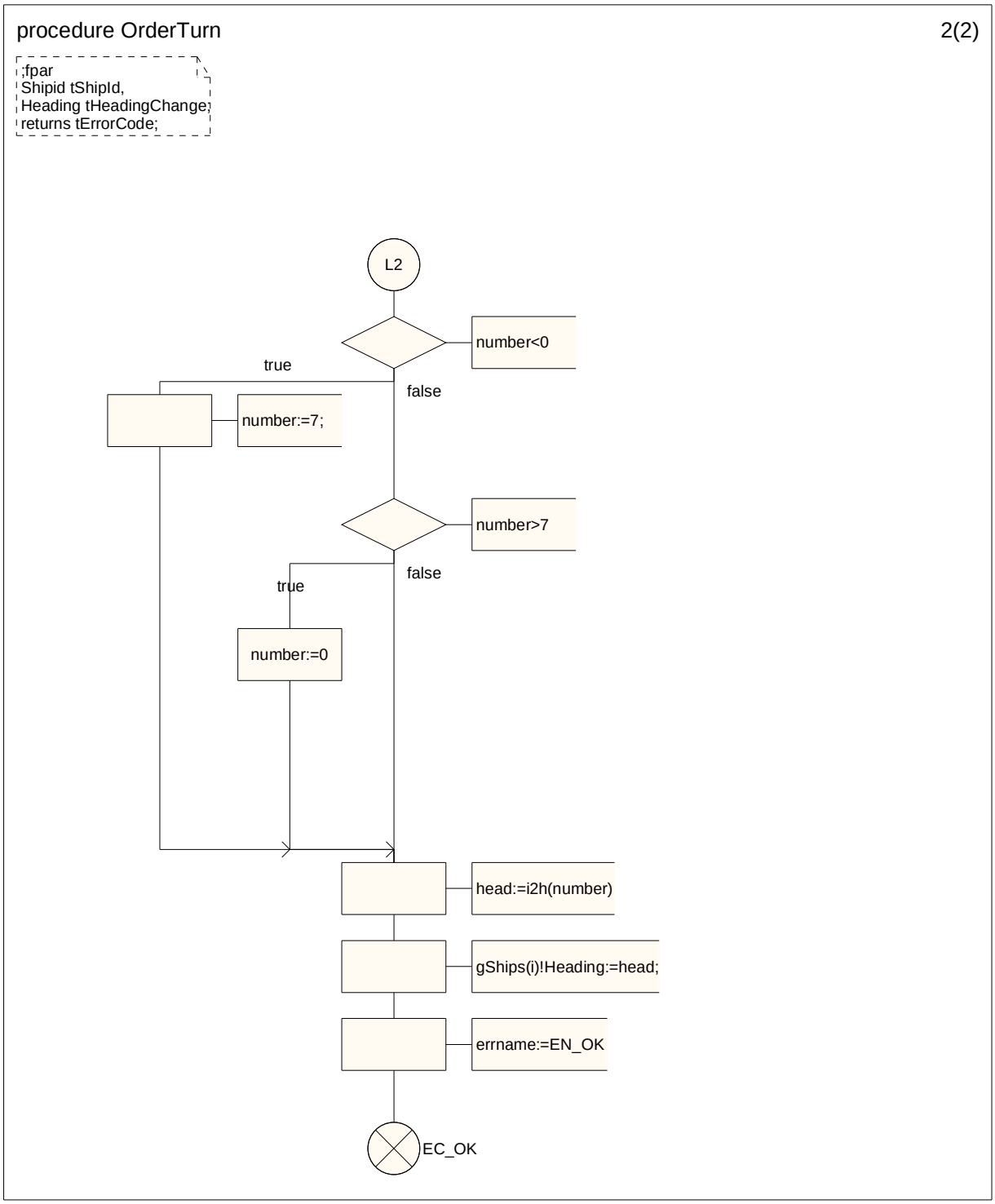

Рисунок 80. Процедура OrderTurn (2).

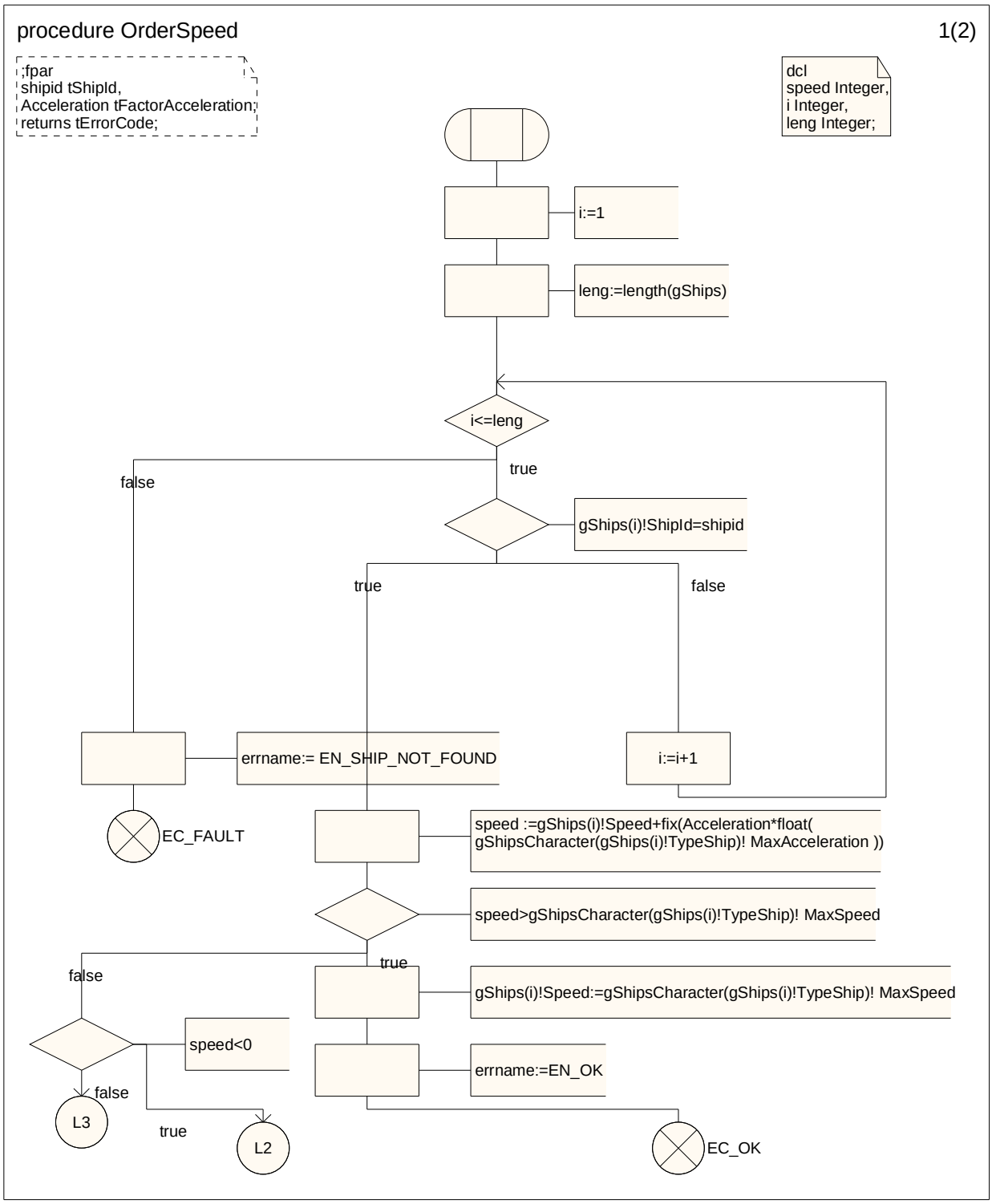

Рисунок 81. Процедура OrderSpeed (1).

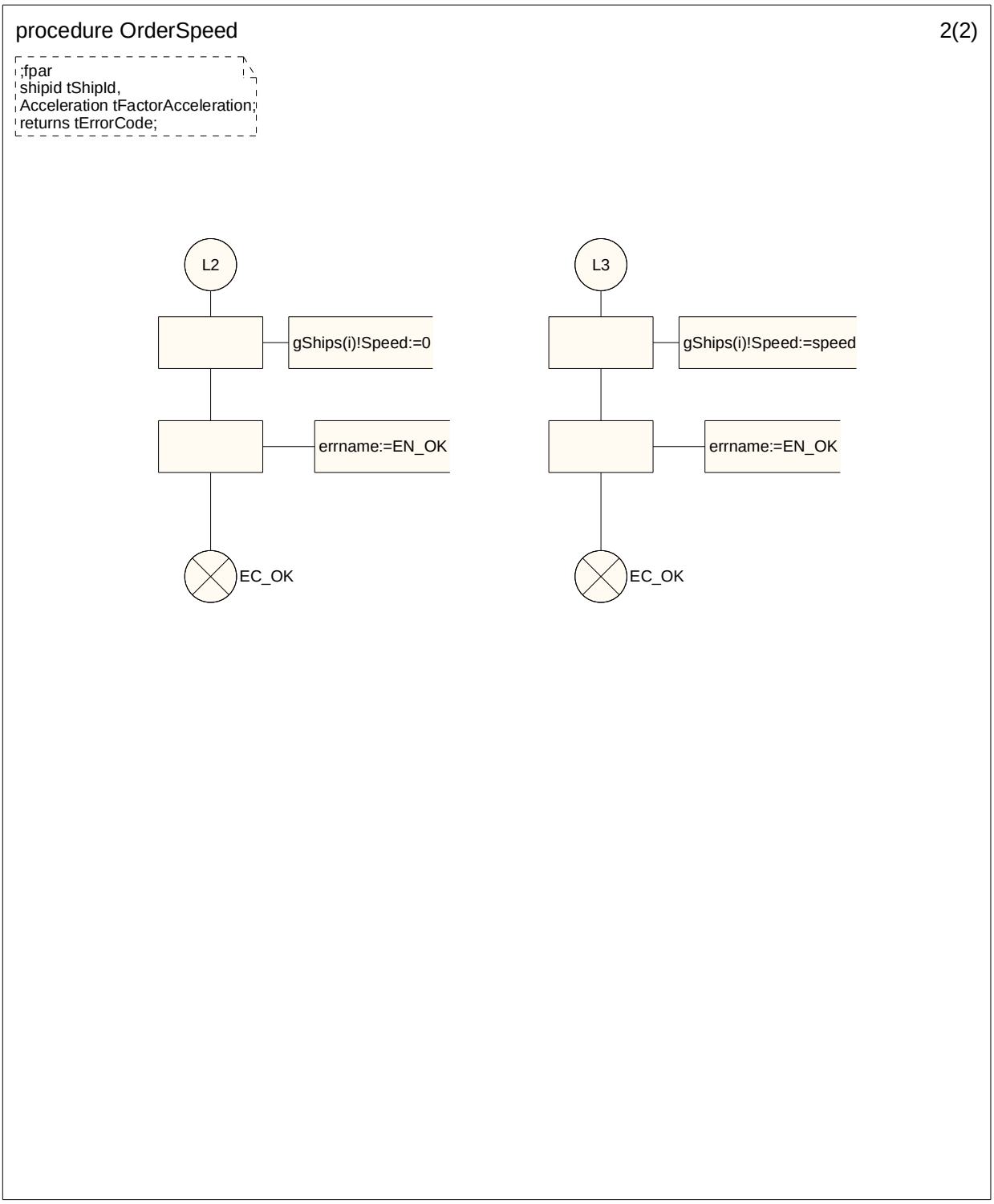

Рисунок 82. Процедура OrderSpeed (2).

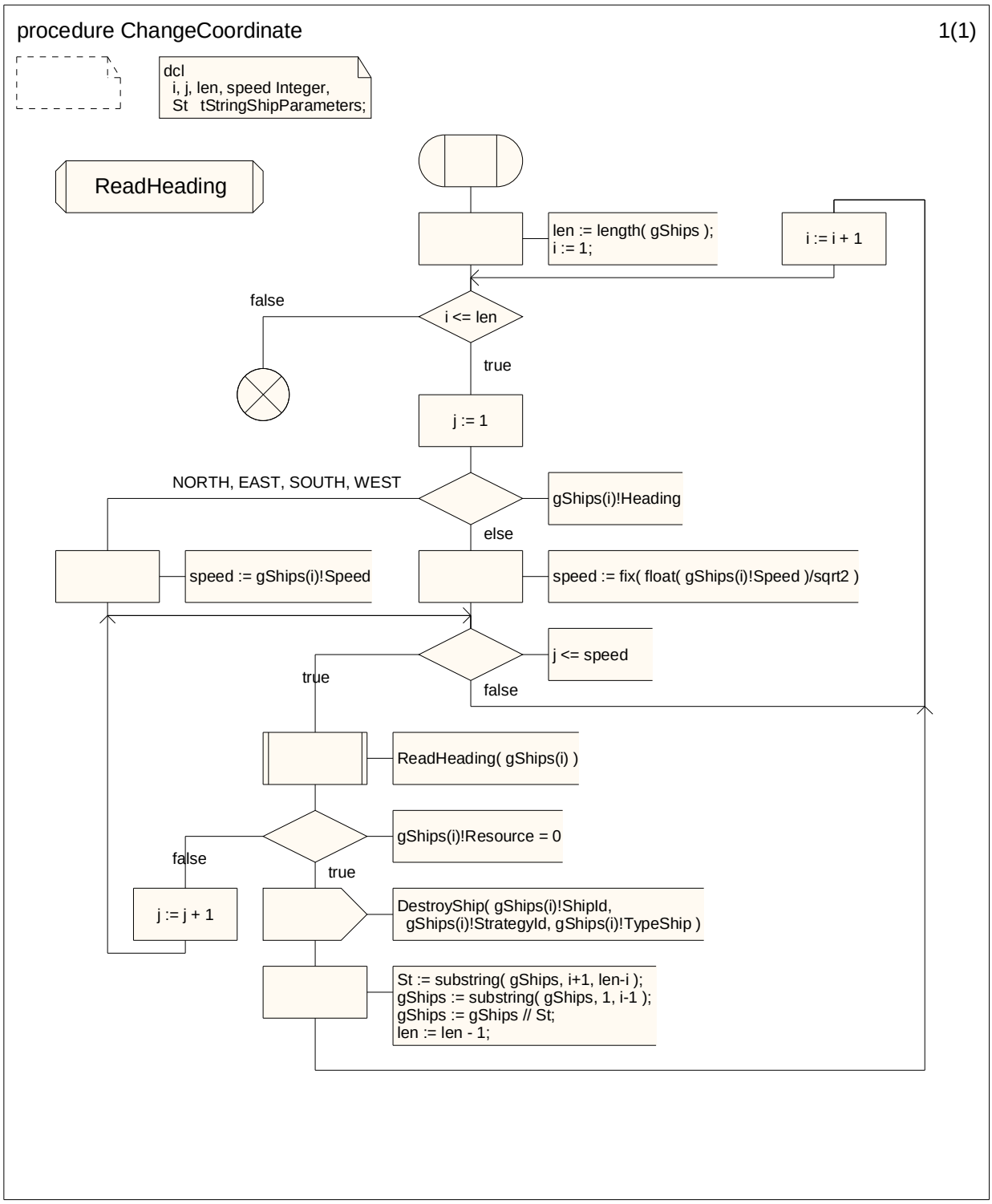

Рисунок 83. Процедура ChangeCoordinate.

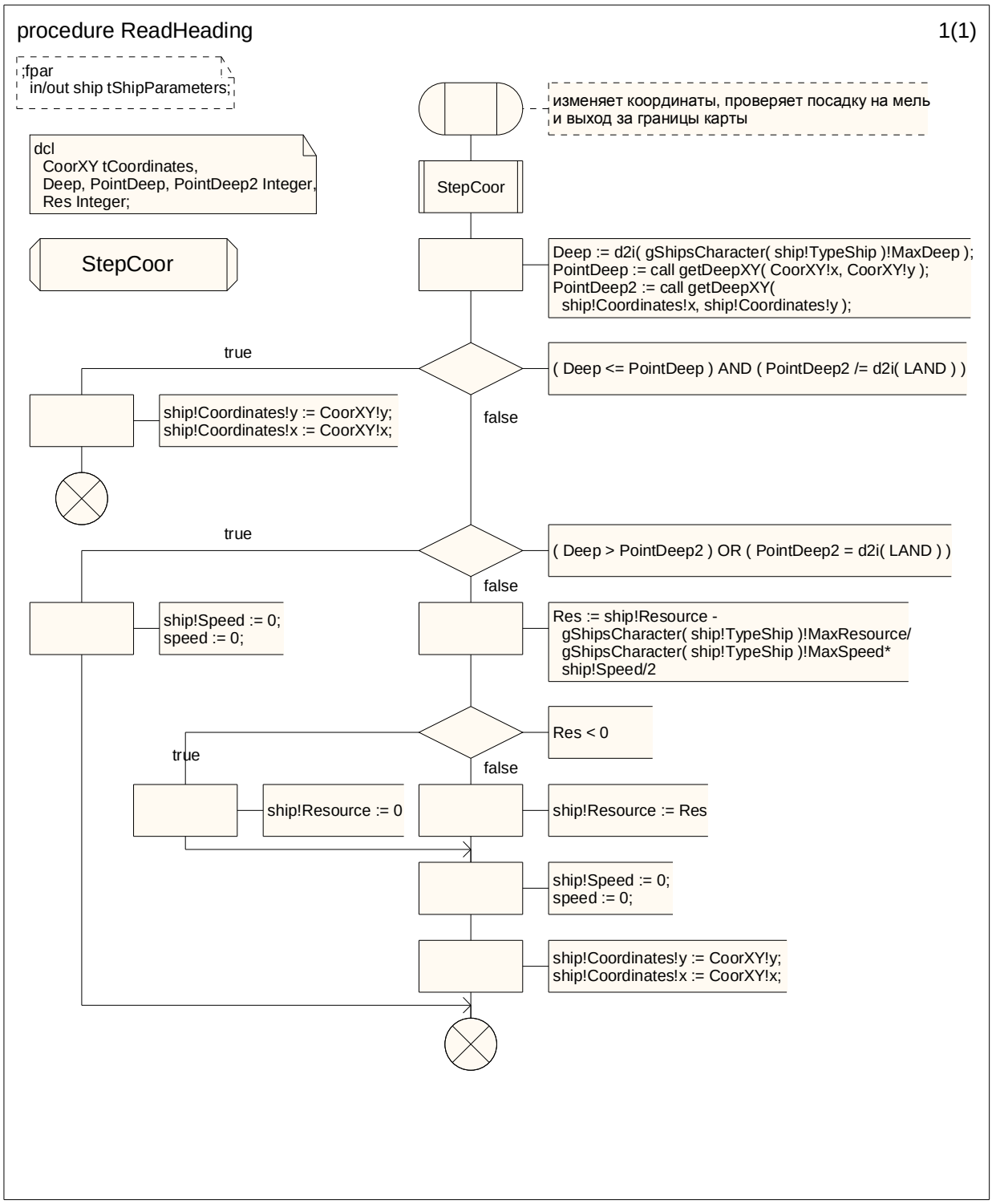

Рисунок 84. Процедура ReadHeading.

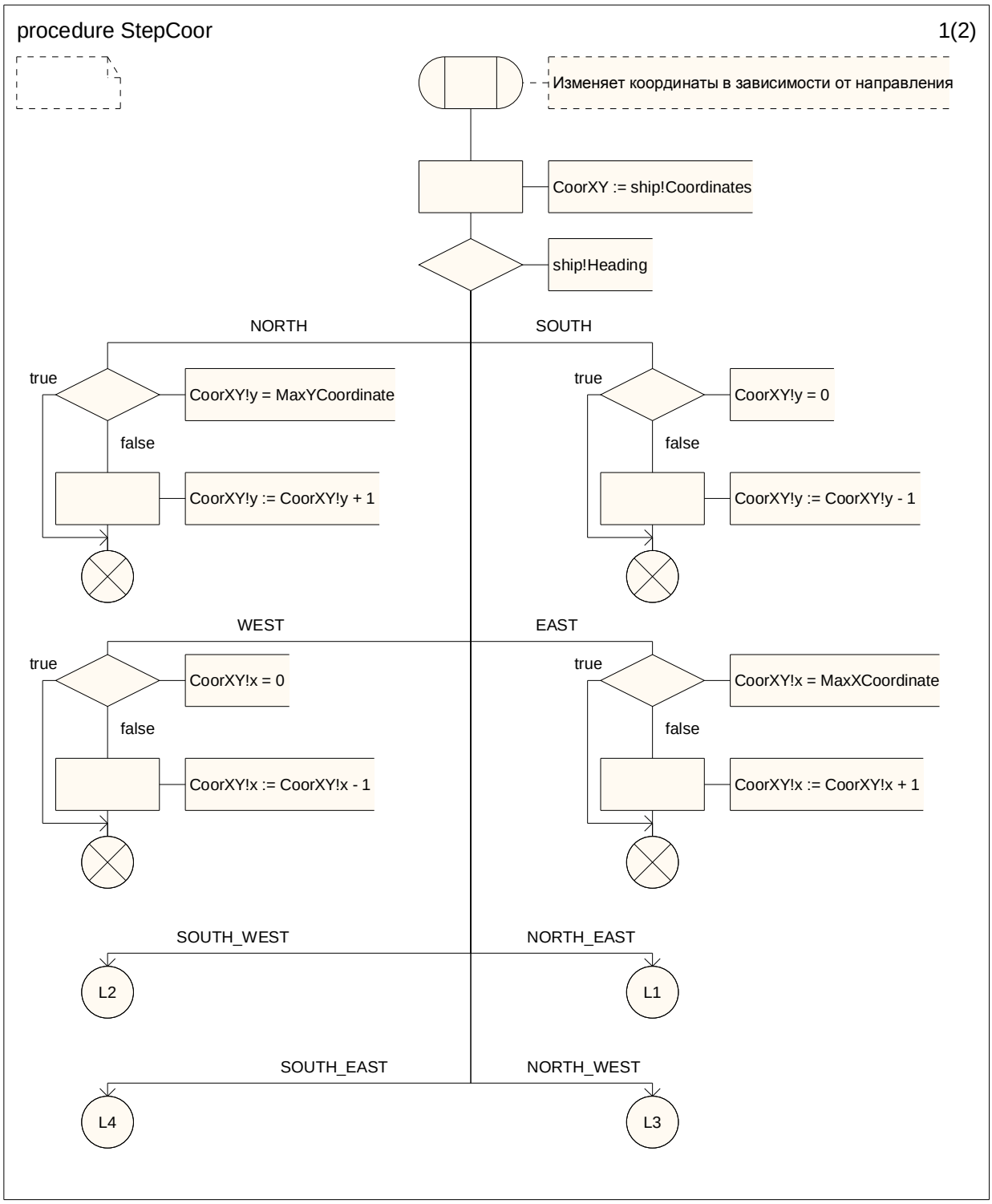

Рисунок 85. Процедура StepCoor (1).

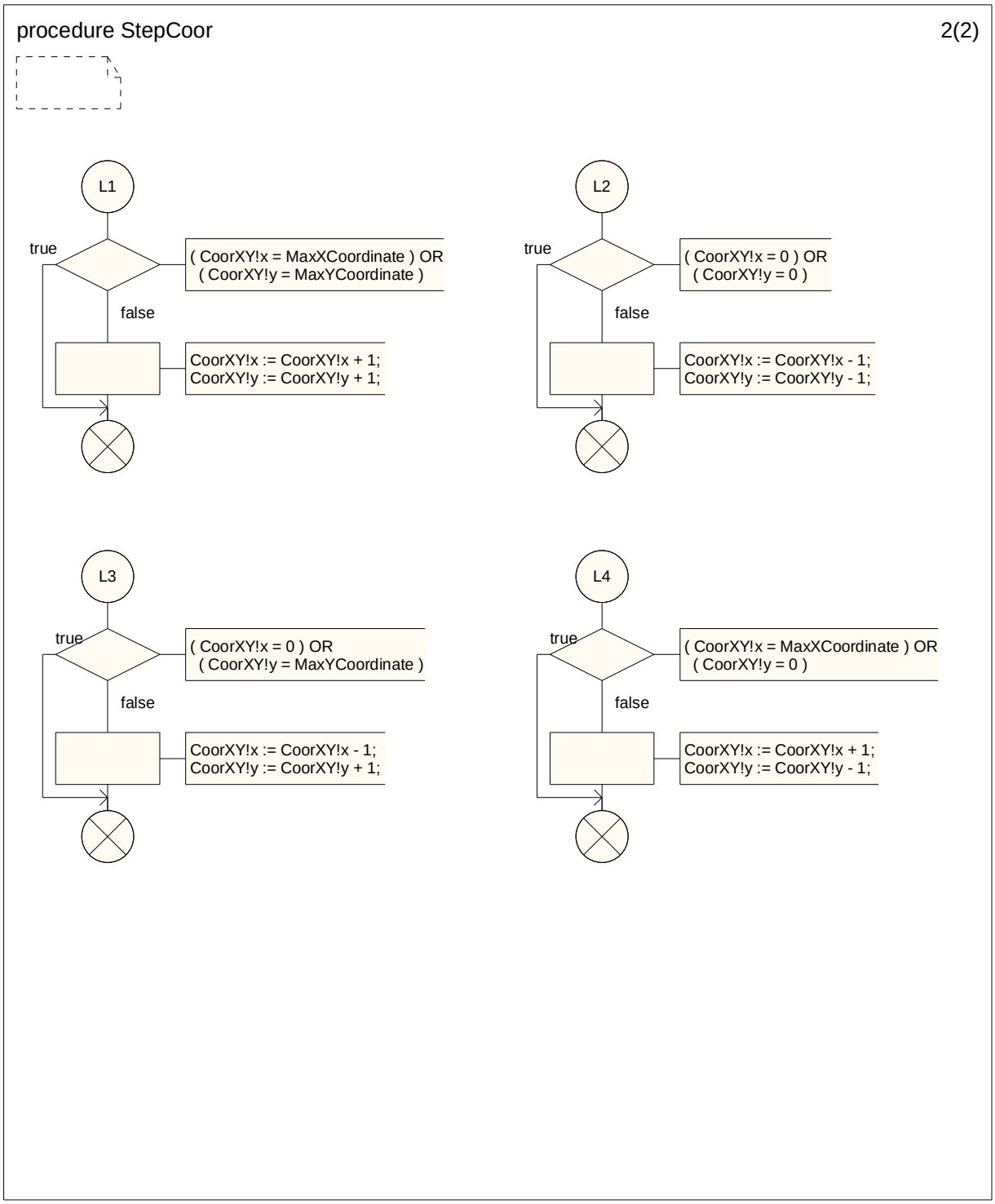

Рисунок 86. Процедура StepCoor (2).

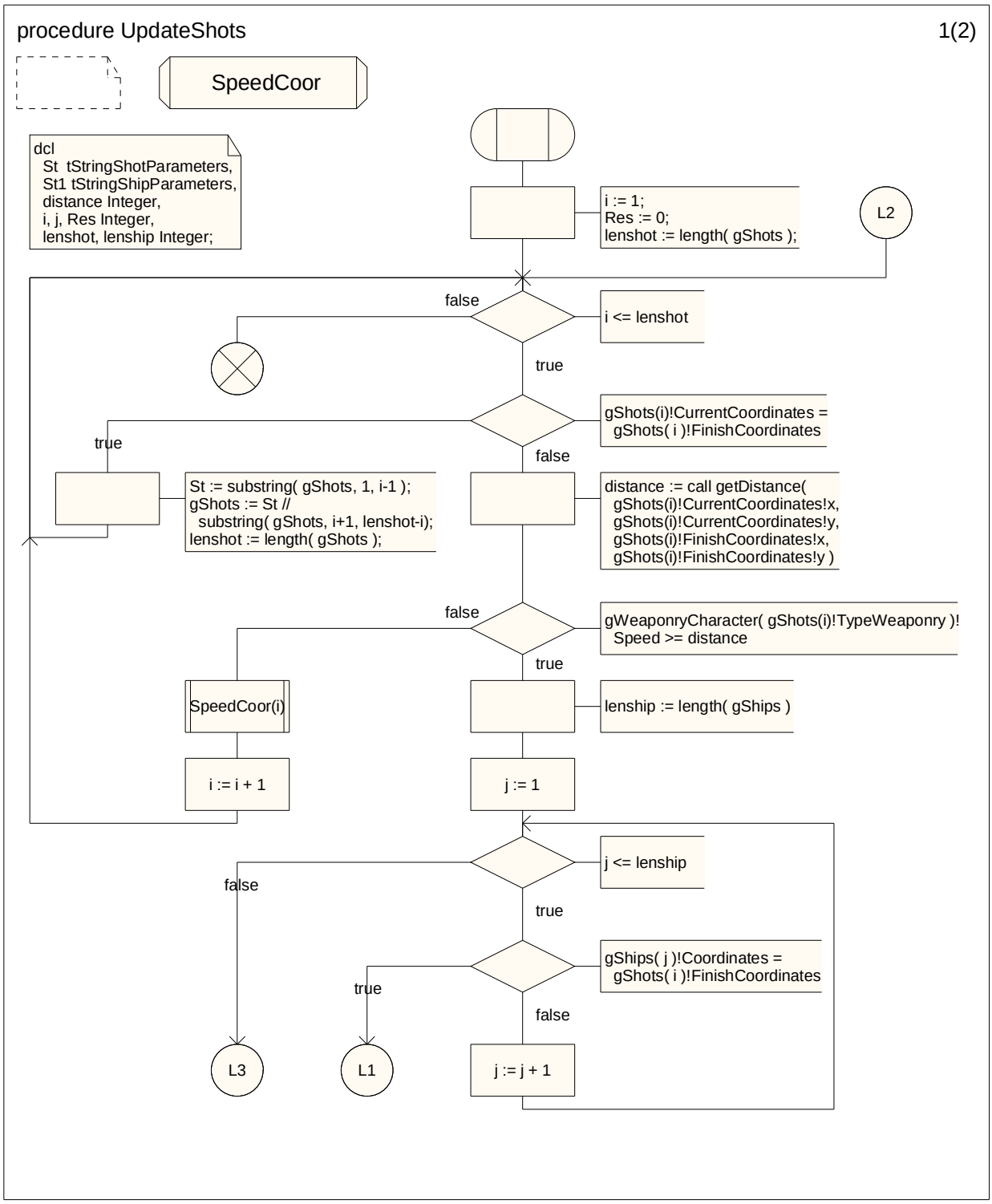

Рисунок 87. Процедура UpdateShots (1).

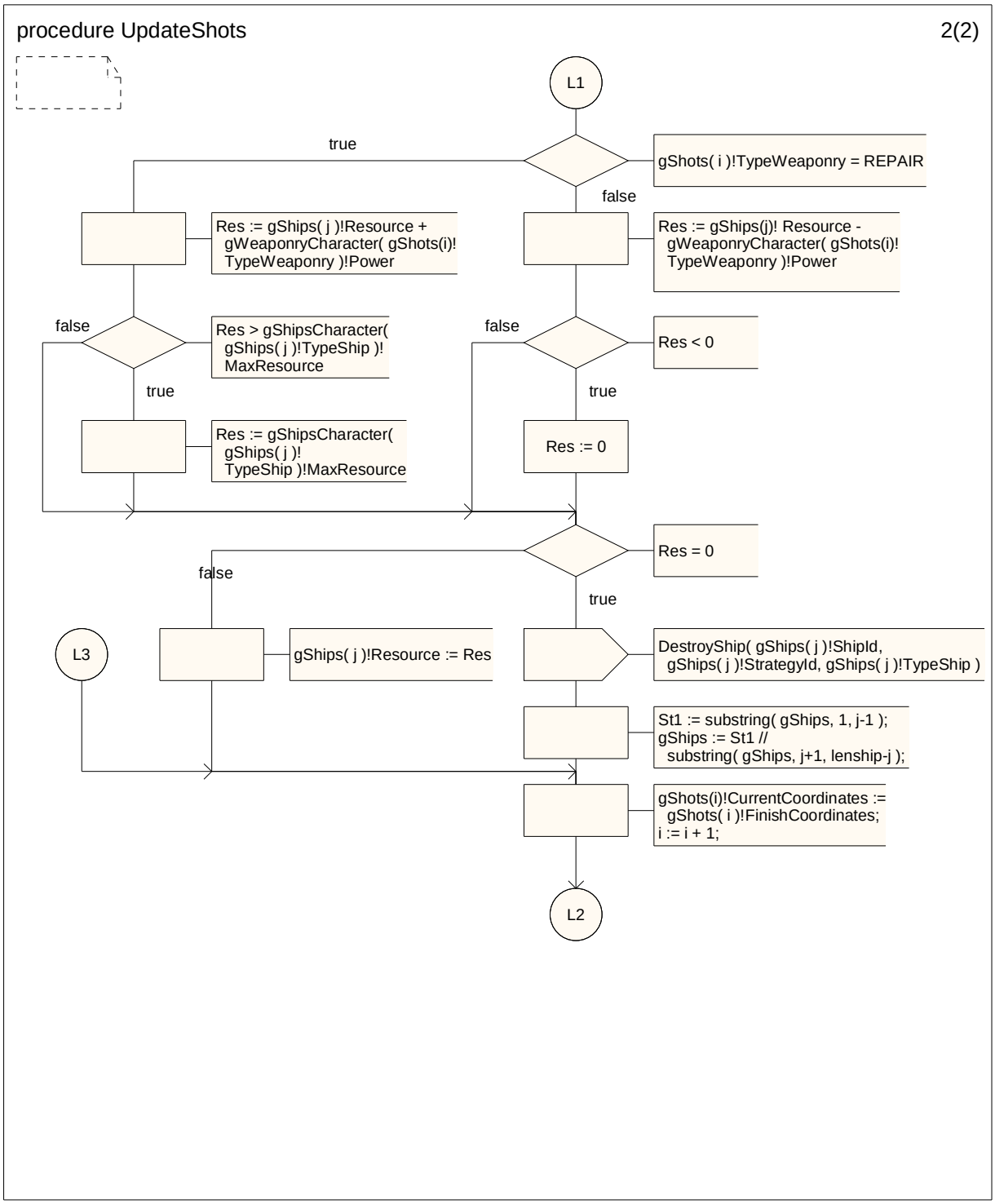

Рисунок 88. Процедура UpdateShots (2).

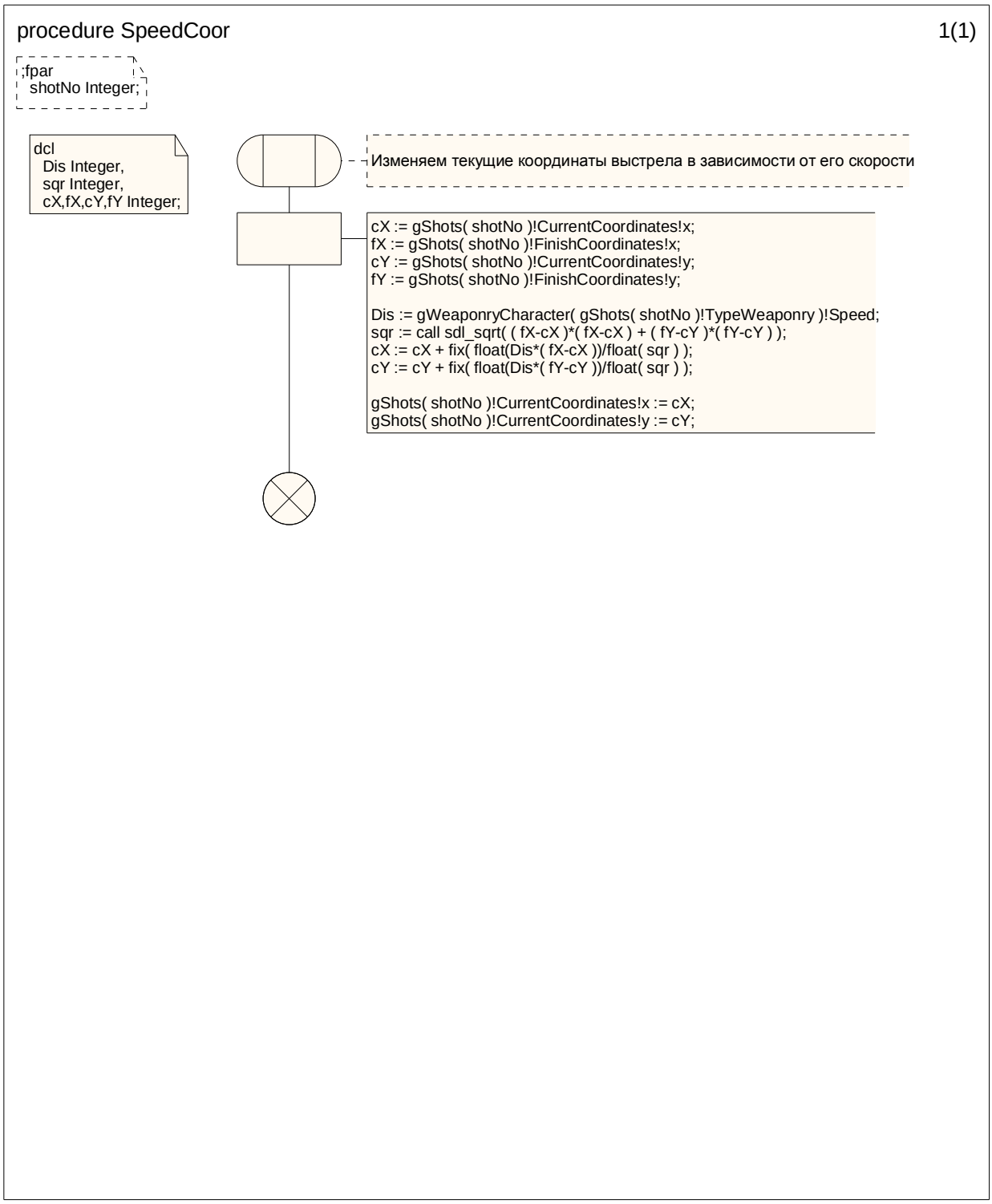

Рисунок 89. Процедура SpeedCoor.

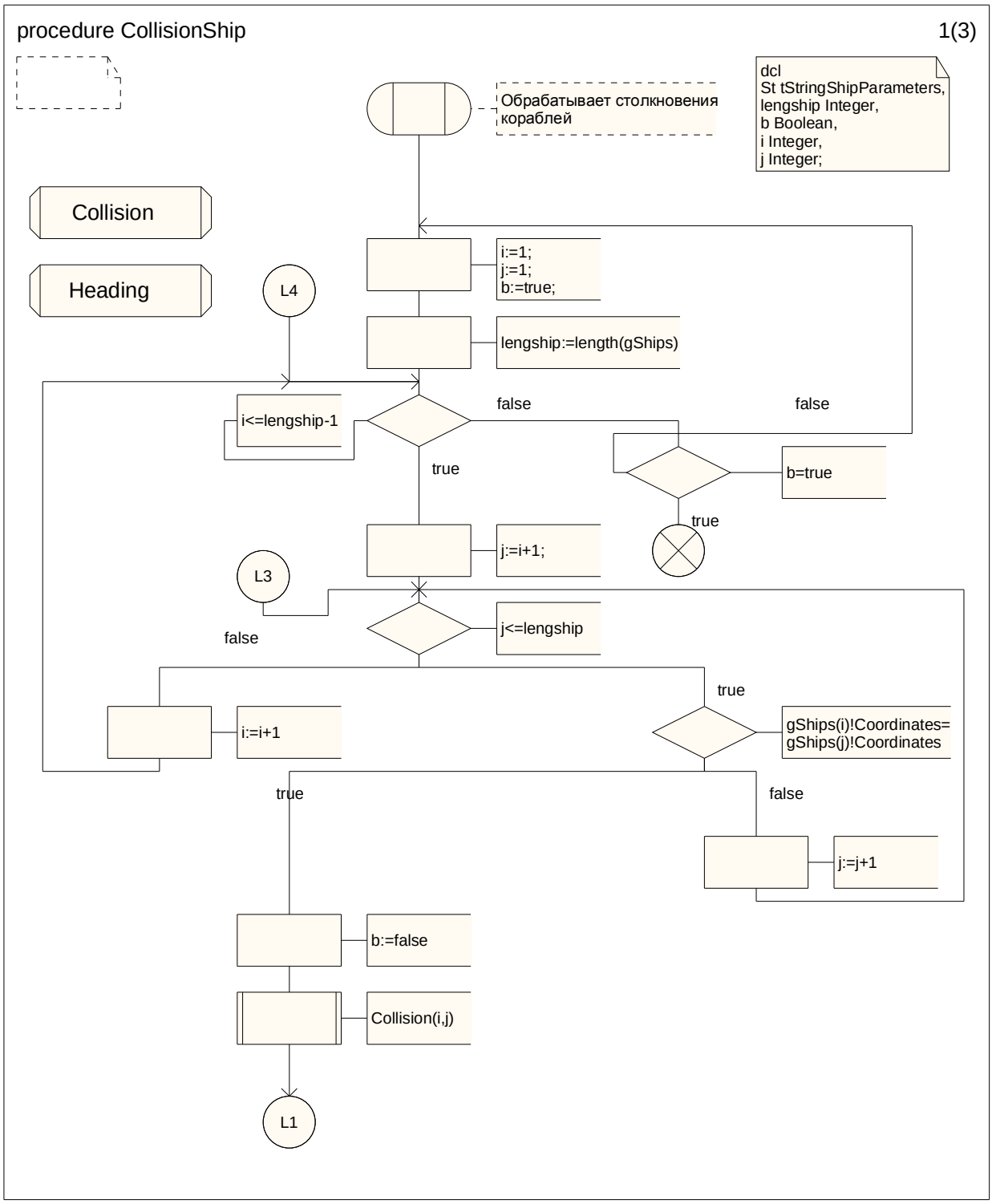

Рисунок 90. Процедура CollisionShip (1).

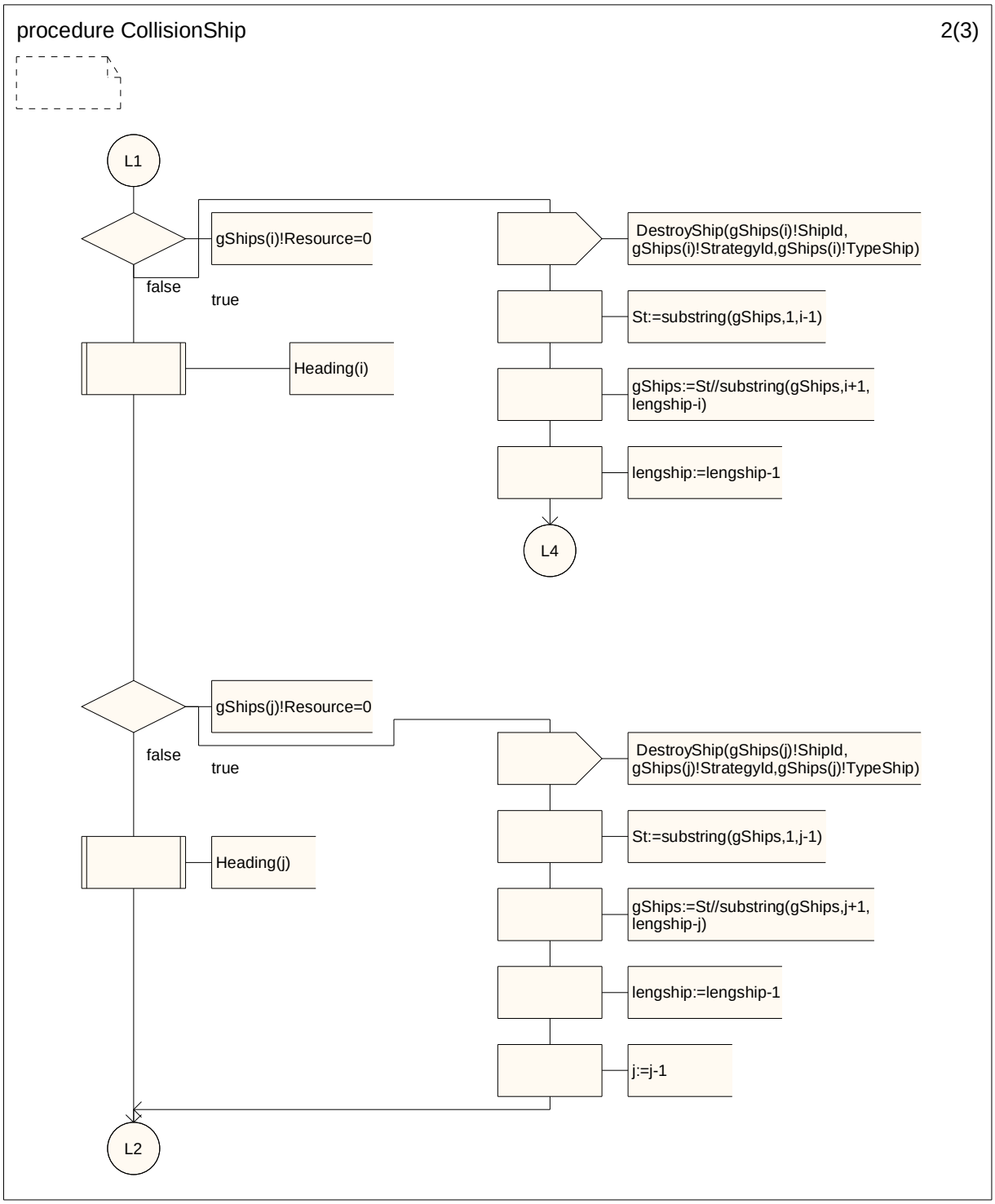

Рисунок 91. Процедура CollisionShip (2).

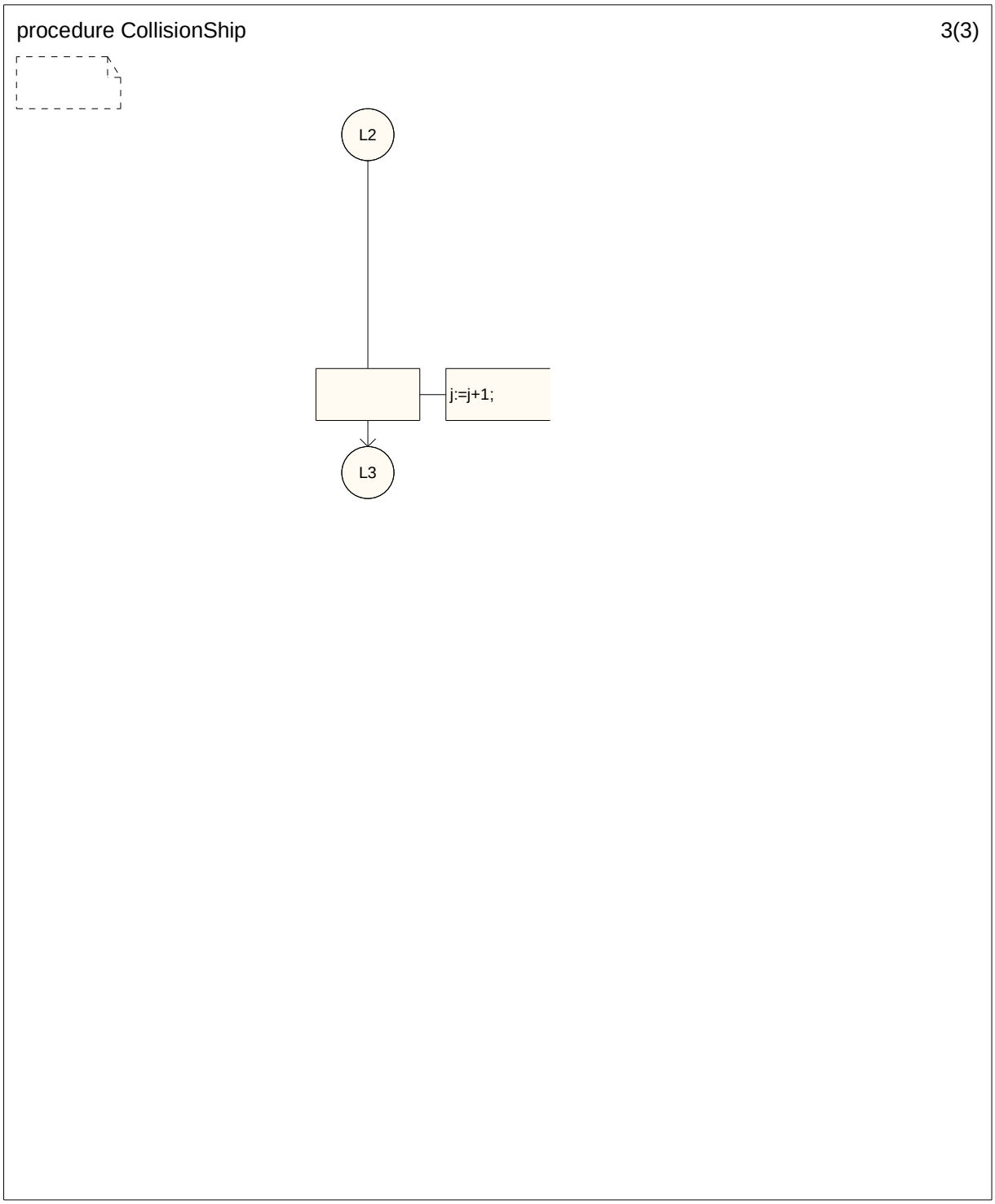

Рисунок 92. Процедура CollisionShip (3).

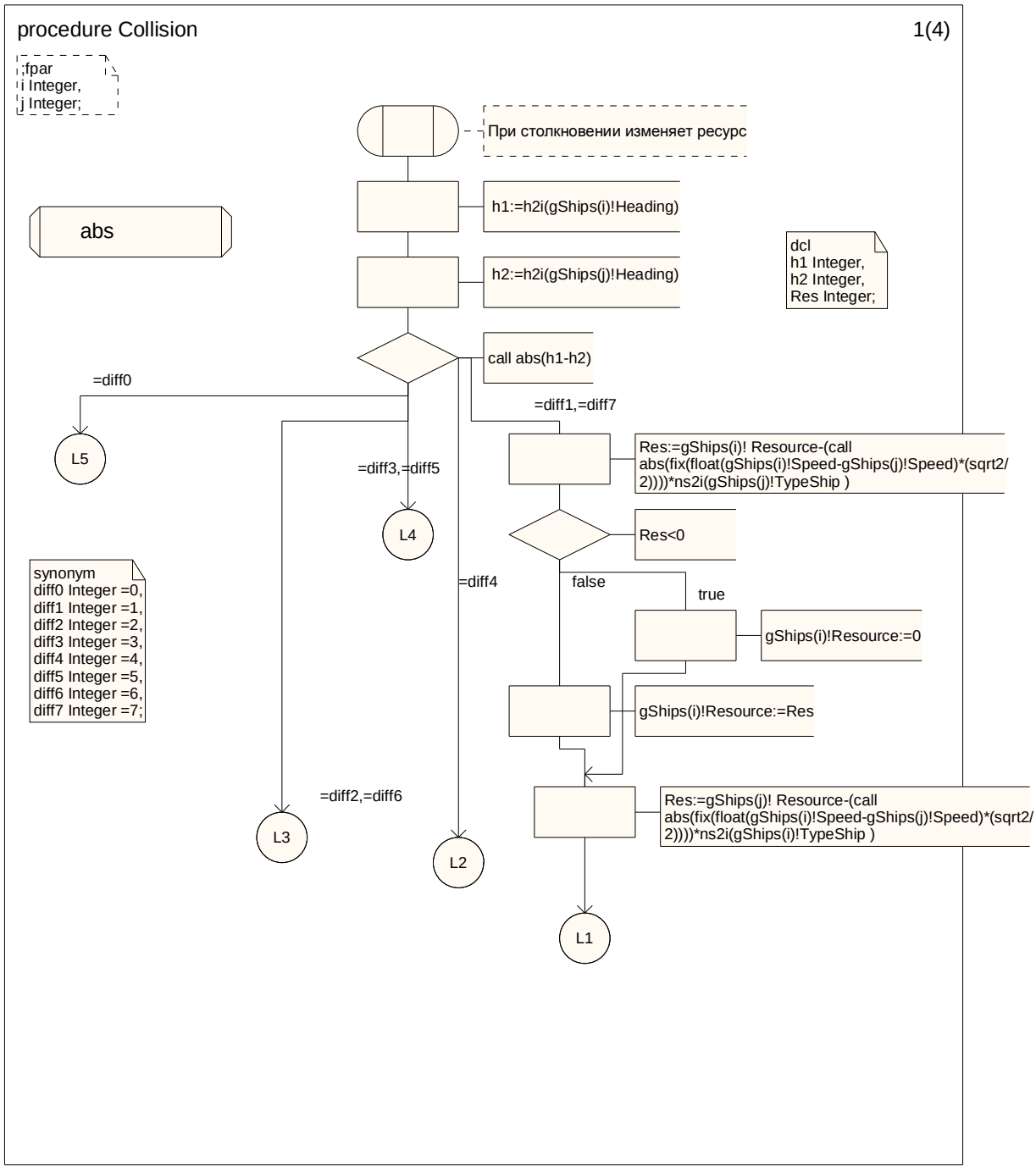

Рисунок 93. Процедура Collision (1).

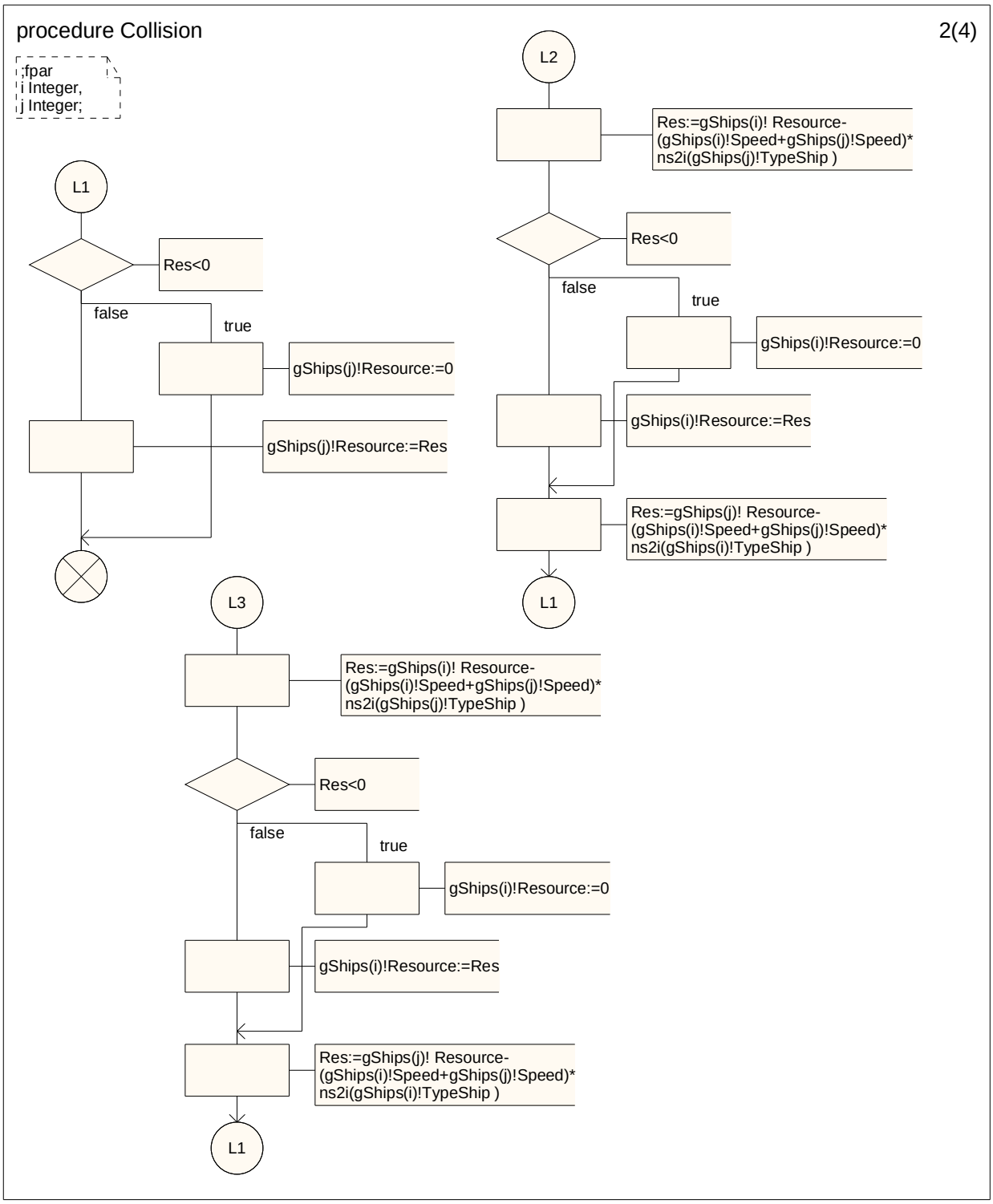

Рисунок 94. Процедура Collision (2).

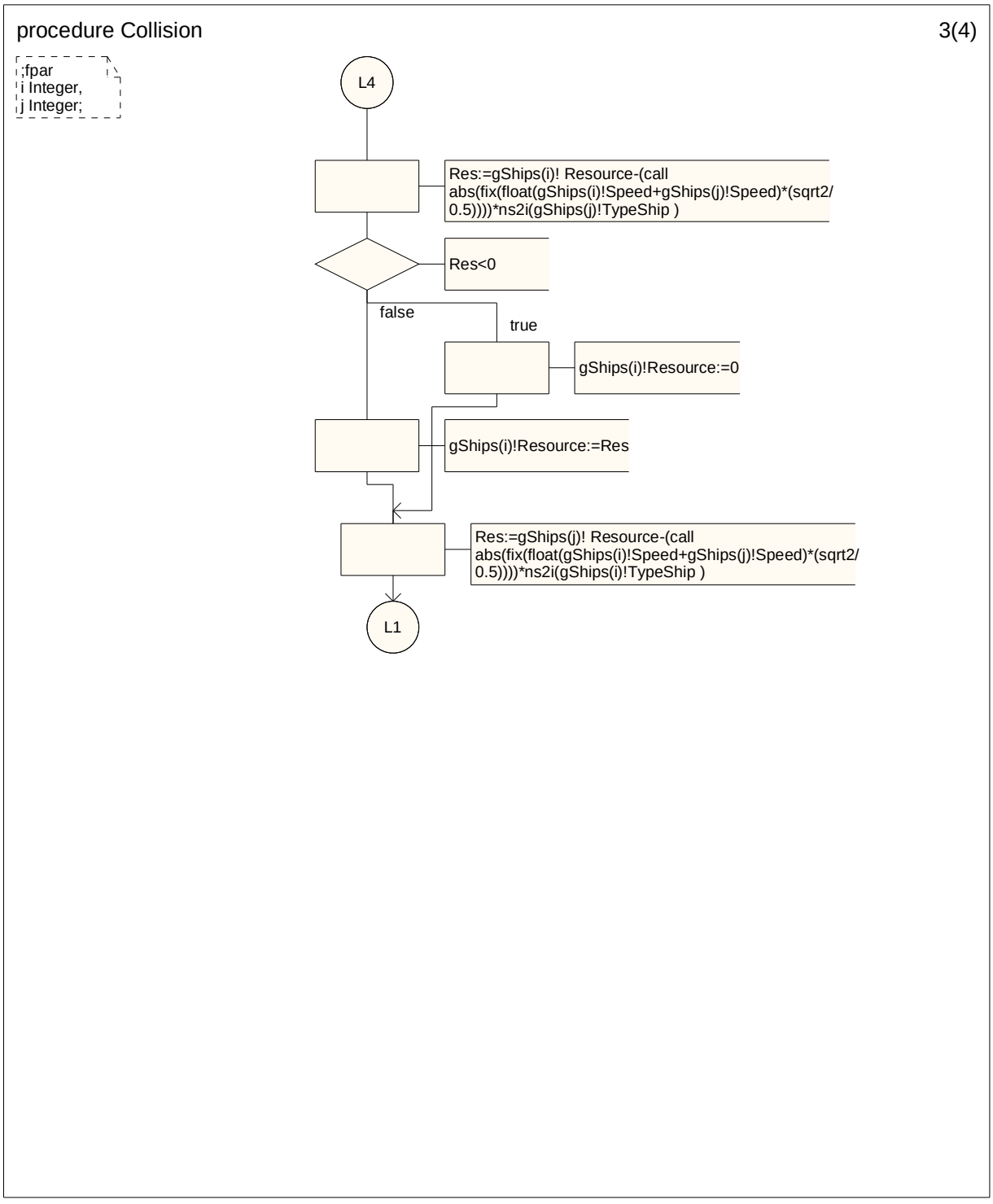

Рисунок 95. Процедура Collision (3).

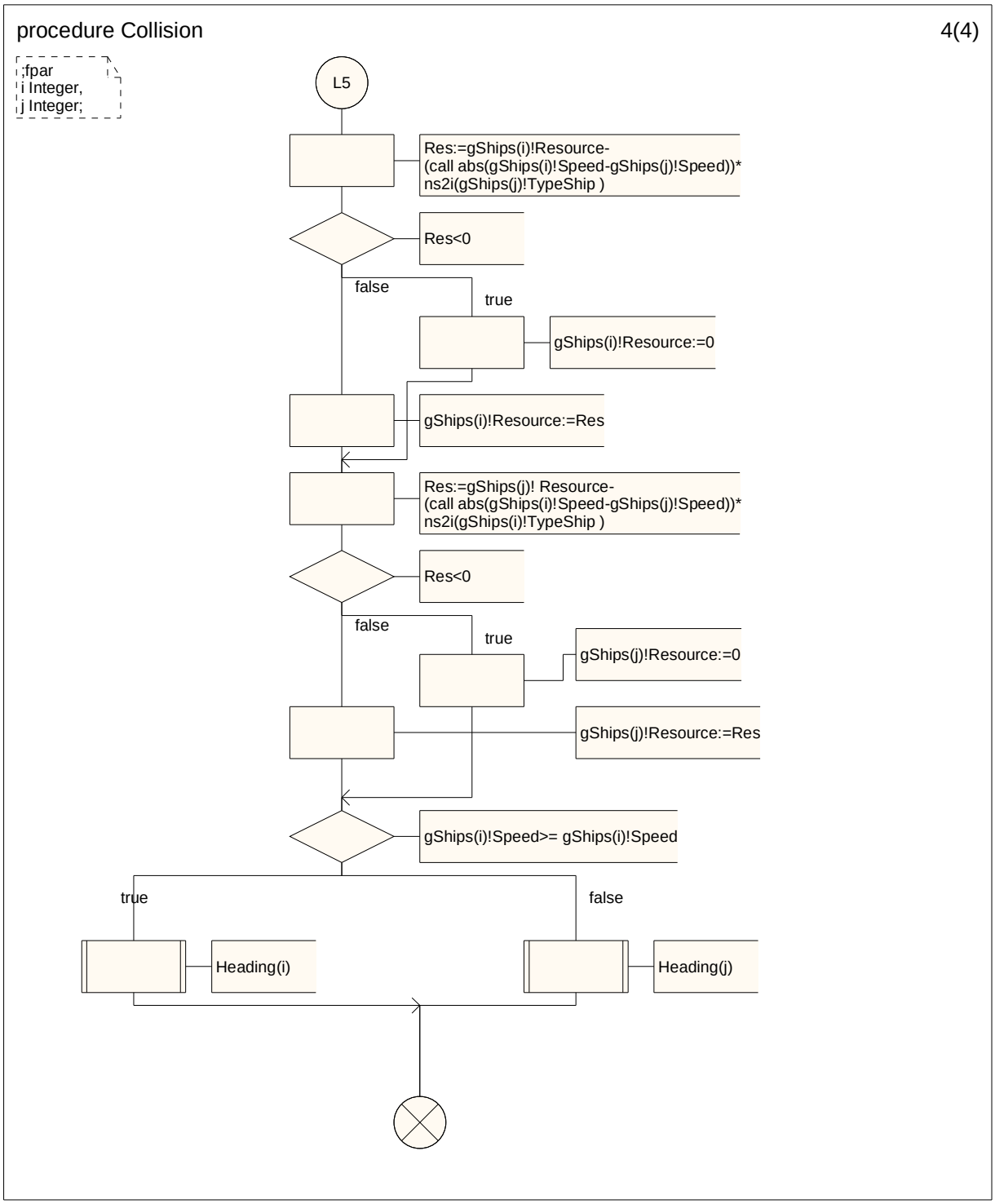

Рисунок 96. Процедура Collision (4).

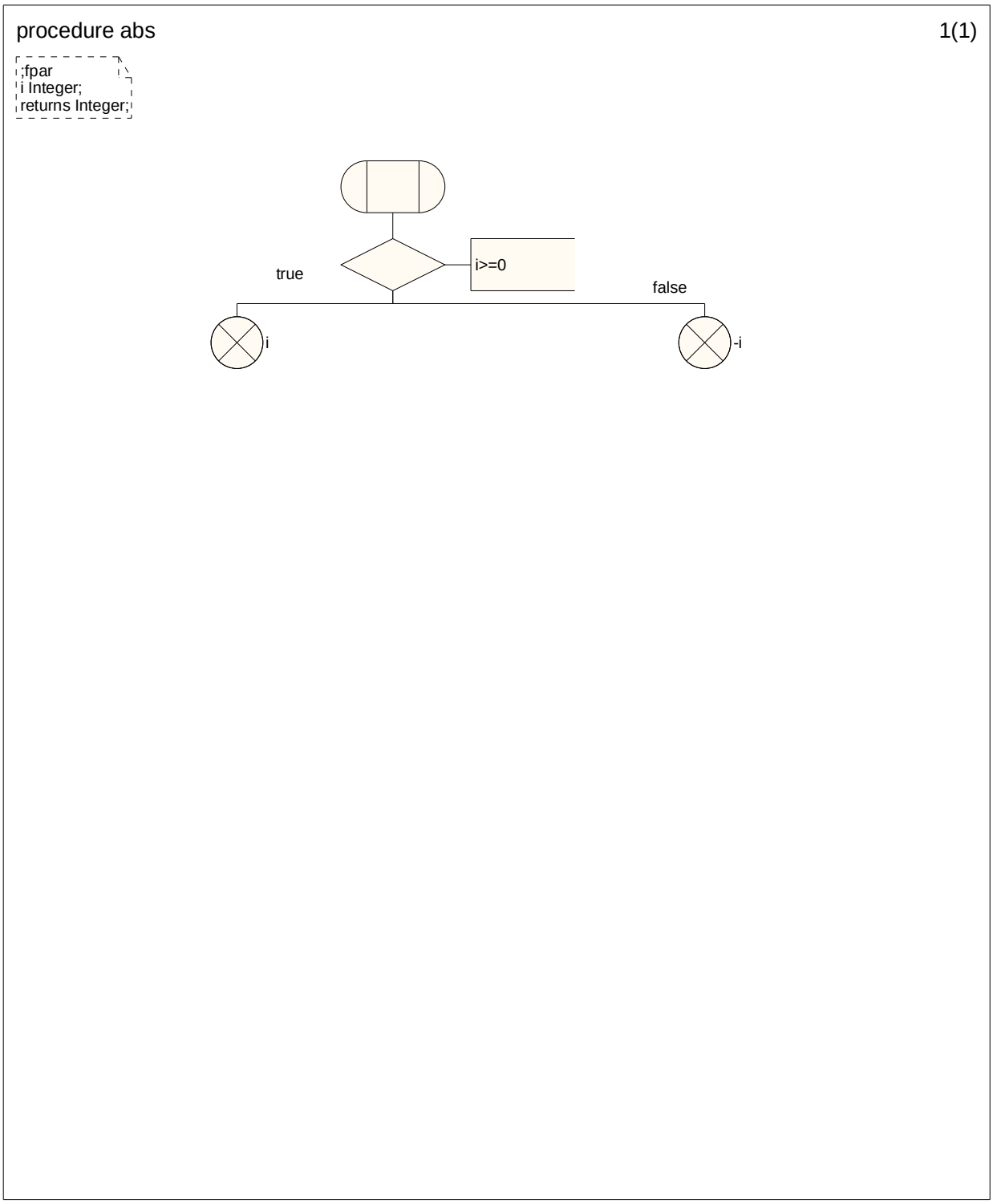

Рисунок 97. Процедура abs.

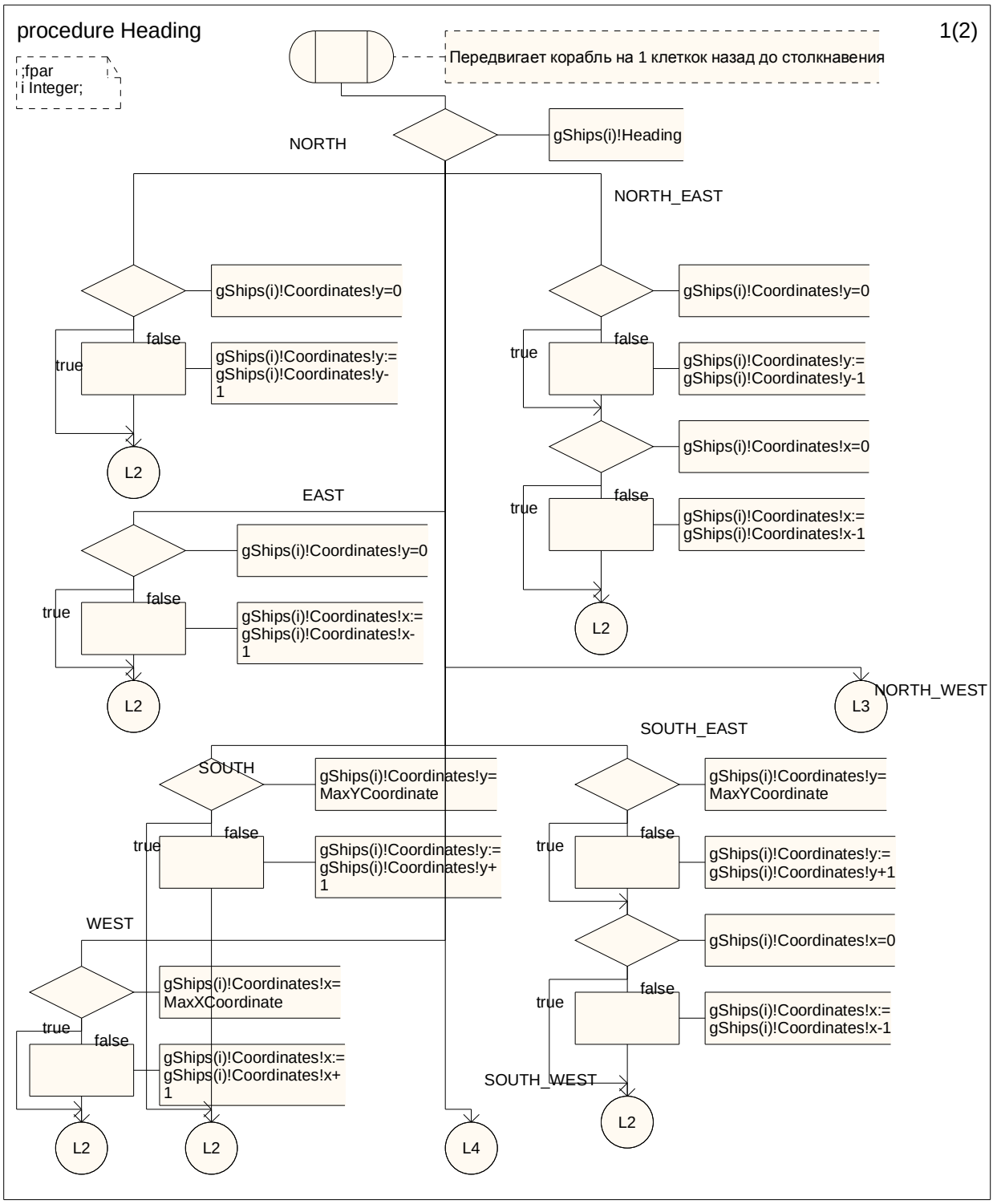

Рисунок 98. Процедура Heading (1).

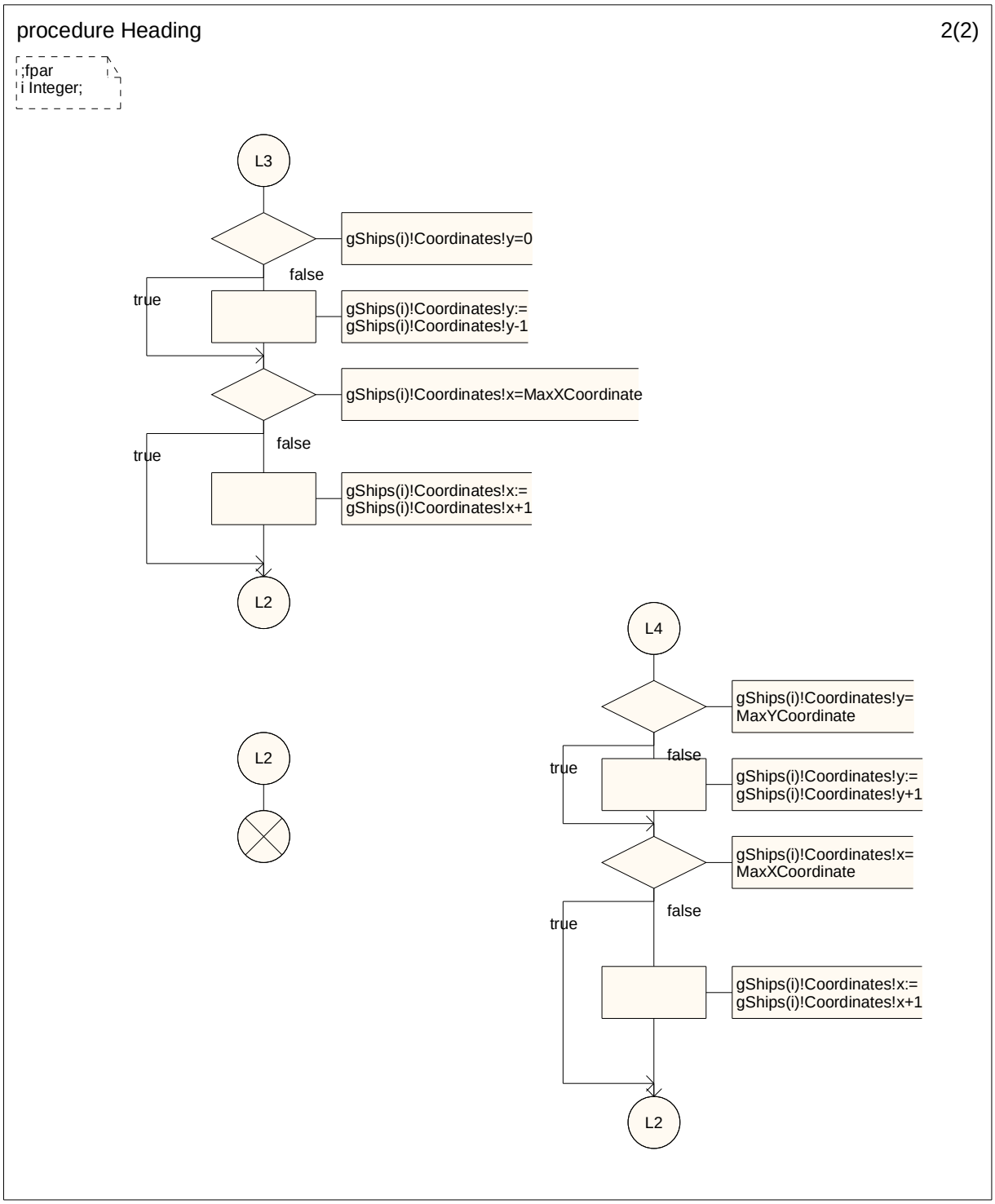

Рисунок 99. Процедура Heading (2).

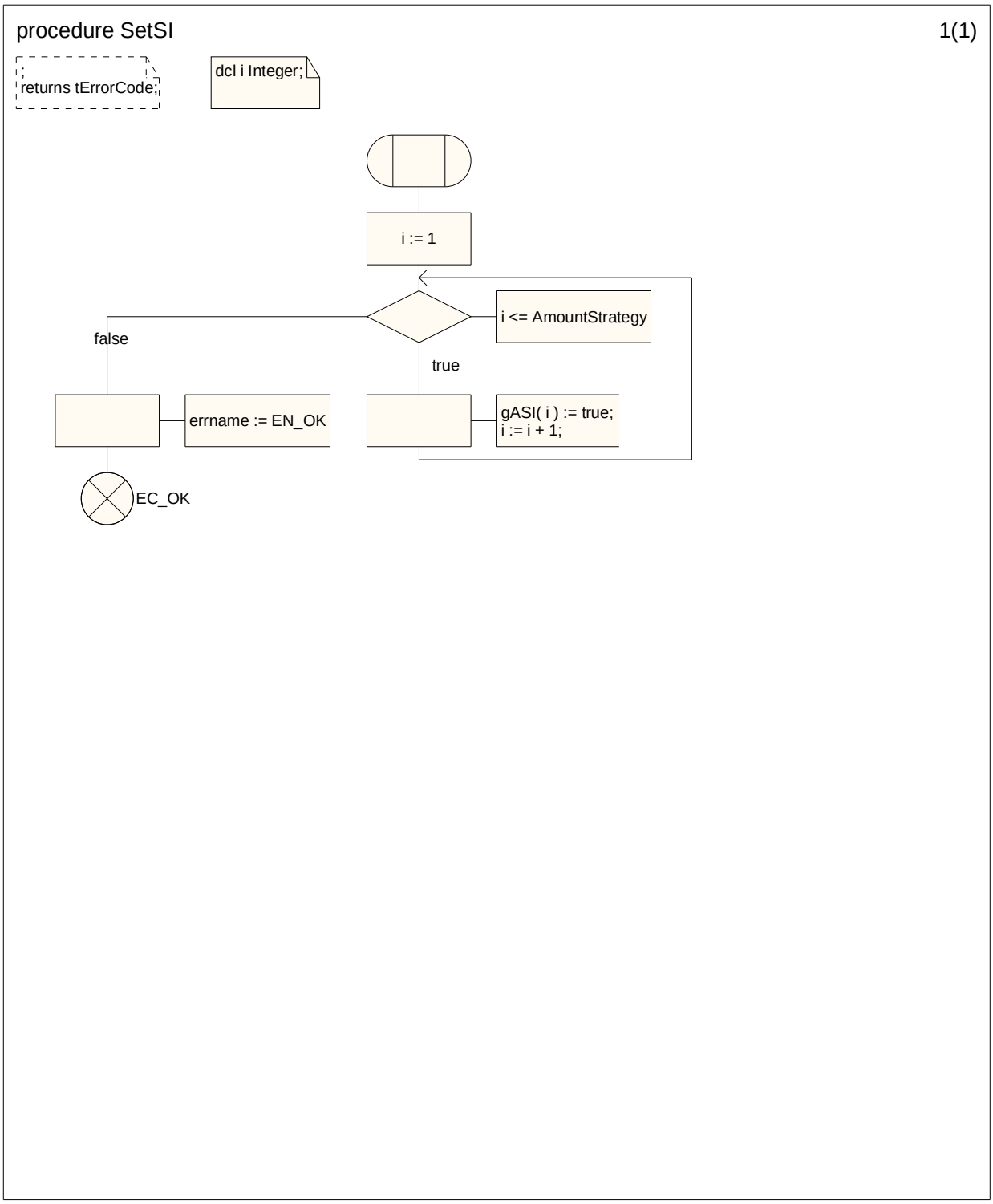

Рисунок 100. Процедура SetSI.

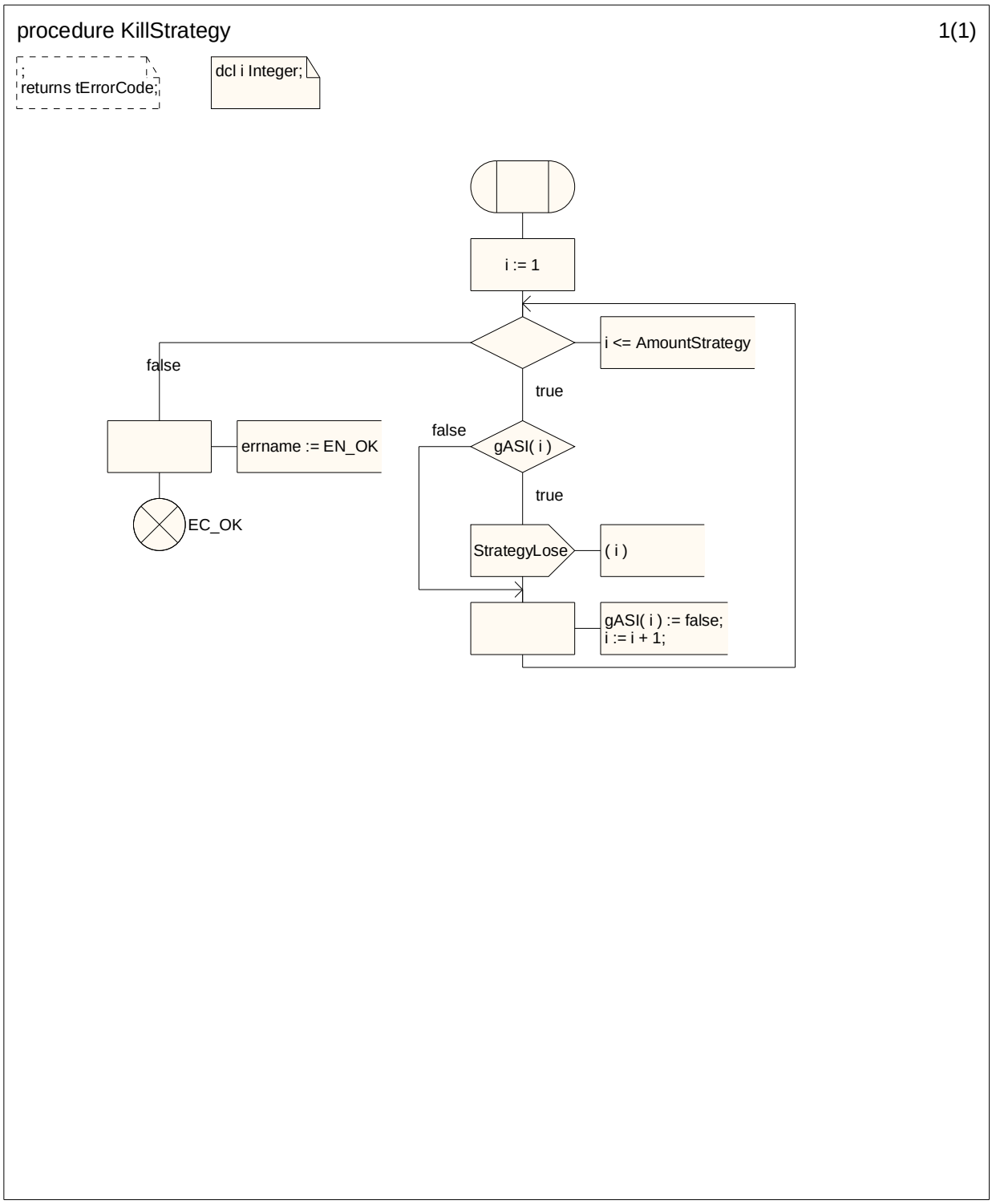

Рисунок 101. Процедура KillStrategy.

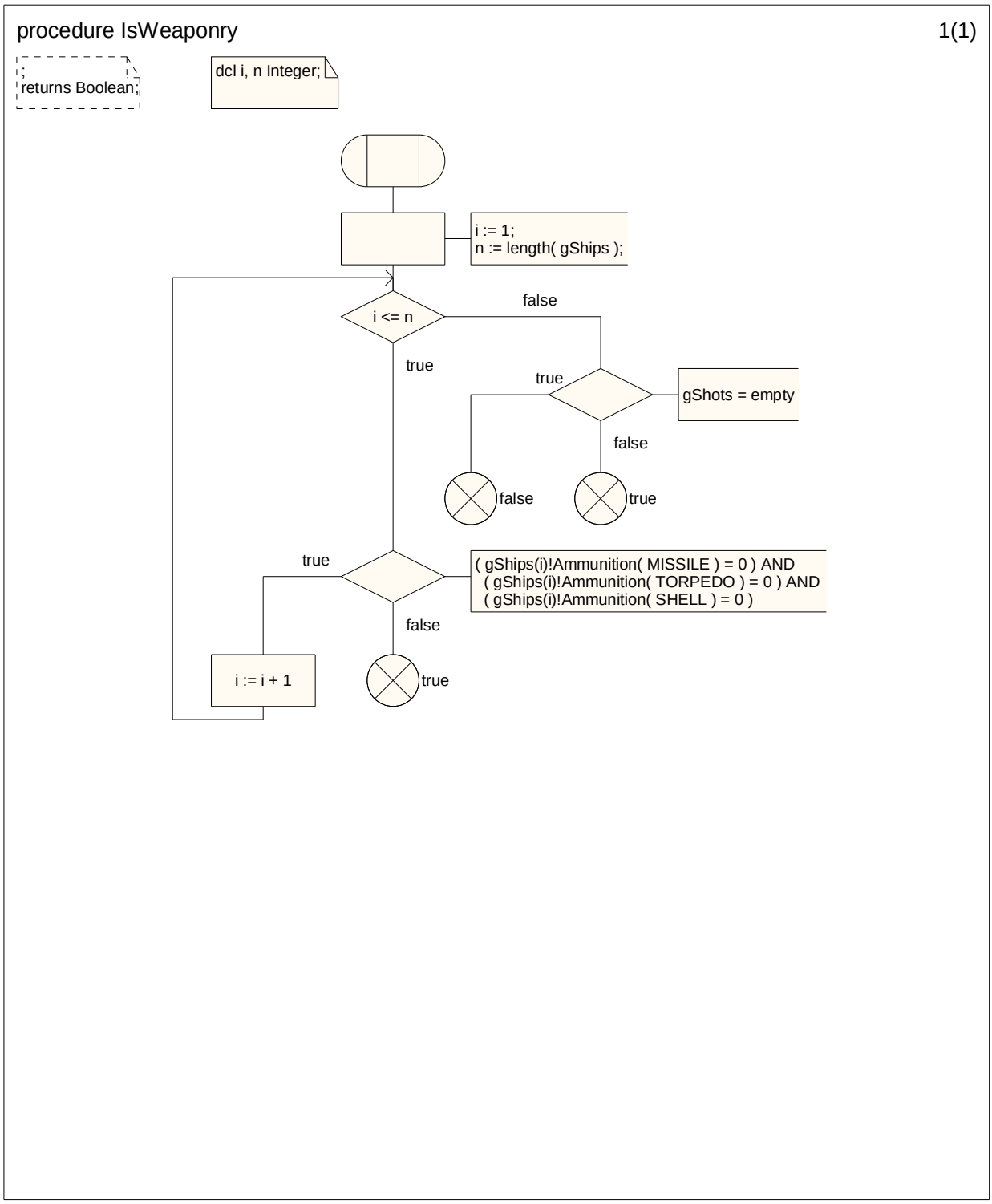

Рисунок 102. Процедура IsWeaponry.

Приложение D -- Код процесса DataSender (блок симулятора).

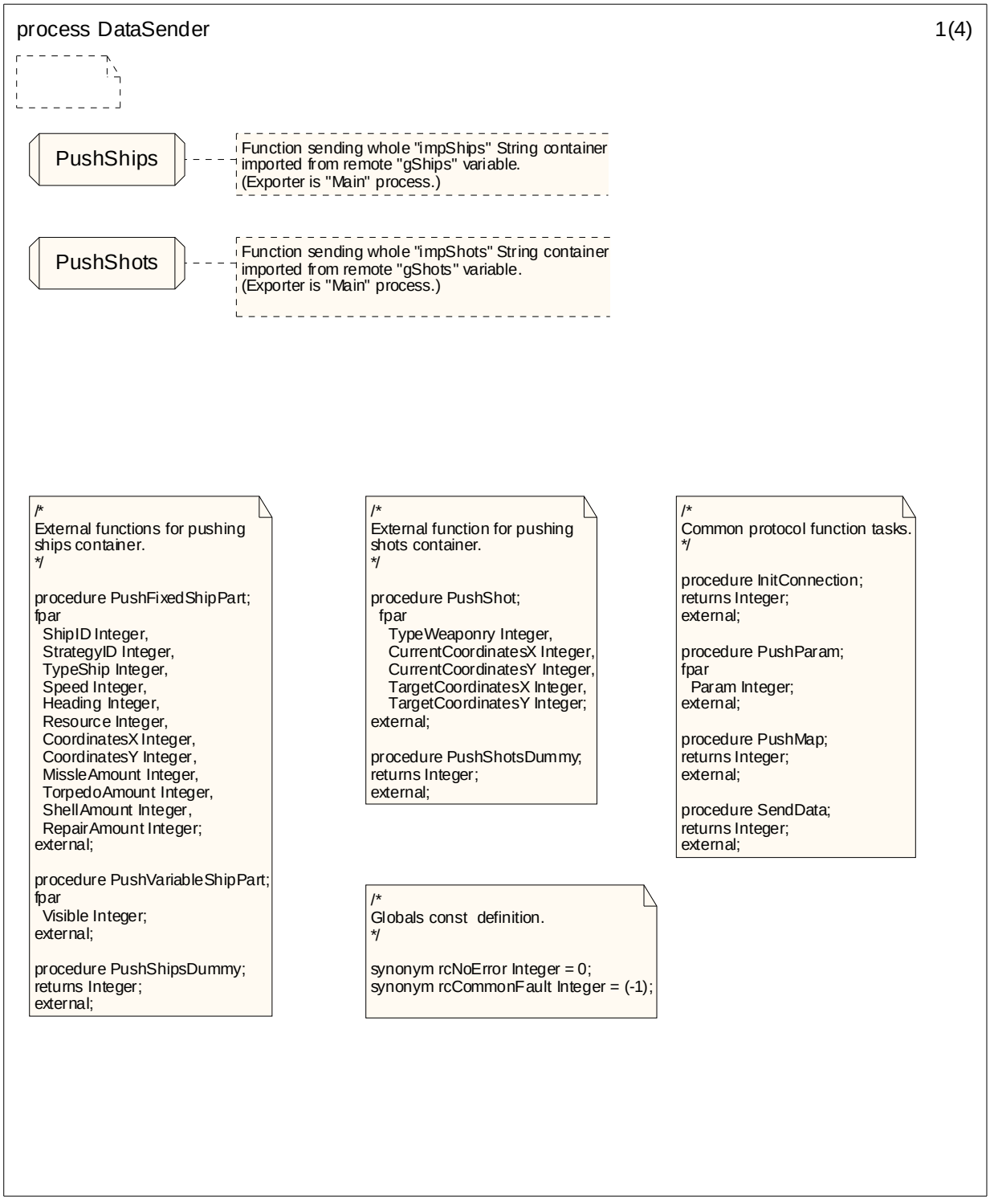

Рисунок 103. Процесс DataSender (1).

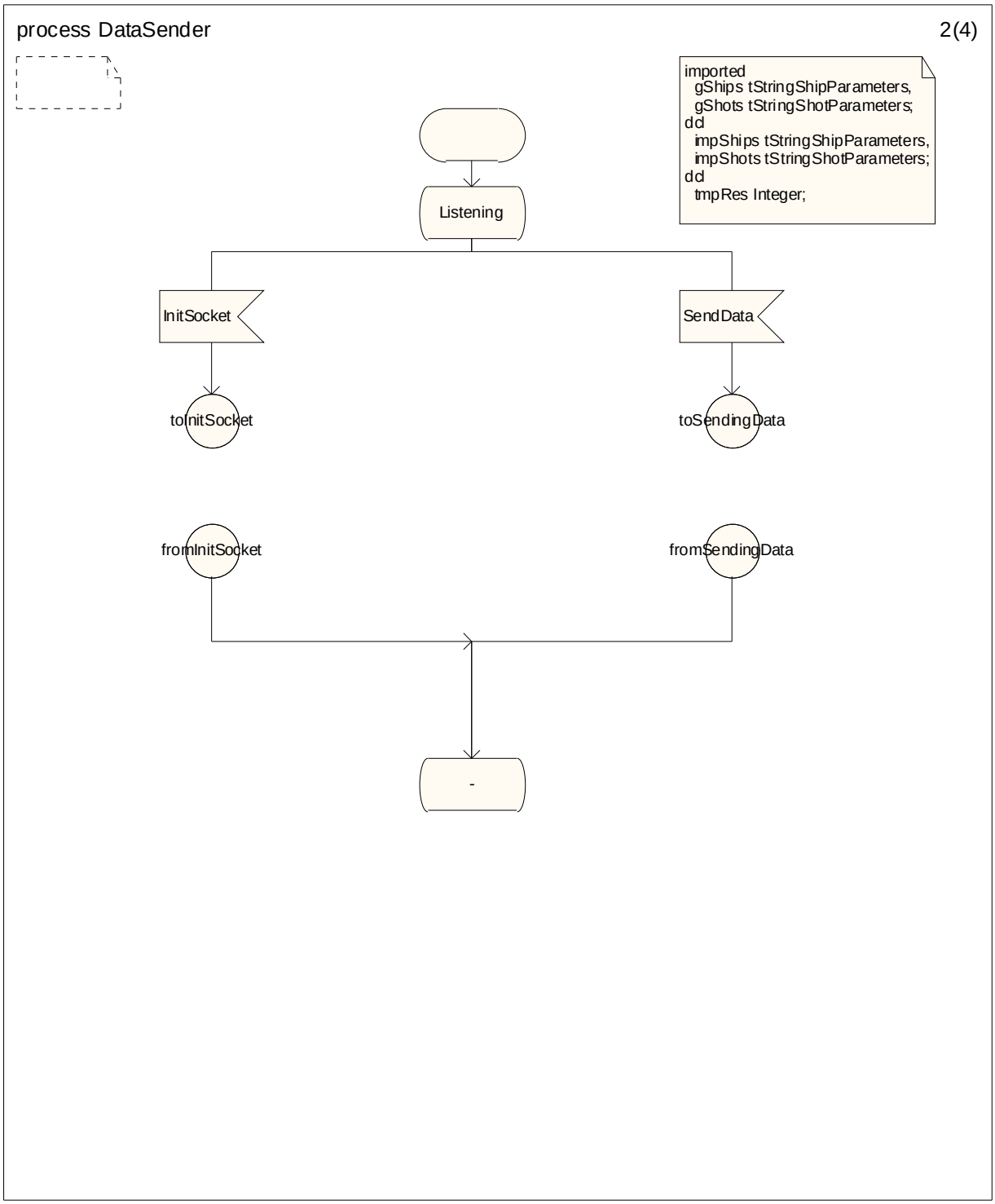

Рисунок 104. Процесс DataSender (2).

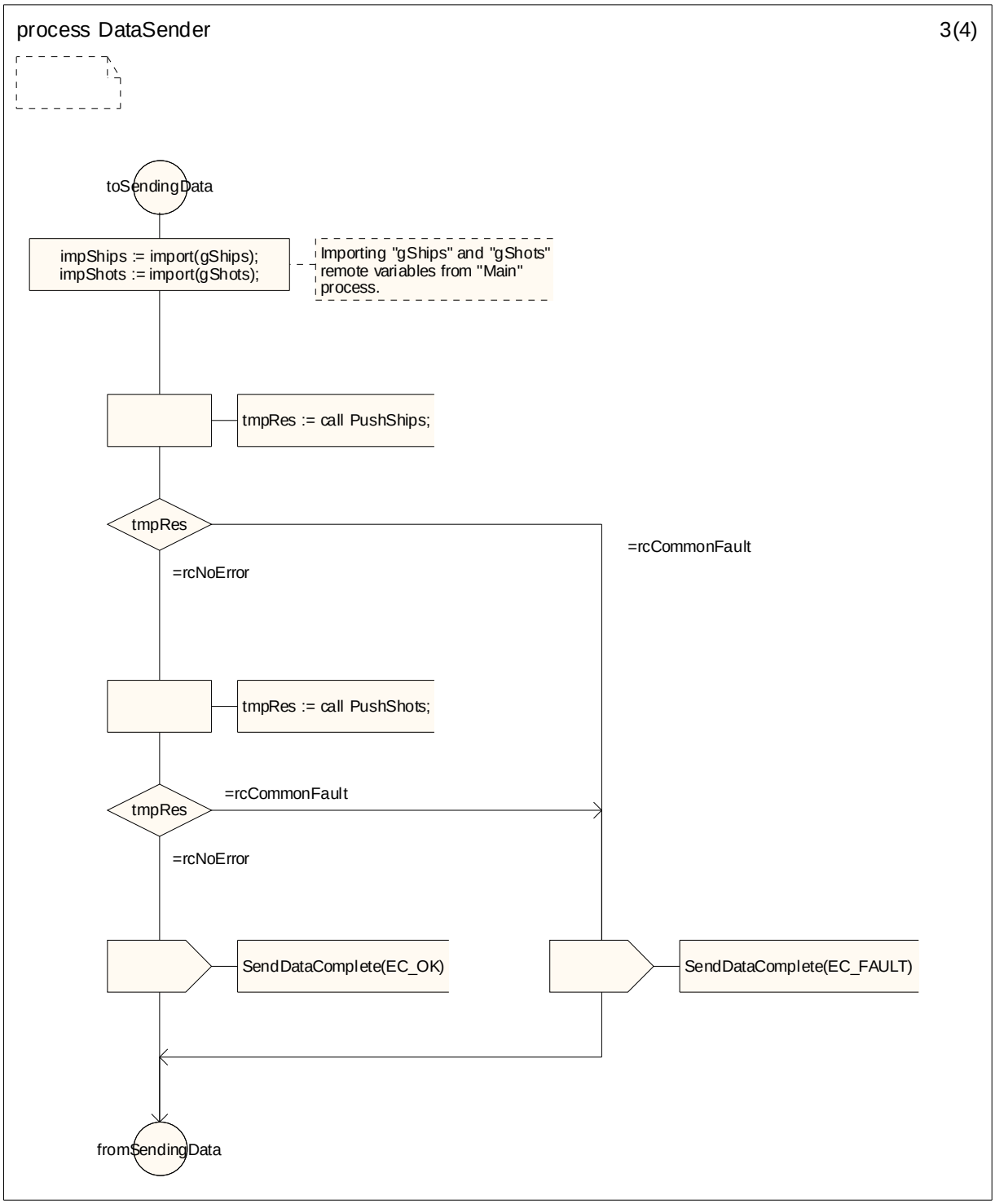

Рисунок 105. Процесс DataSender (3).

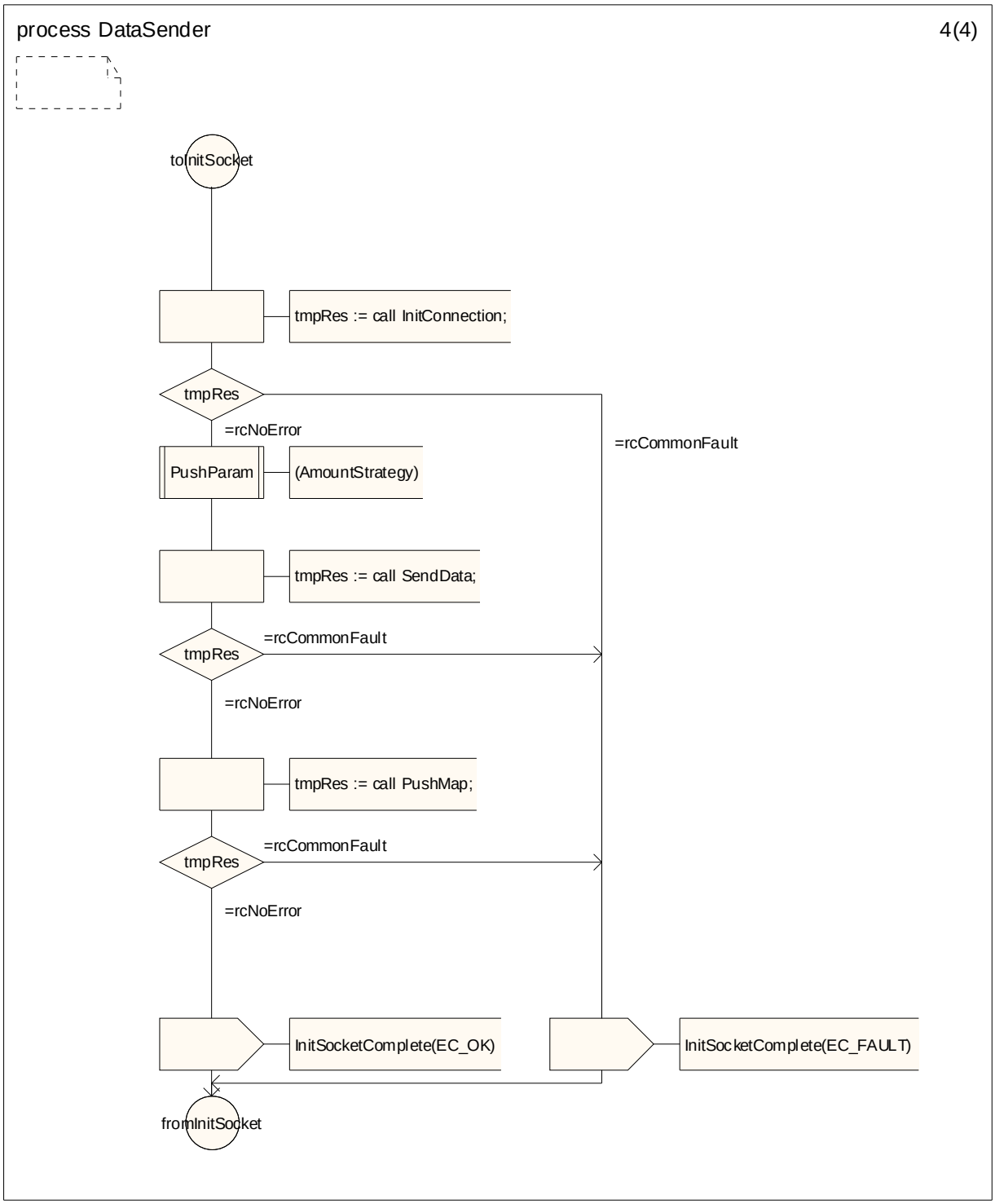

Рисунок 106. Процесс DataSender (4).

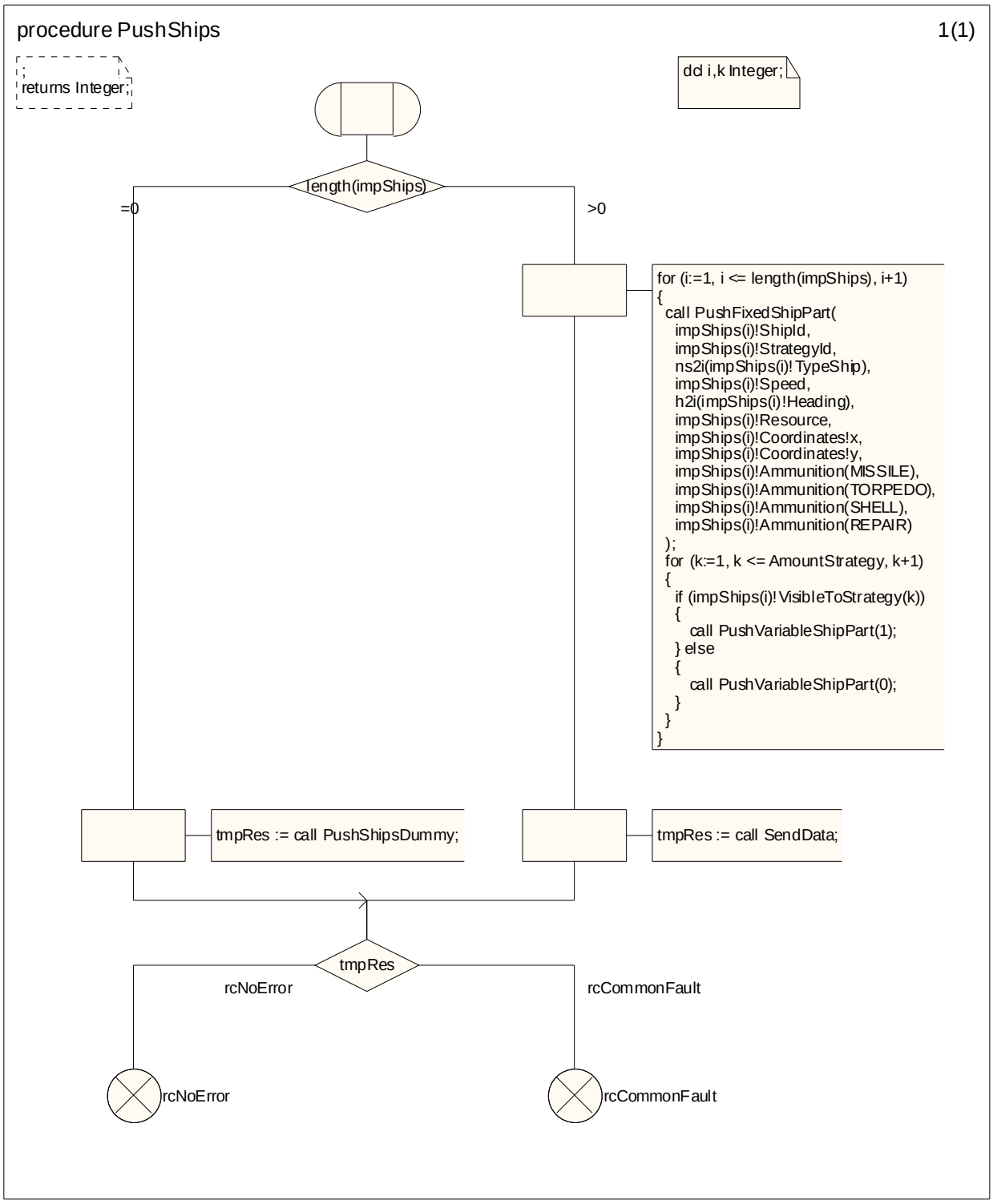

Рисунок 107. Процедура PushShips.

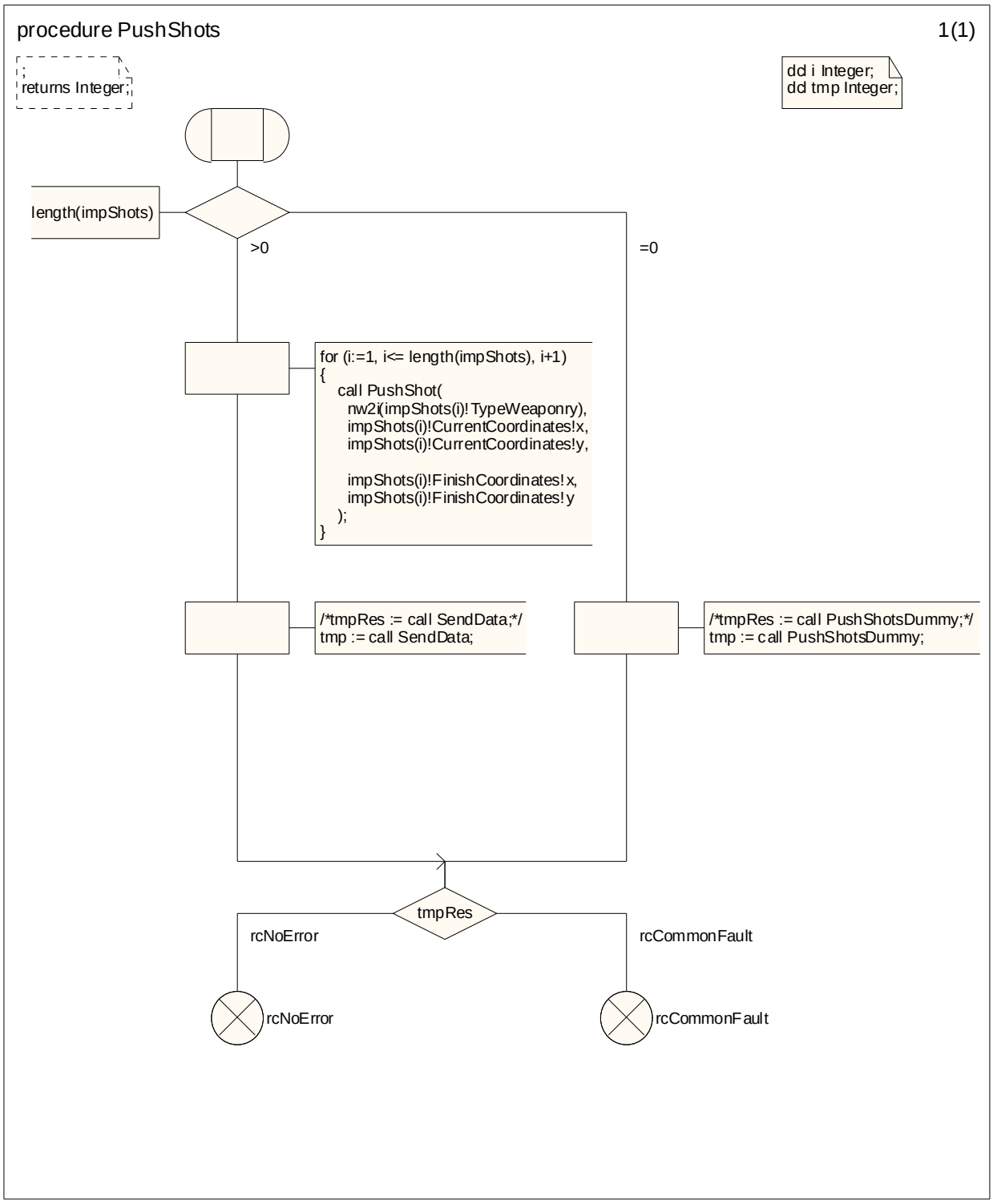

Рисунок 108. Процедура PushShots.

Приложение E -- Код процесса Main1 (блок стратегии).

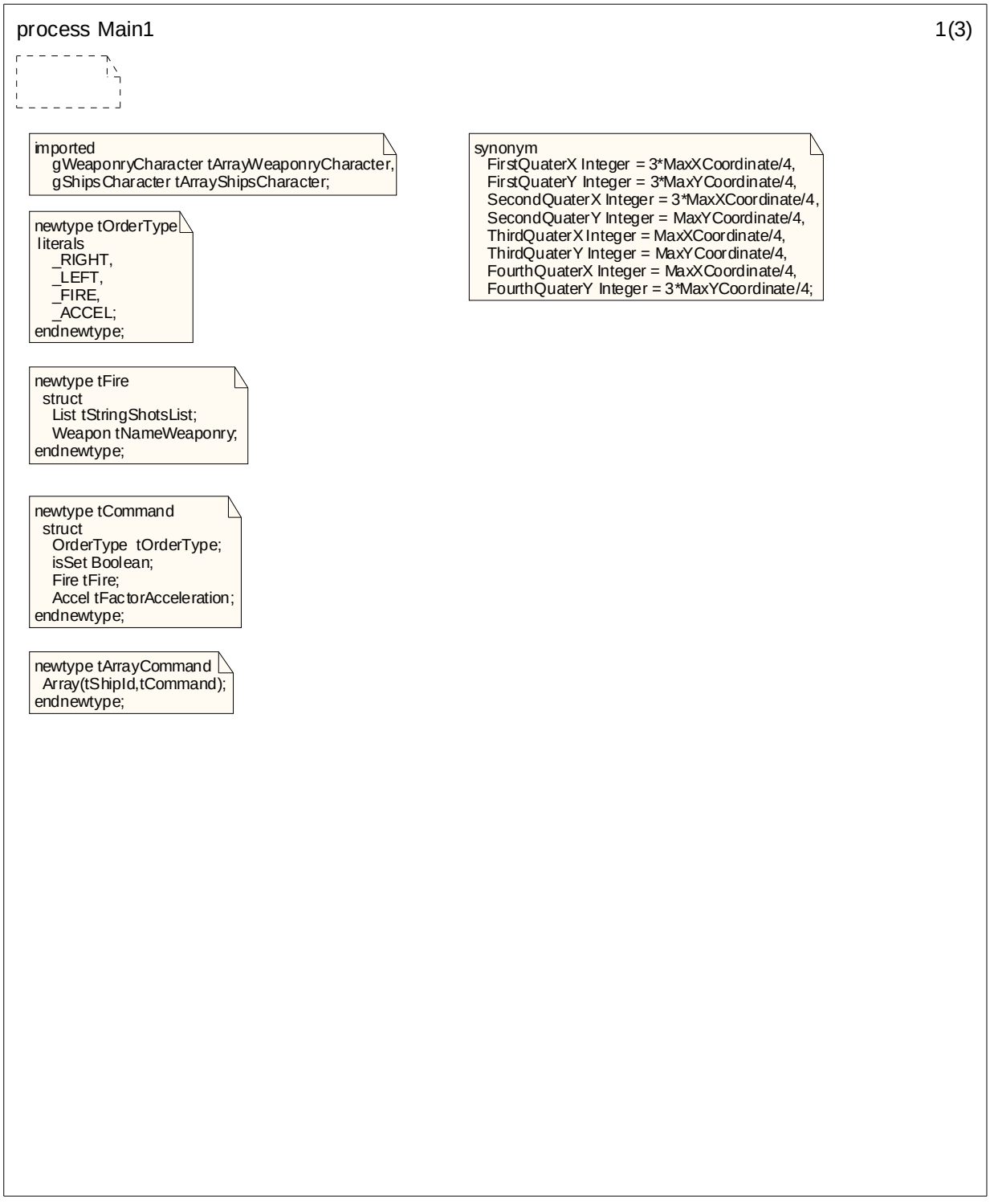

Рисунок 109. Процесс Main1 (1).

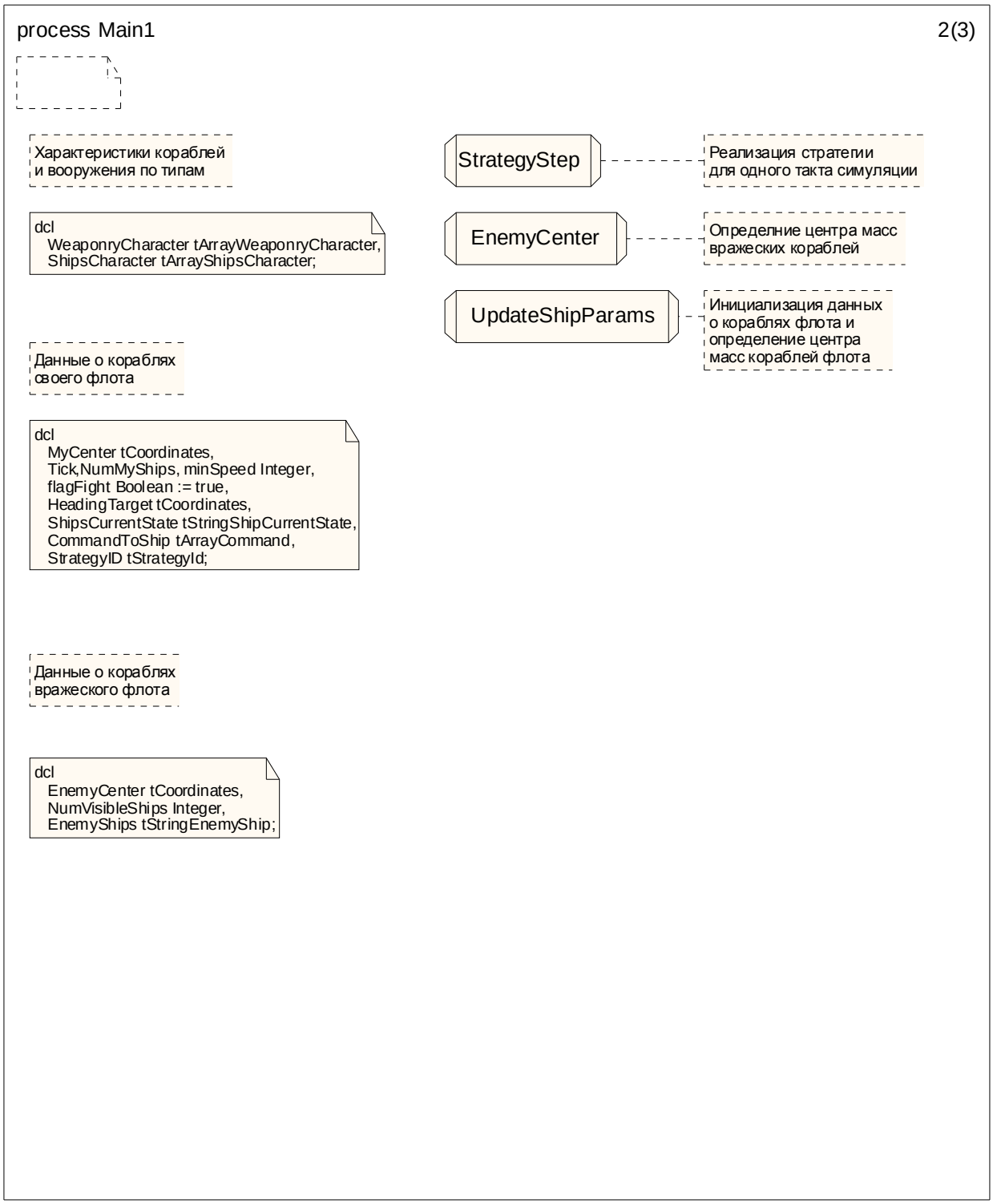

Рисунок 110. Процесс Main1 (2).

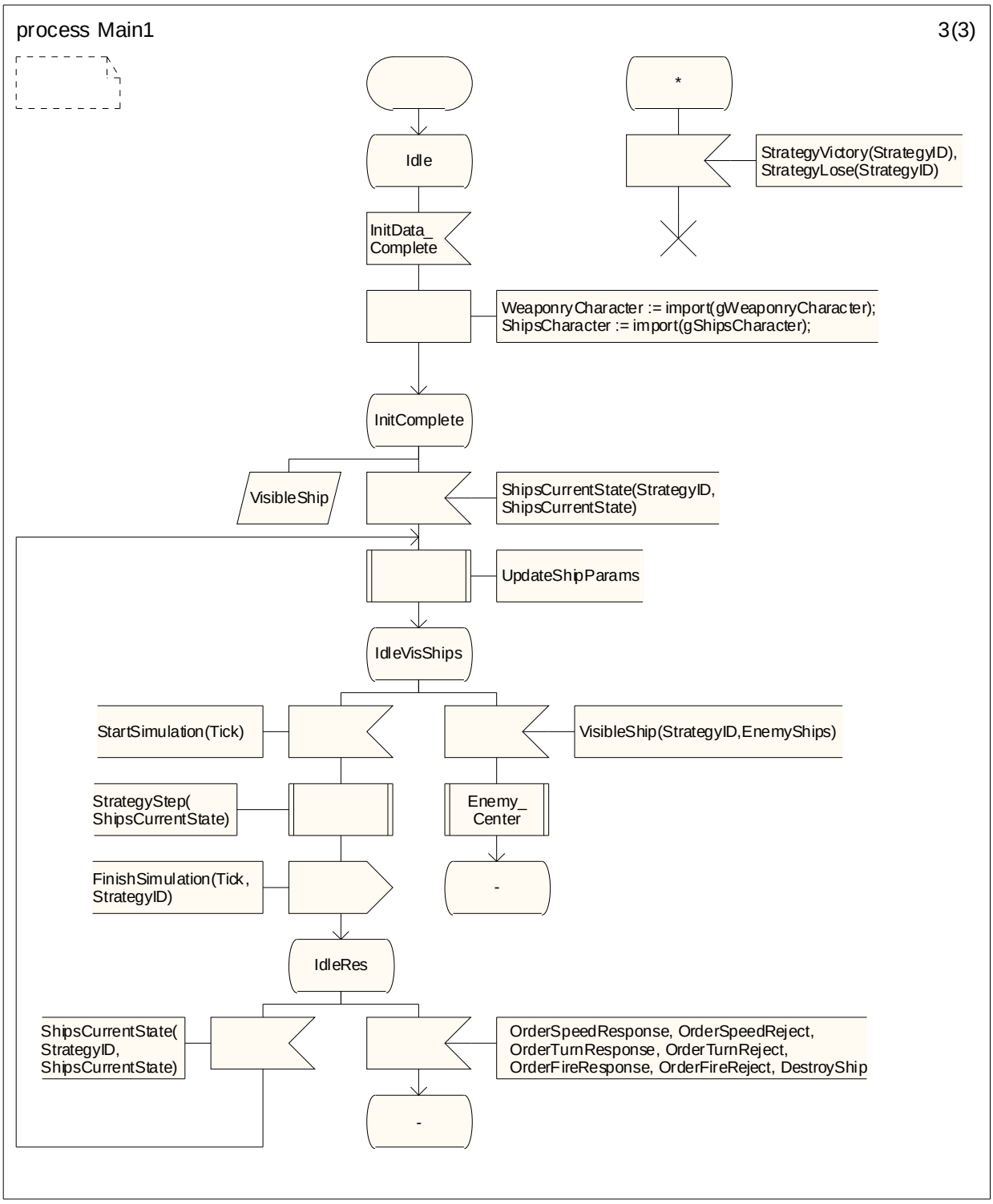

Рисунок 111. Процесс Main1 (3).

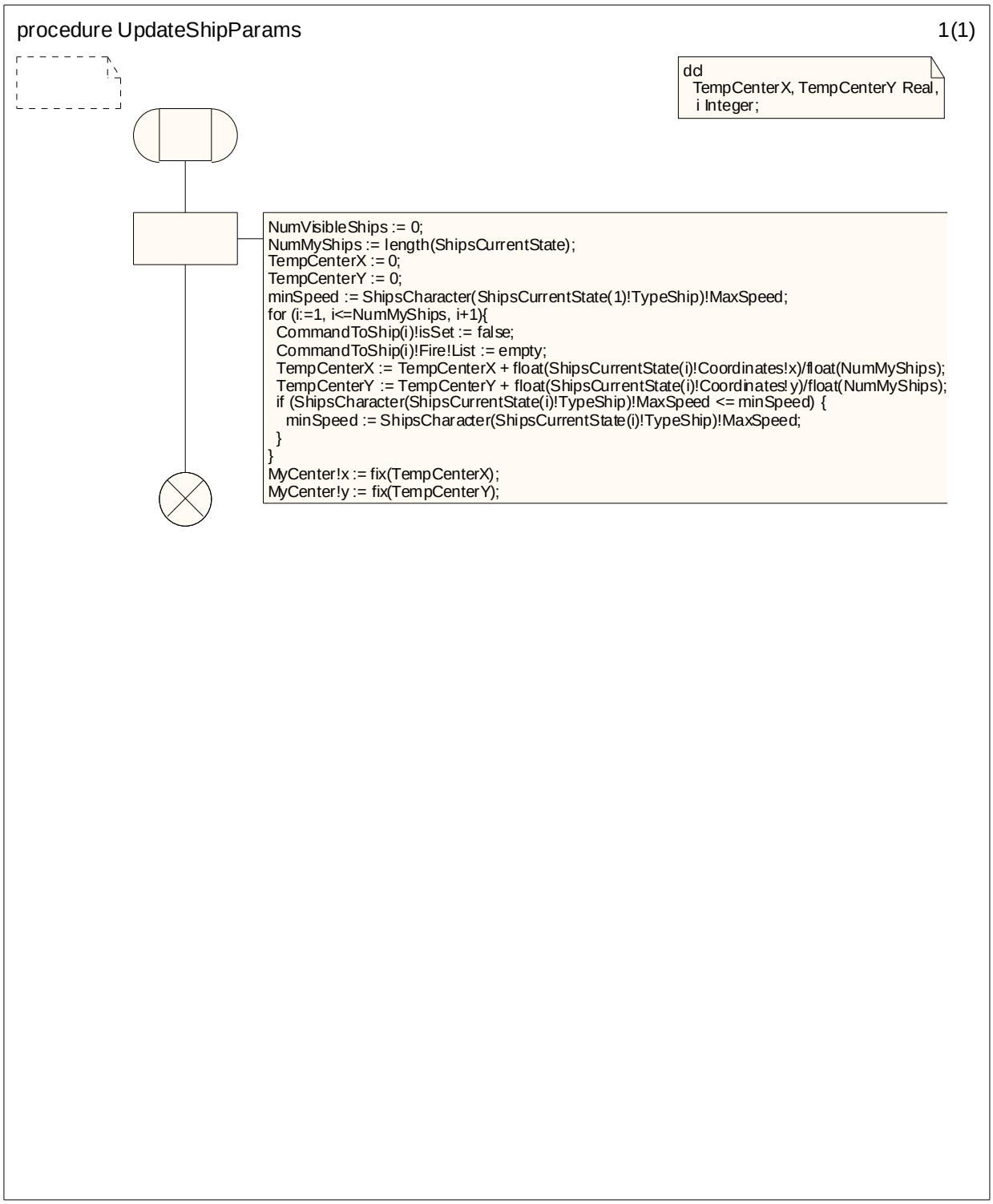

Рисунок 112. Процедура UpdateShipParams.

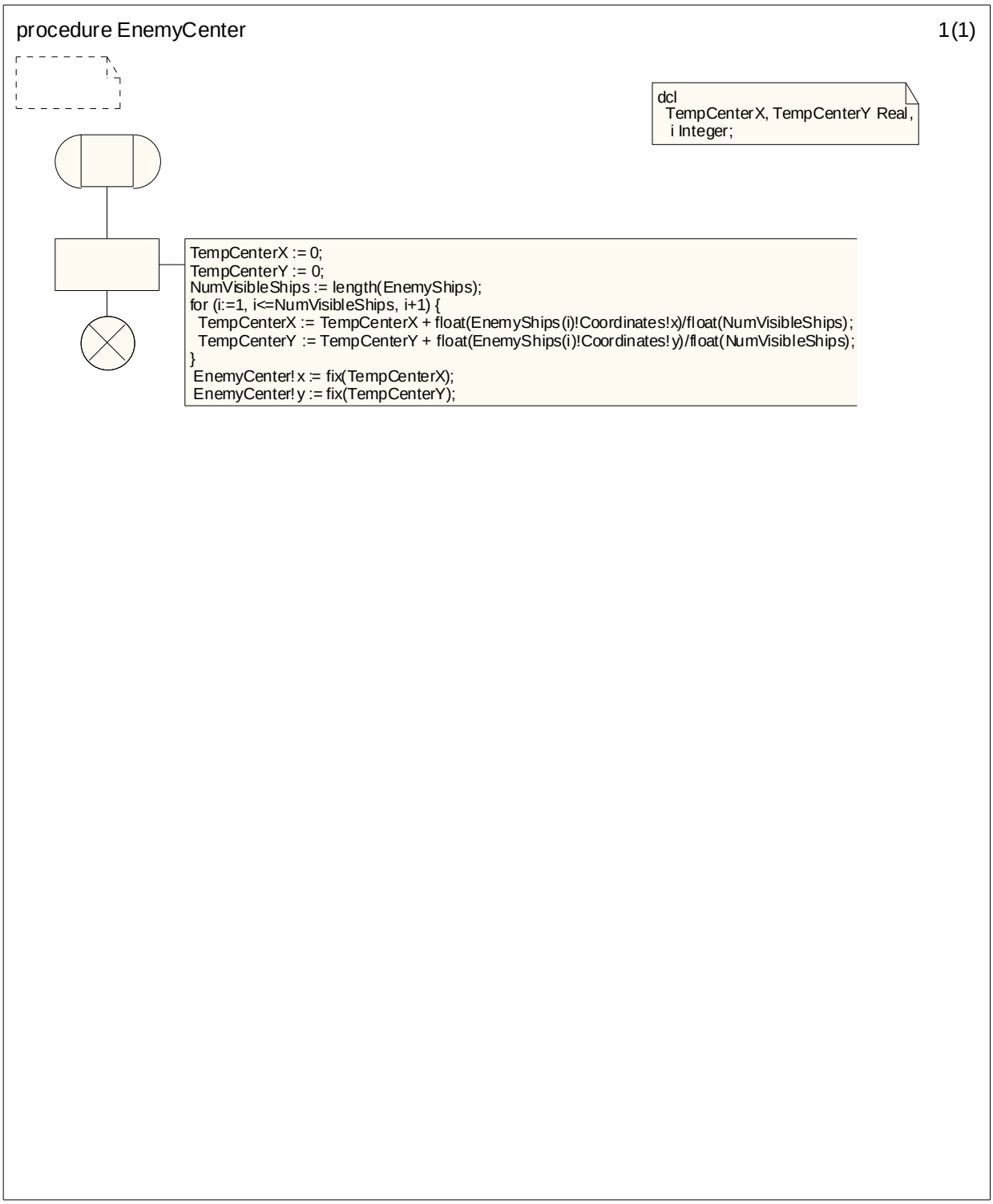

Рисунок 113. Процедура EnemyCenter.

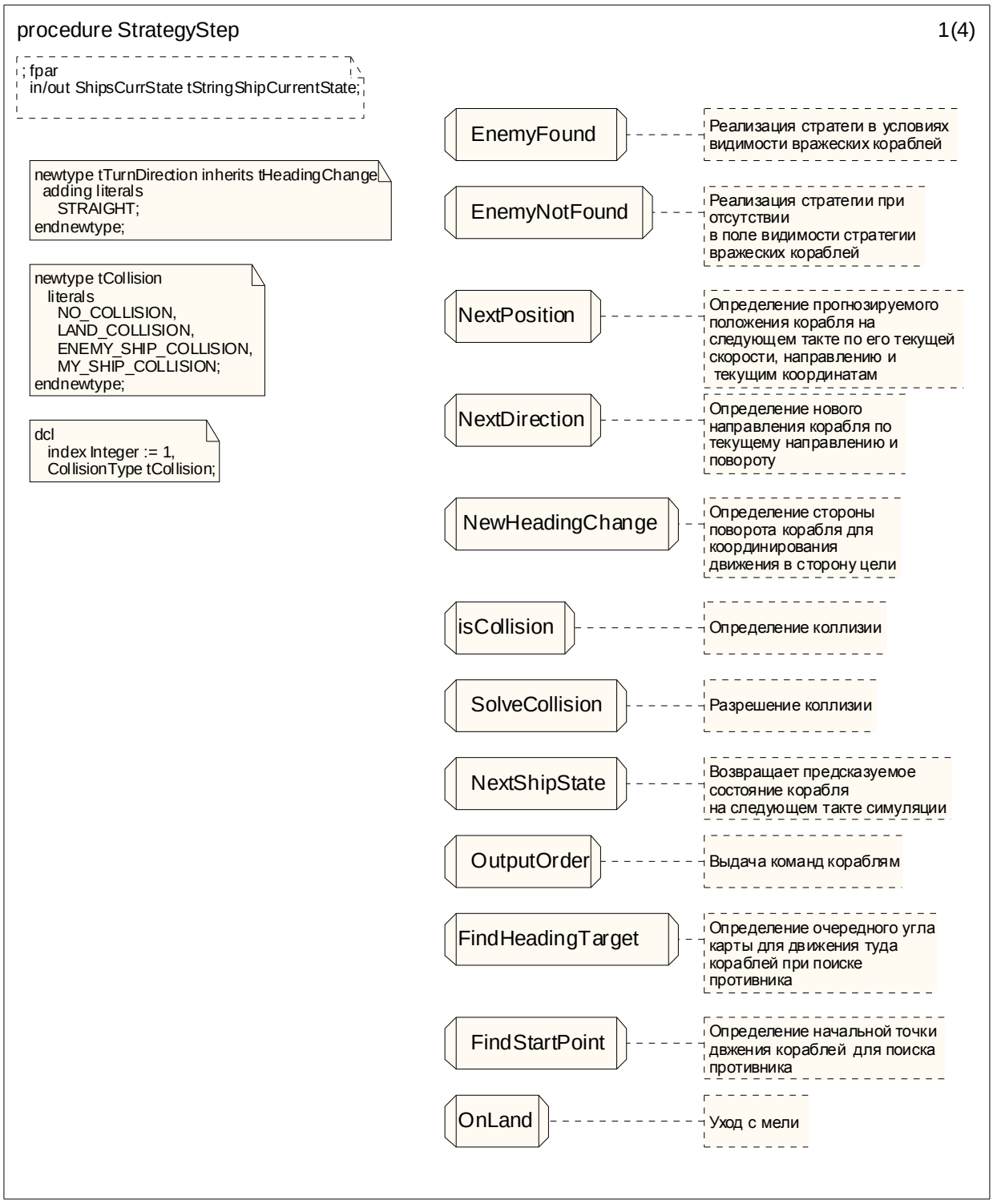

Рисунок 114. Процедура StrategyStep (1).

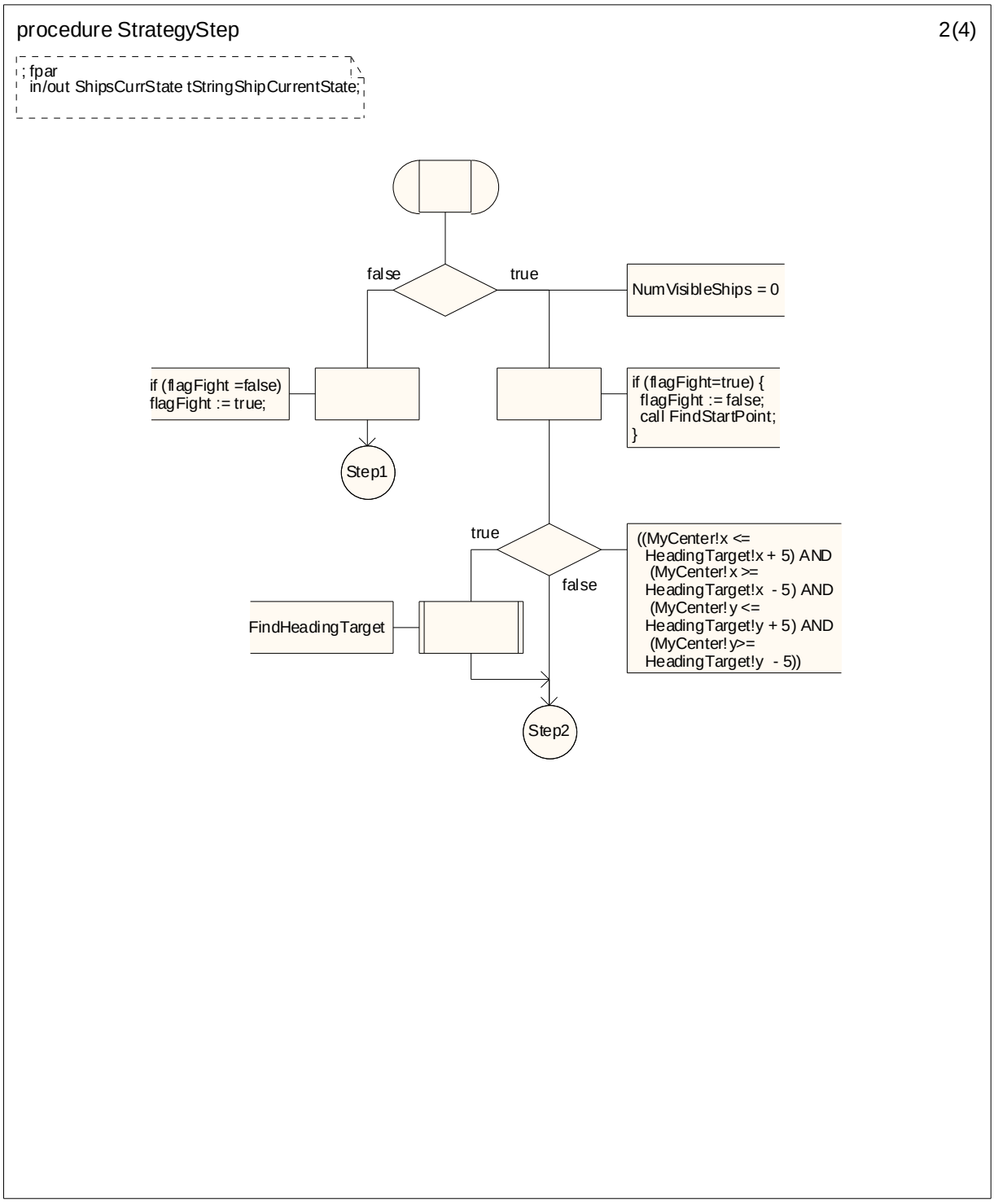

Рисунок 115. Процедура StrategyStep (2).

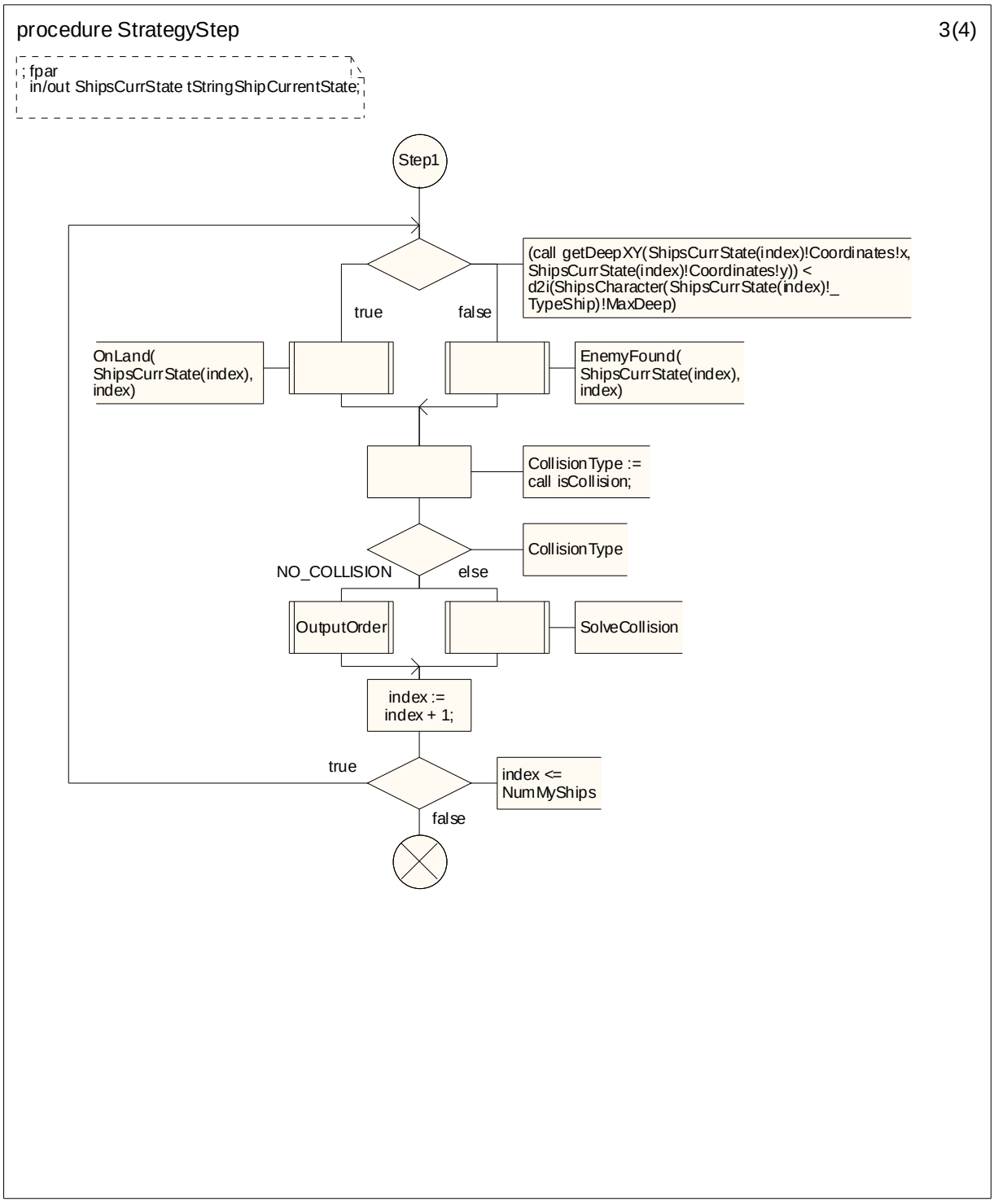

Рисунок 116. Процедура StrategyStep (3).

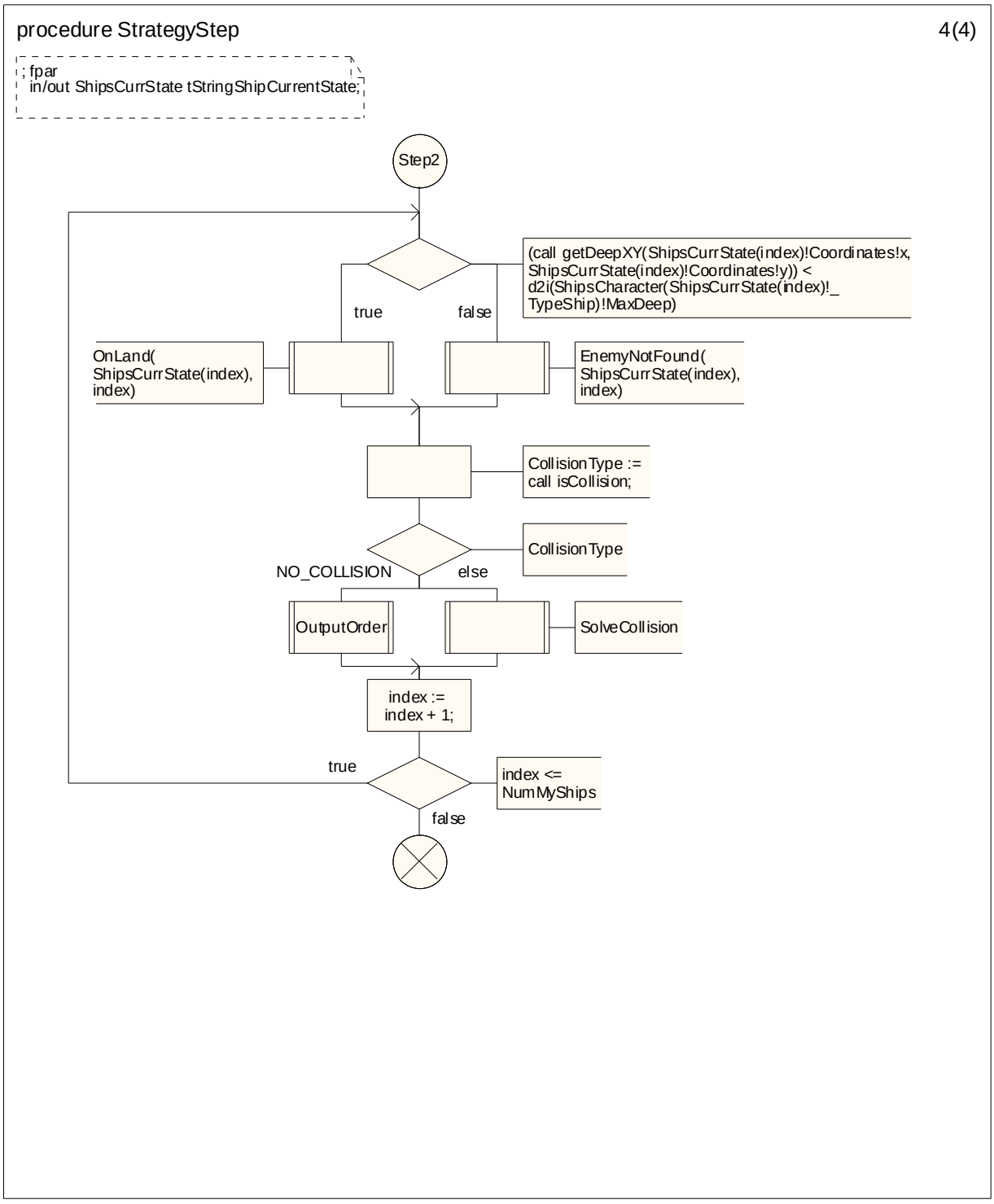

Рисунок 117. Процедура StrategyStep (4).

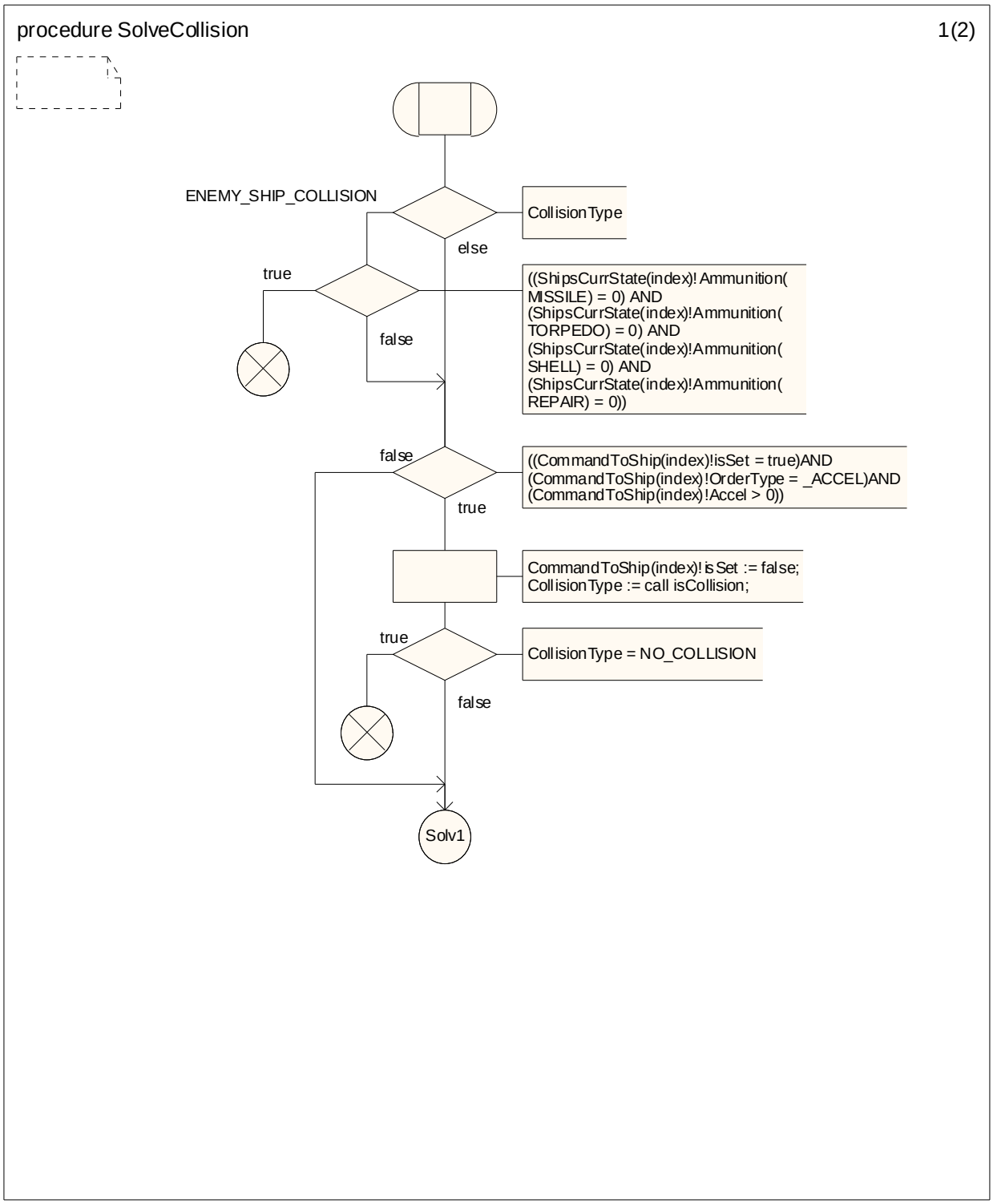

Рисунок 118. Процедура SolveCollision (1).

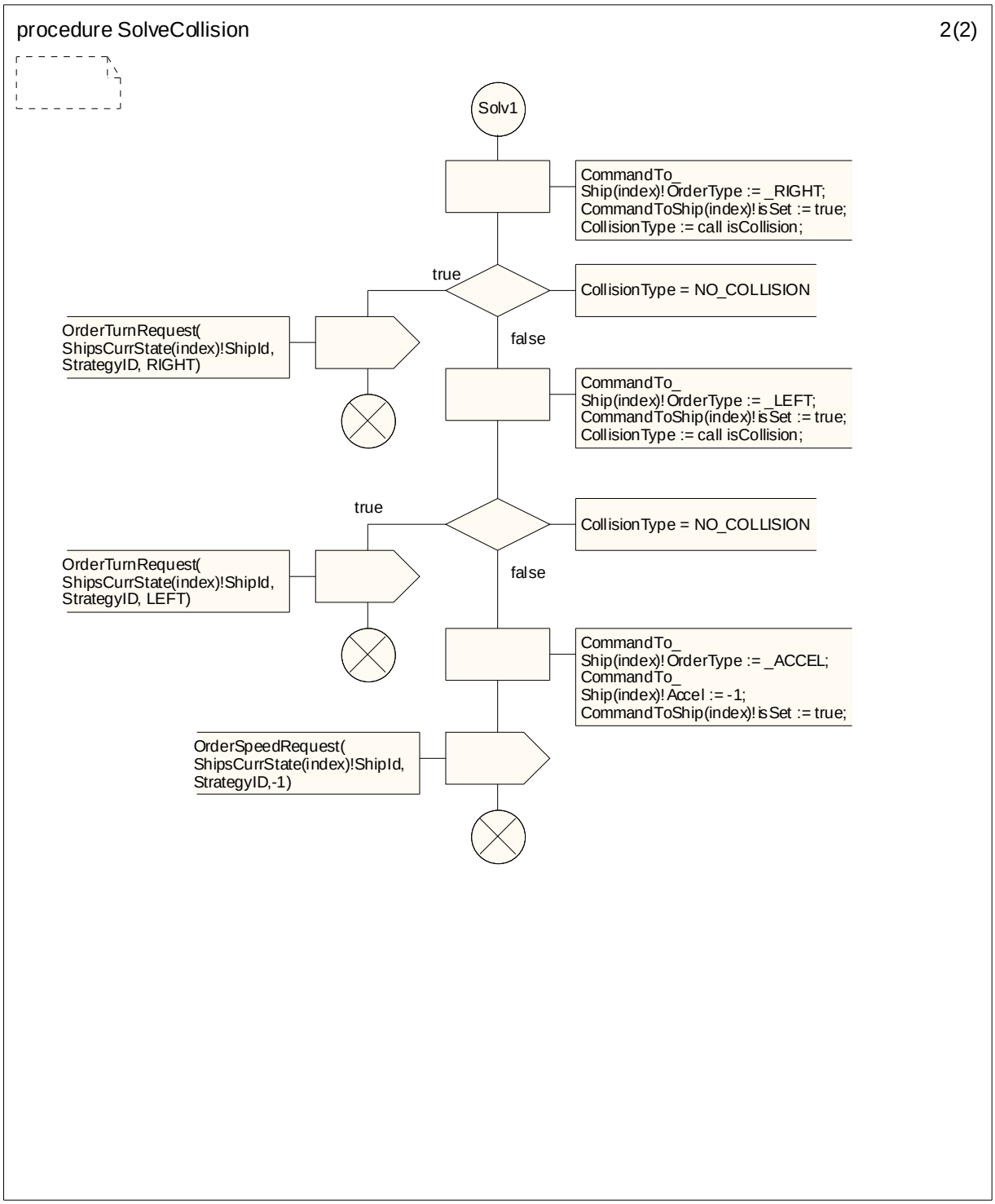

Рисунок 119. Процедура SolveCollision (2).

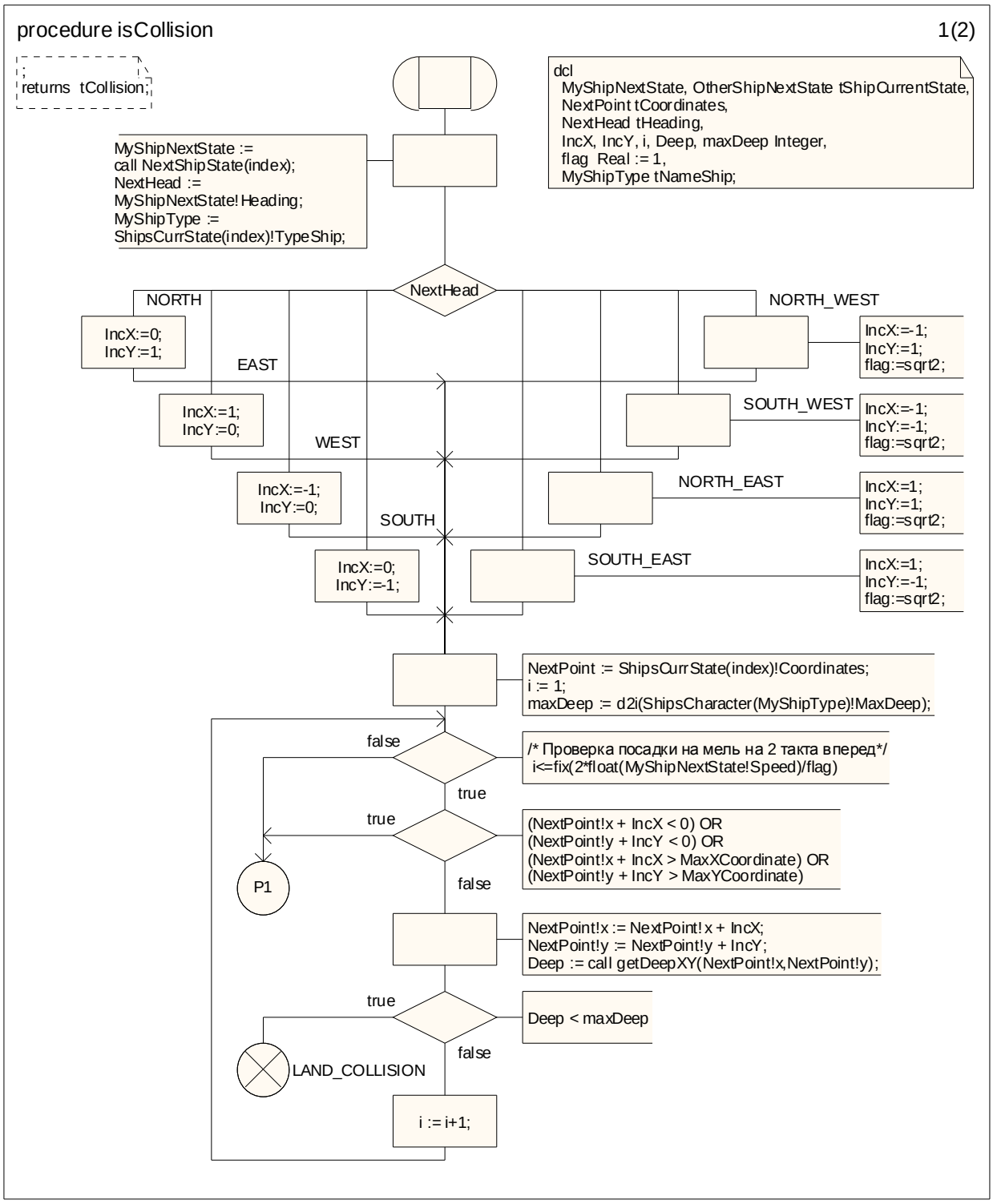

Рисунок 120. Процедура isCollision (1).

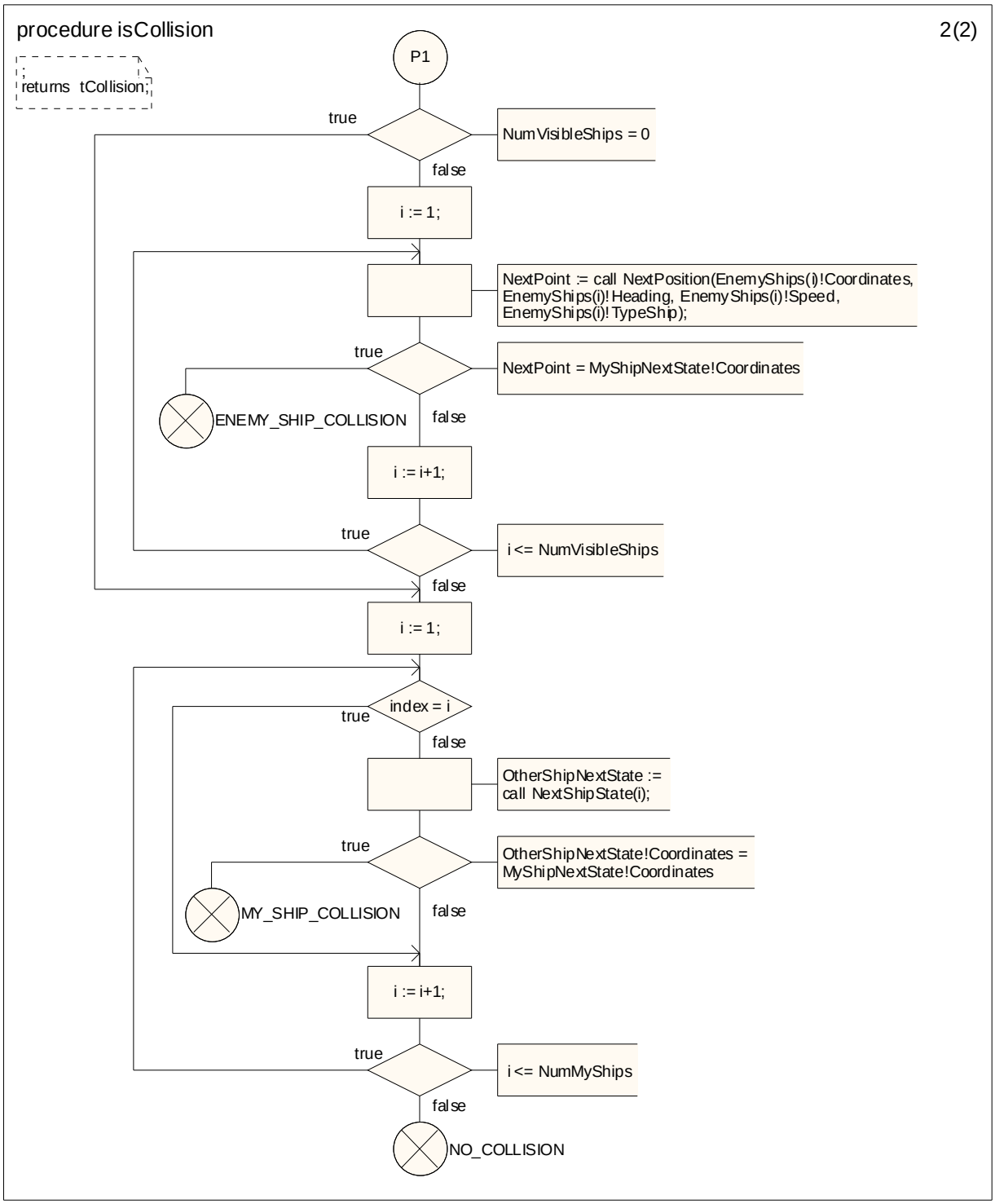

Рисунок 121. Процедура isCollision (2).

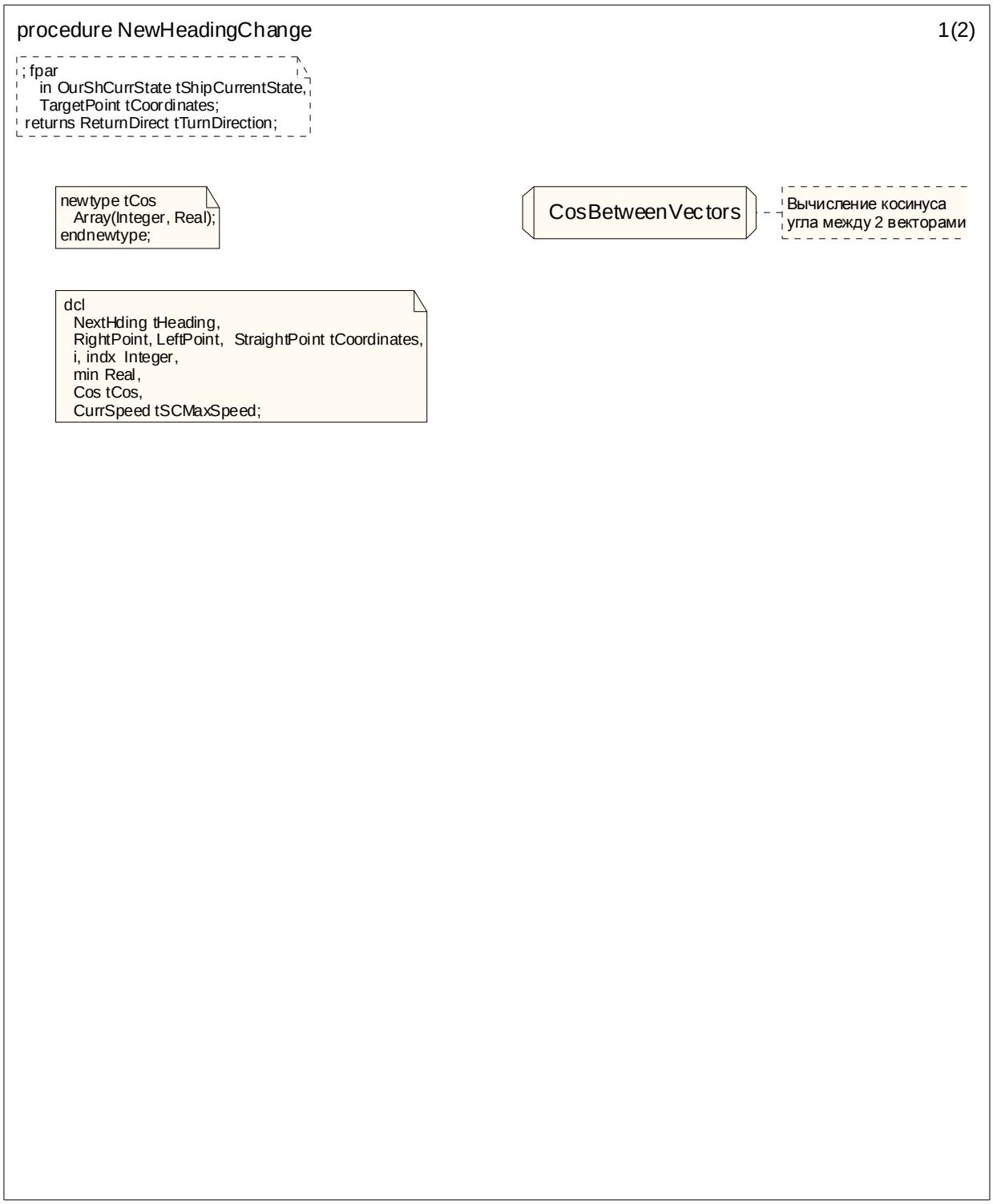

Рисунок 122. Процедура NewHeadingChange (1).

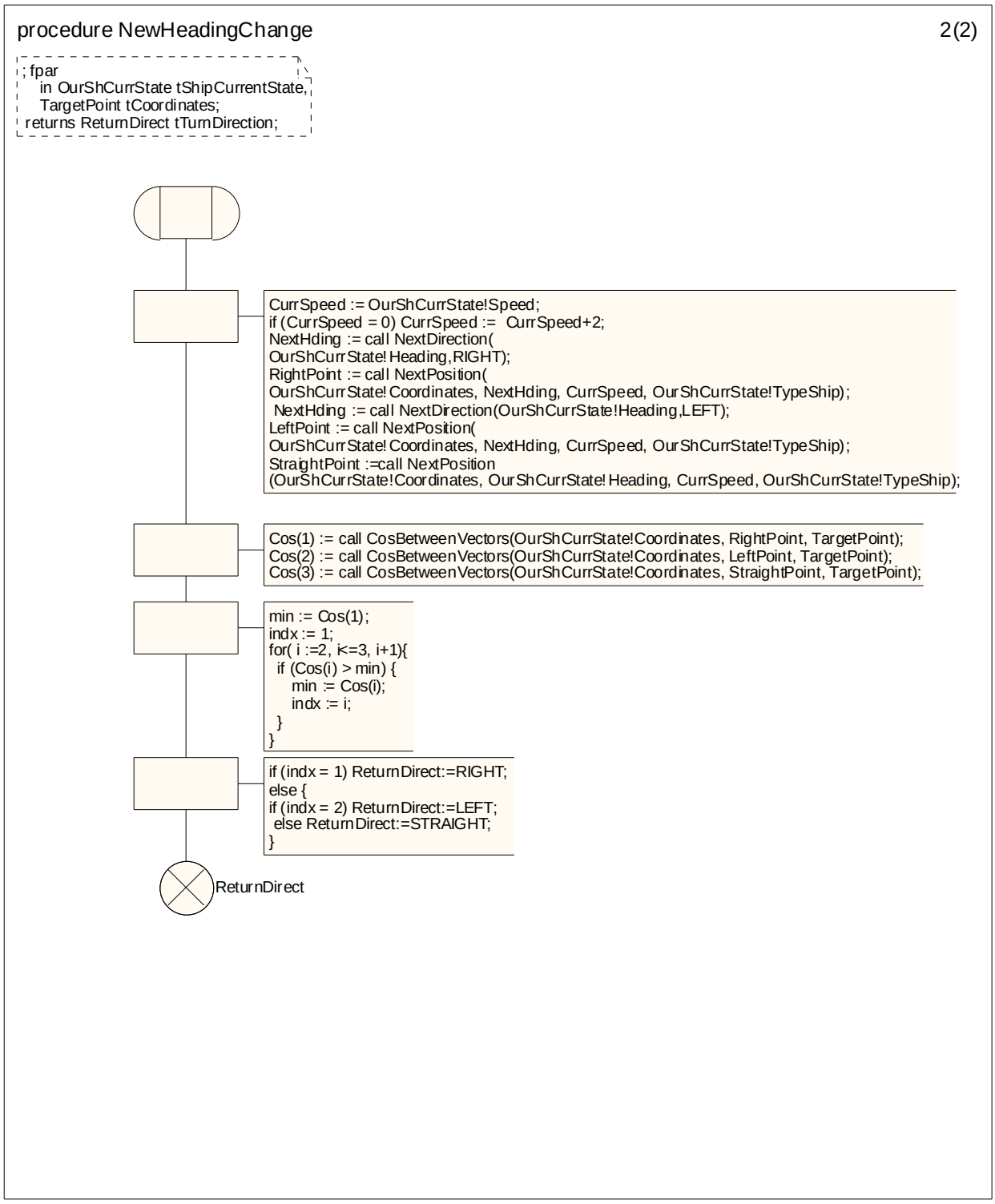

Рисунок 123. Процедура NewHeadingChange (2).

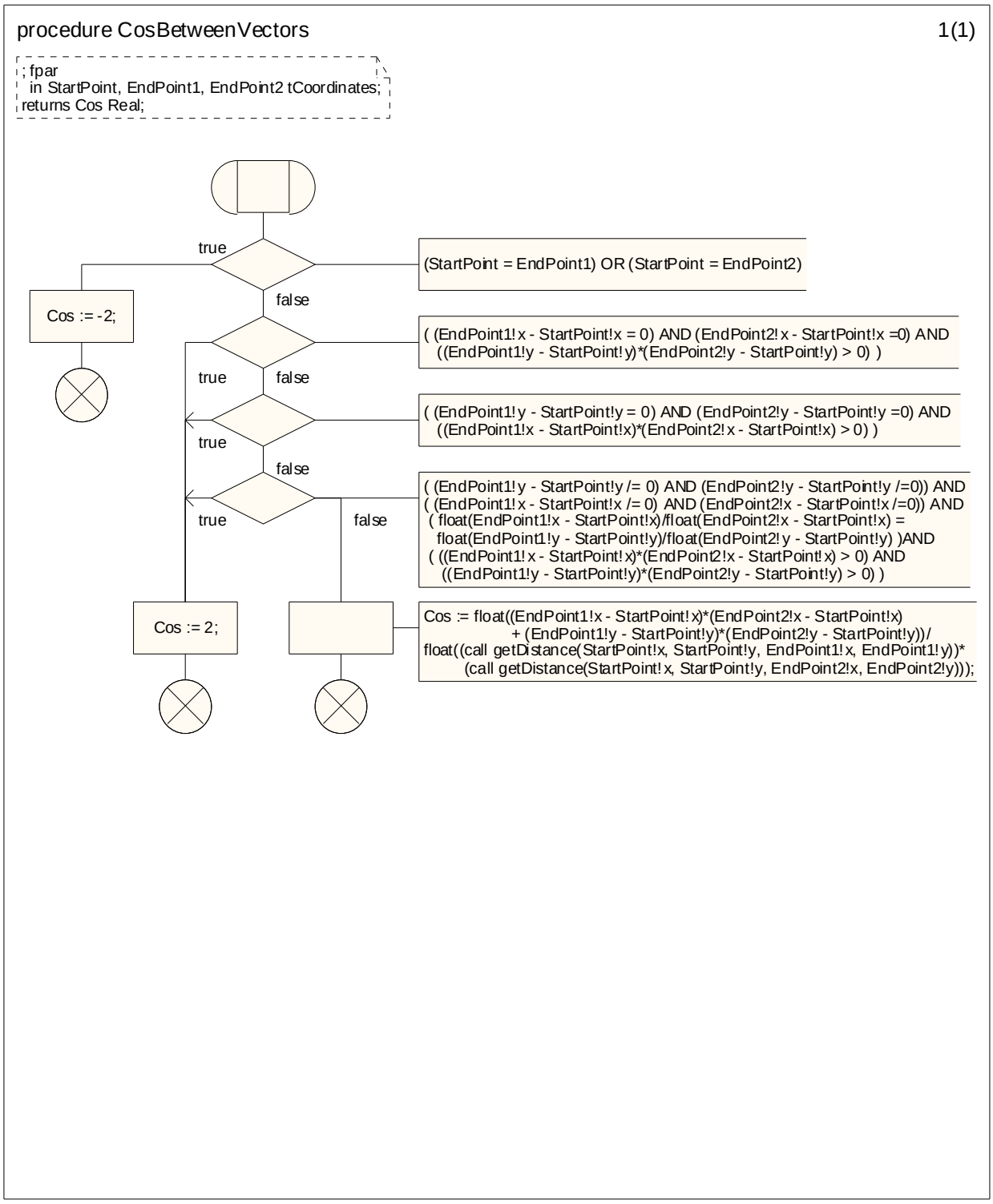

Рисунок 124. Процедура CosBetweenVectors.

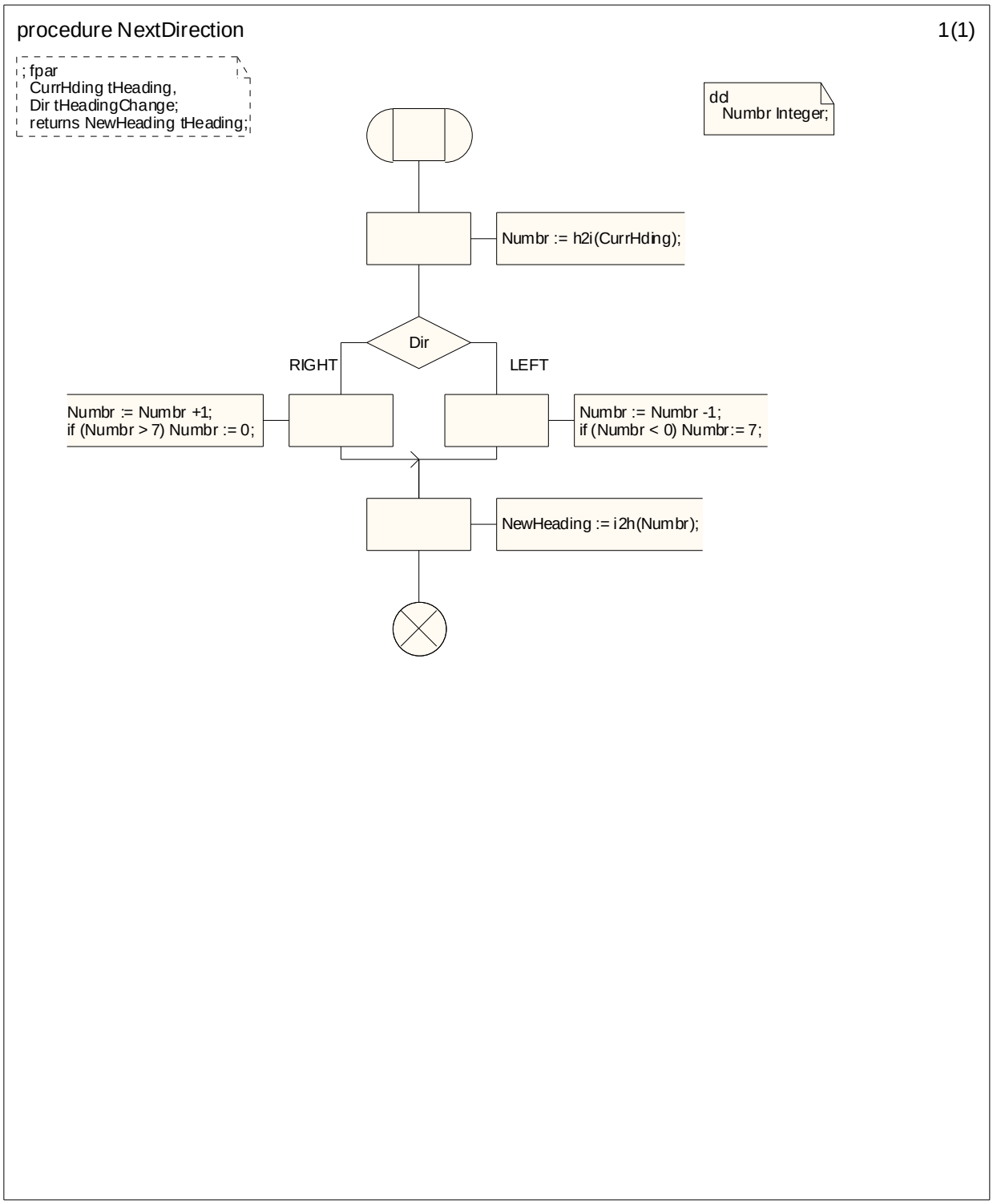

Рисунок 125. Процедура NextDirection.

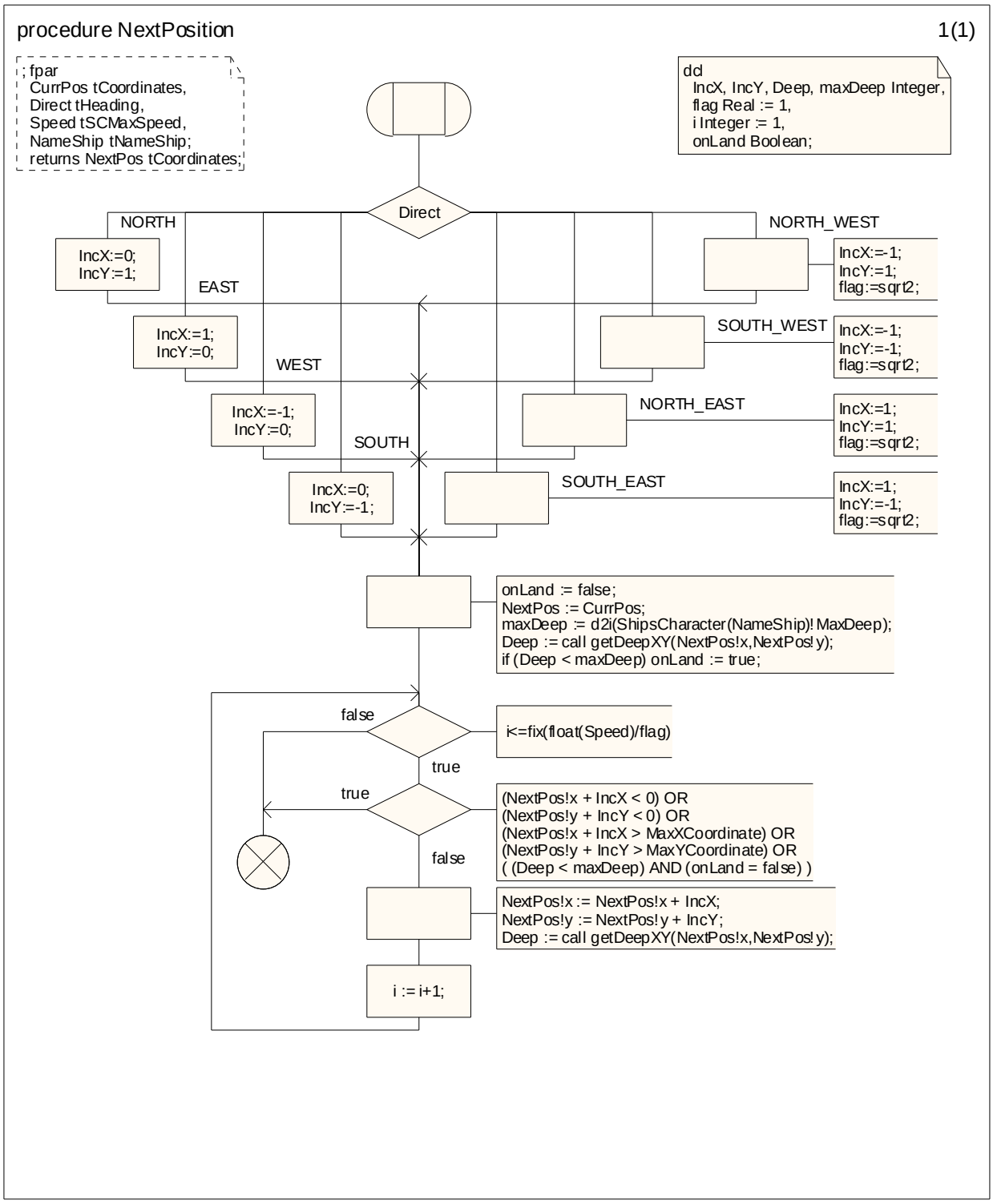

Рисунок 126. Процедура NextPosition.

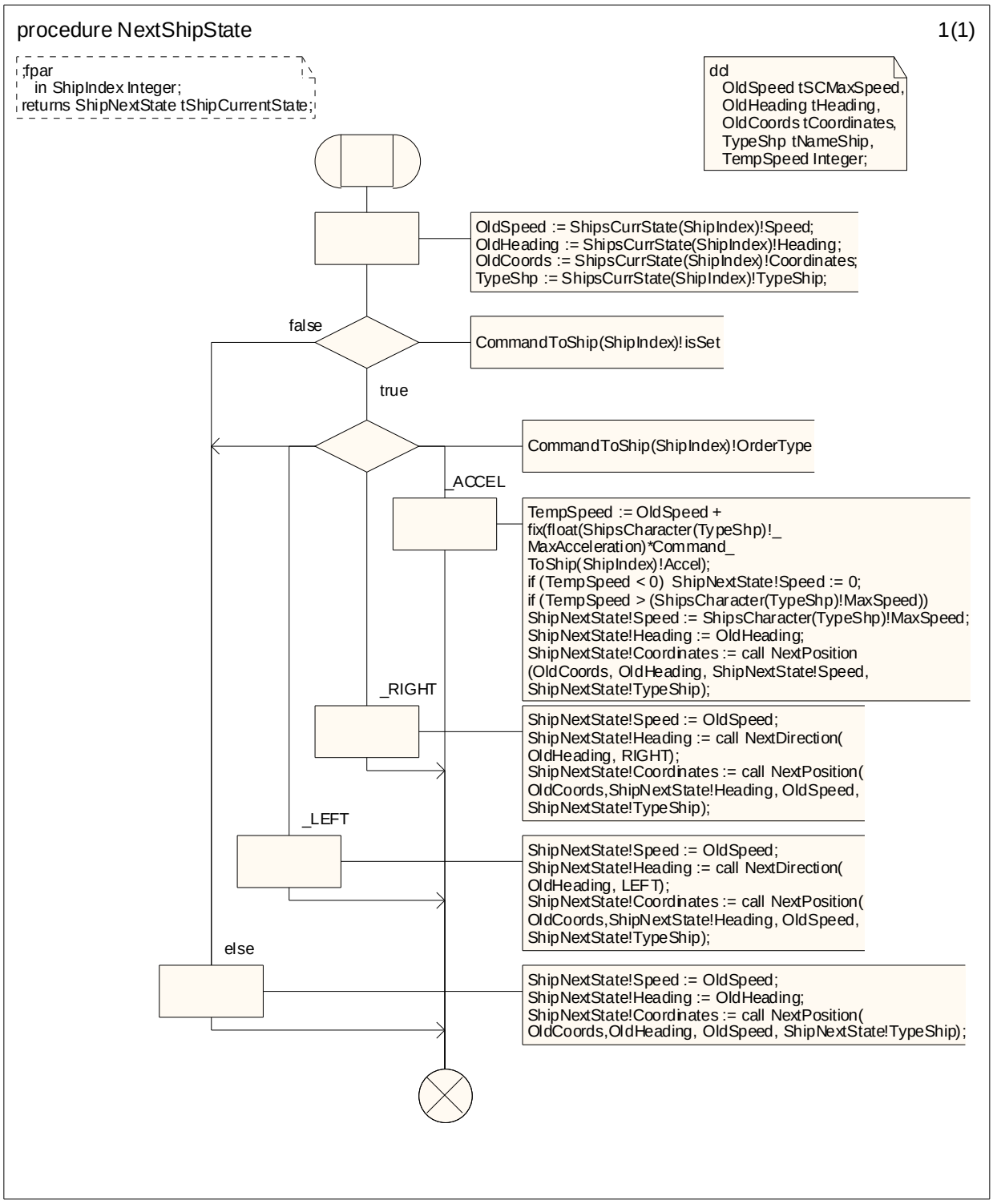

Рисунок 127. Процедура NextShipState.

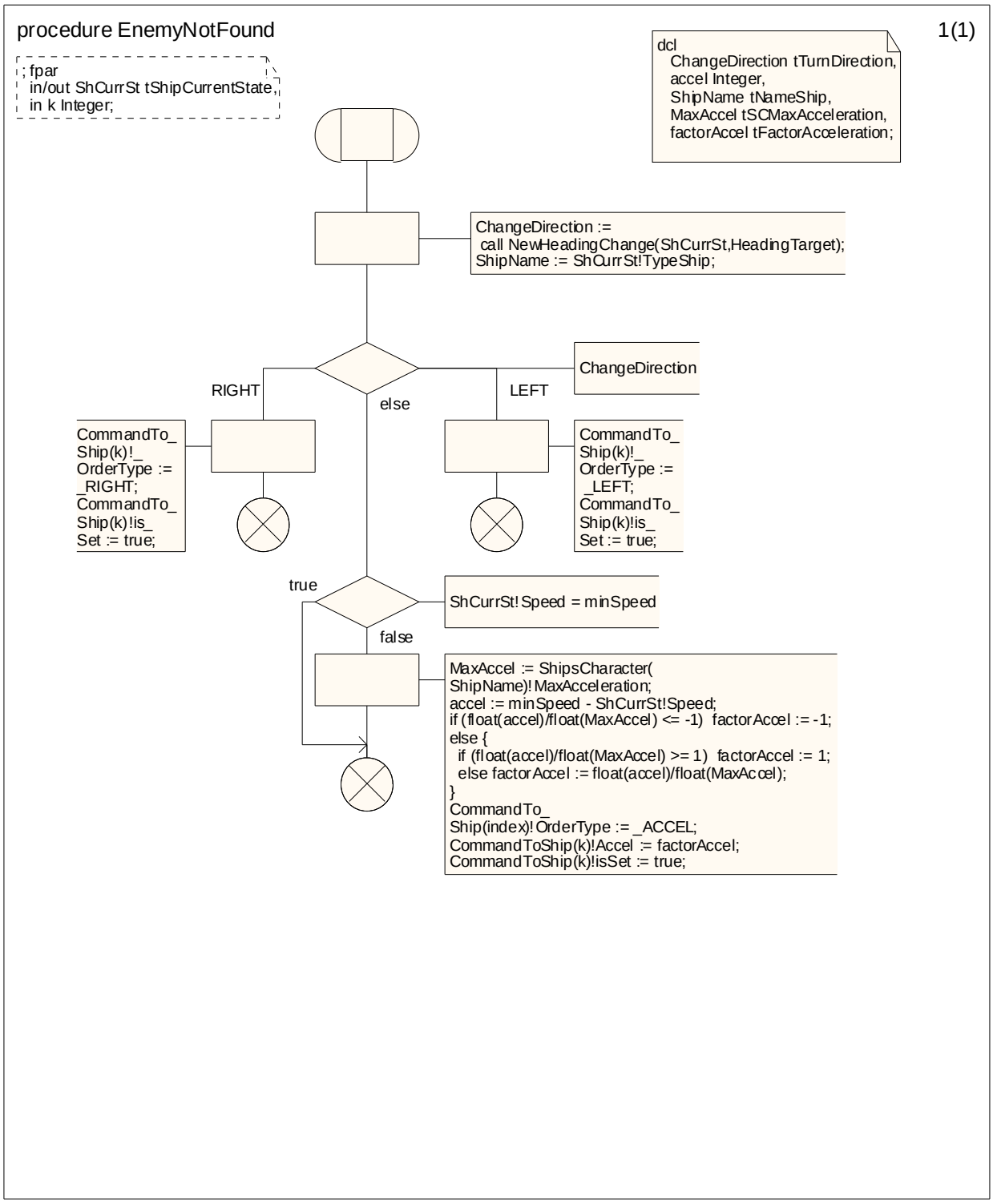

Рисунок 128. Процедура EnemyNotFound.

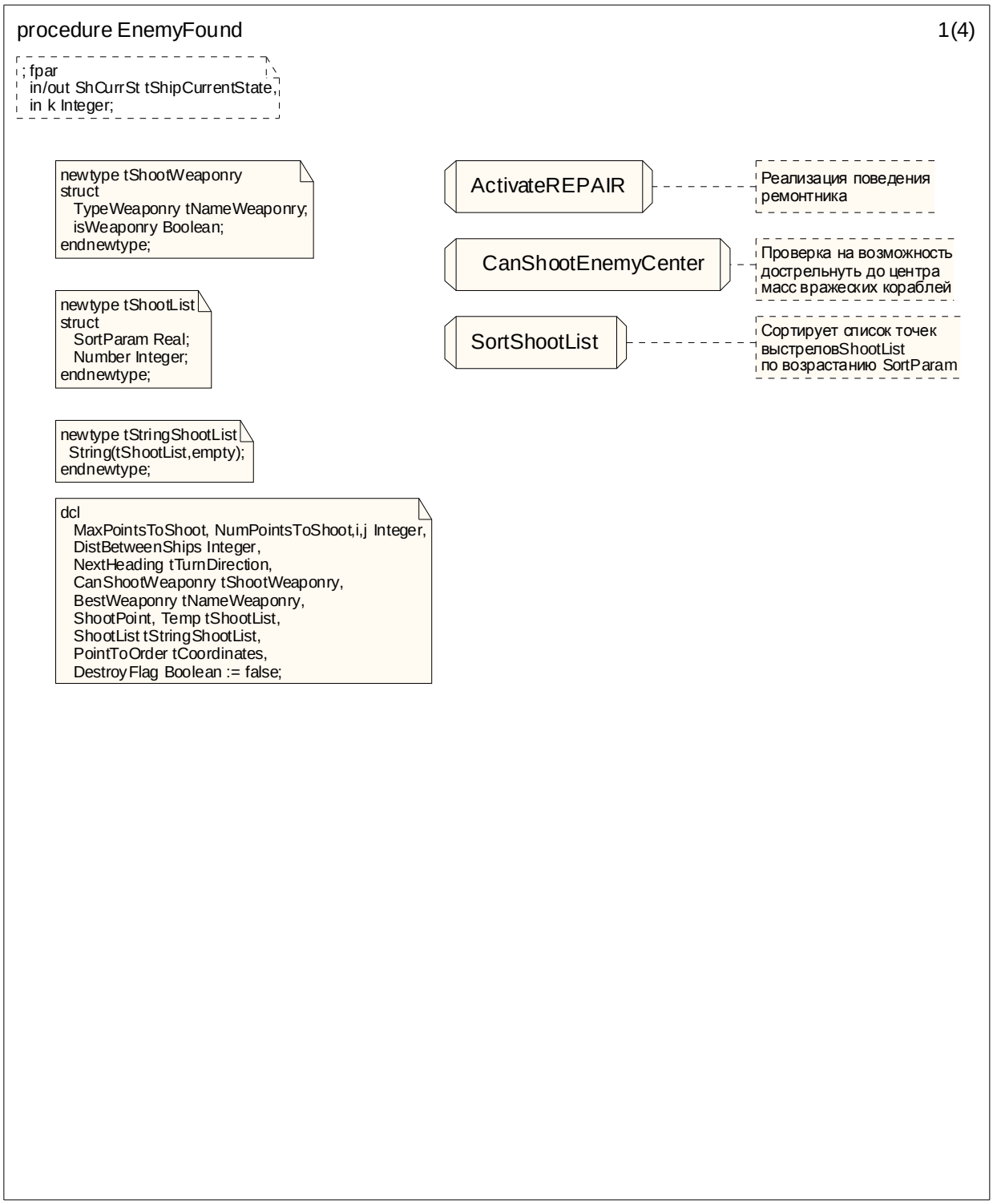

Рисунок 129. Процедура EnemyFound (1).

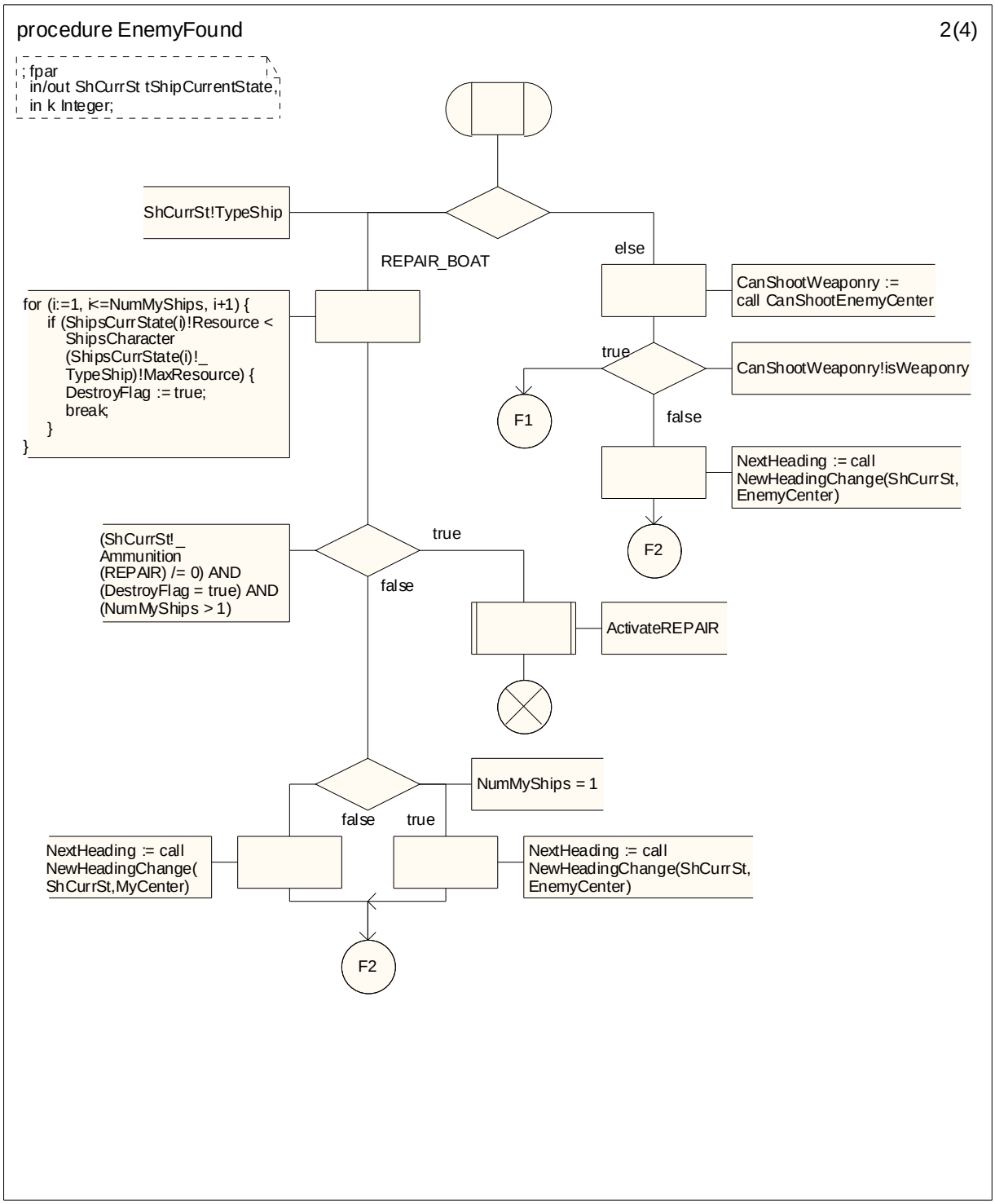

Рисунок 130. Процедура EnemyFound (2).

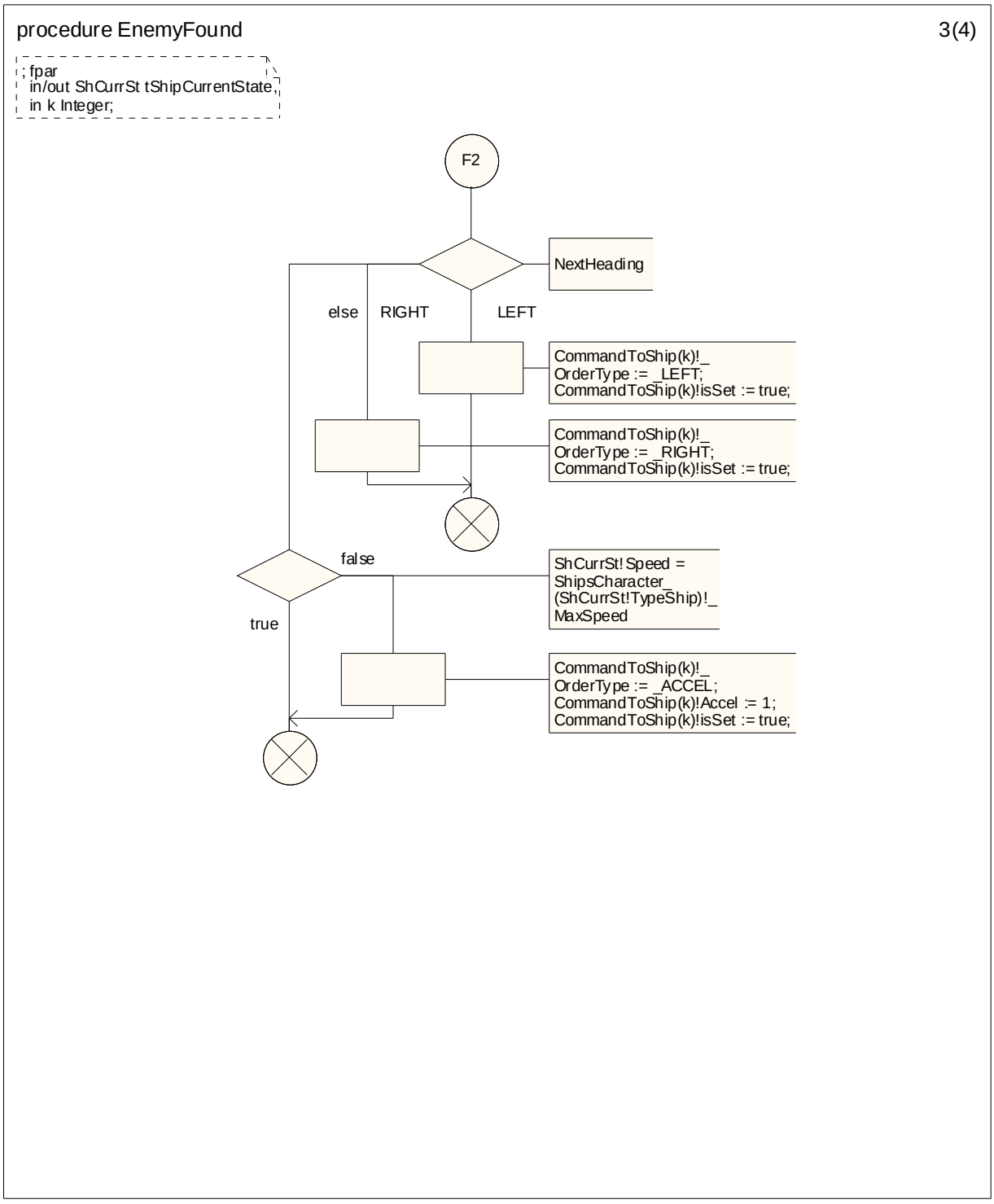

Рисунок 131. Процедура EnemyFound (3).

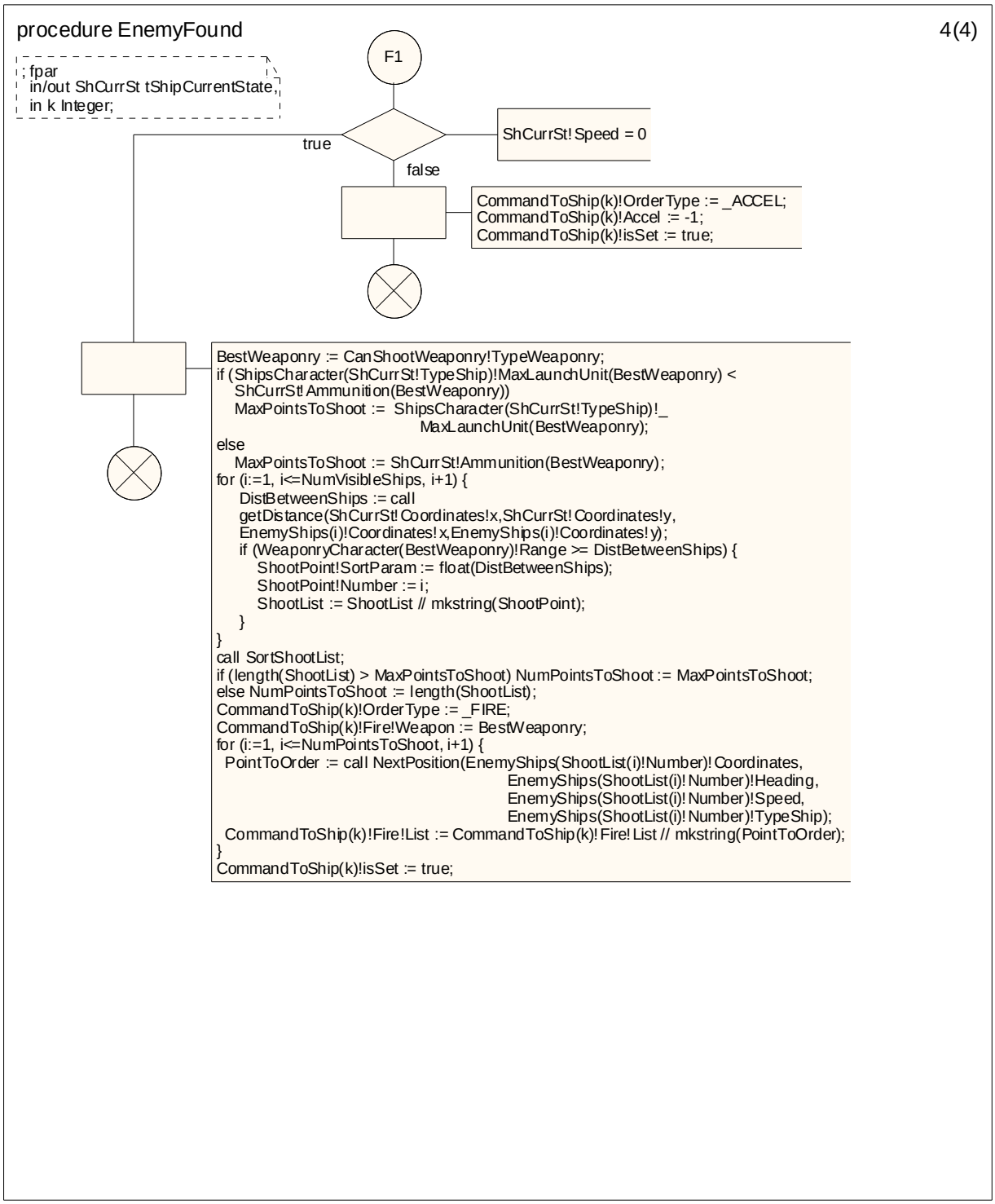

Рисунок 132. Процедура EnemyFound (4).

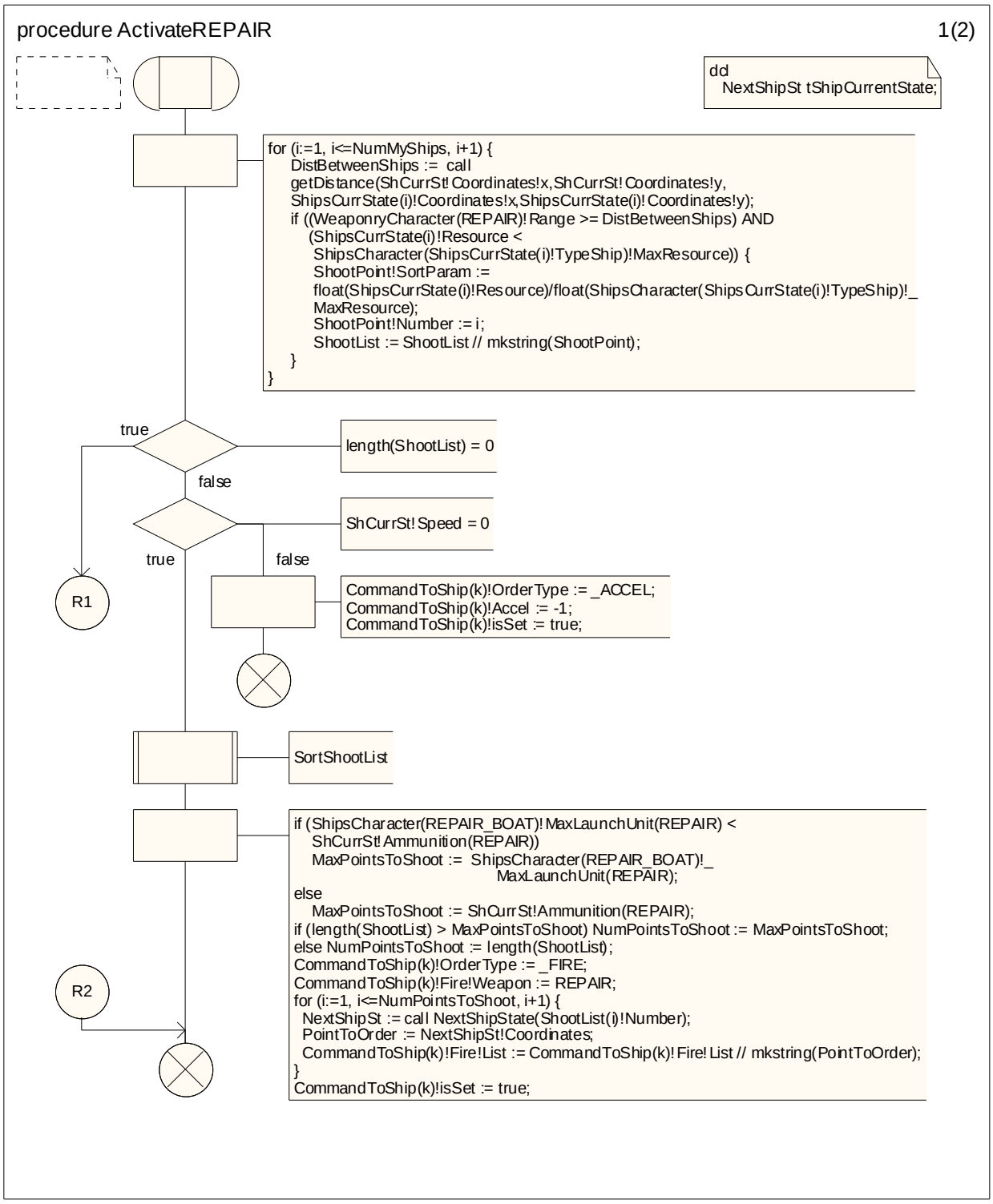

Рисунок 133. Процедура ActivateREPAIR (1).

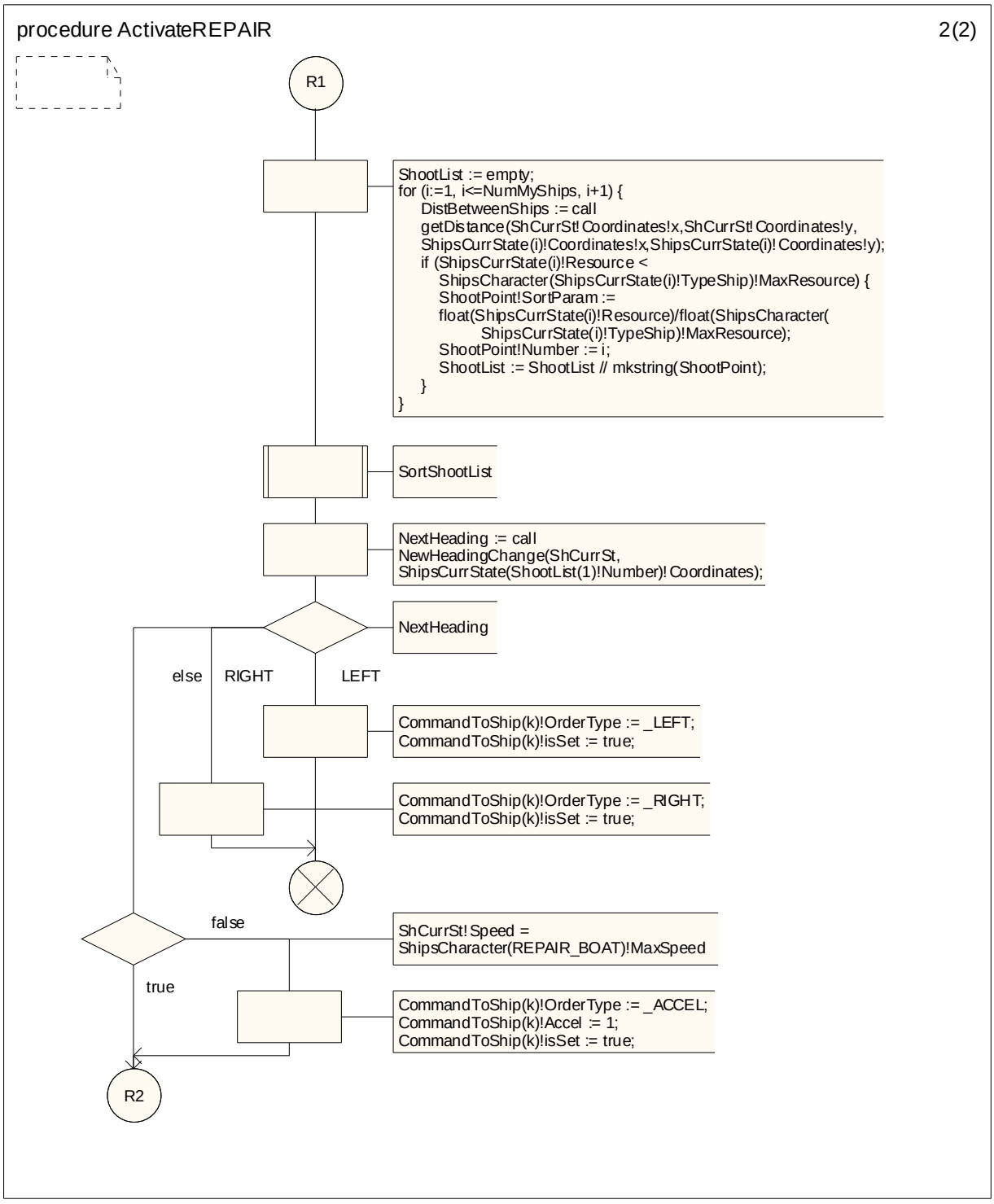

Рисунок 134. Процедура ActivateREPAIR (2).

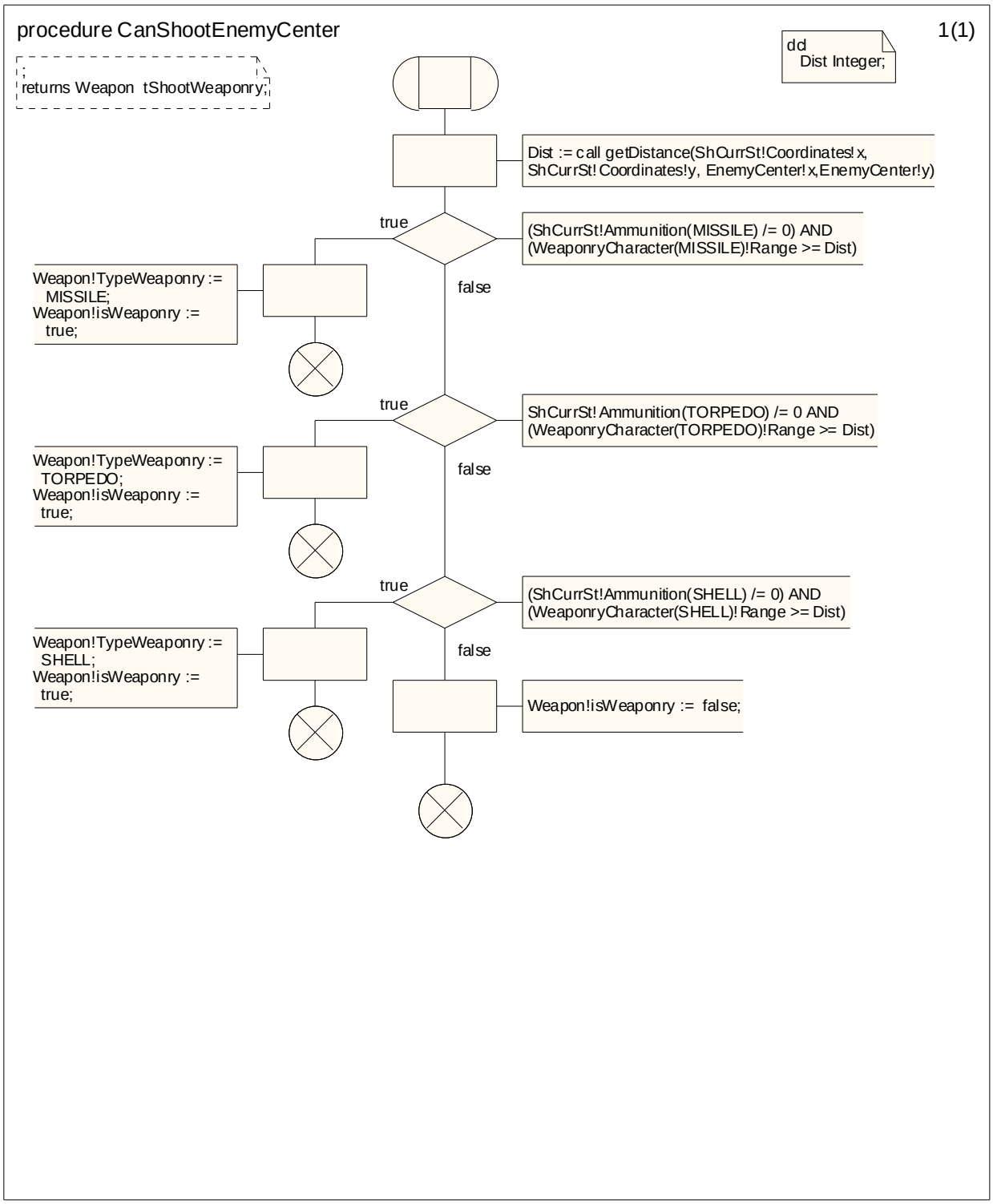

Рисунок 135. Процедура CanShootEnemyCenter.

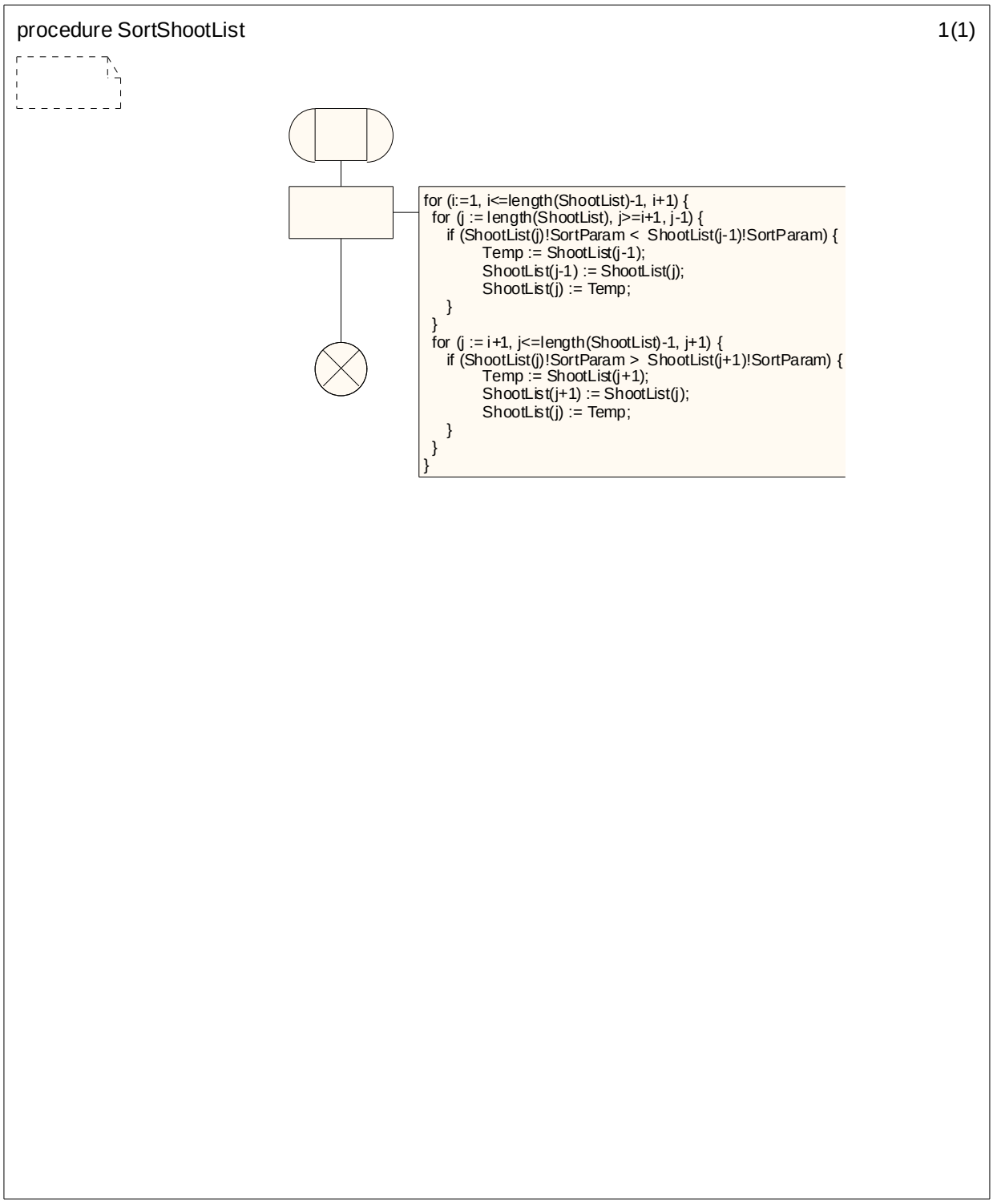

Рисунок 136. Процедура SortShootList.

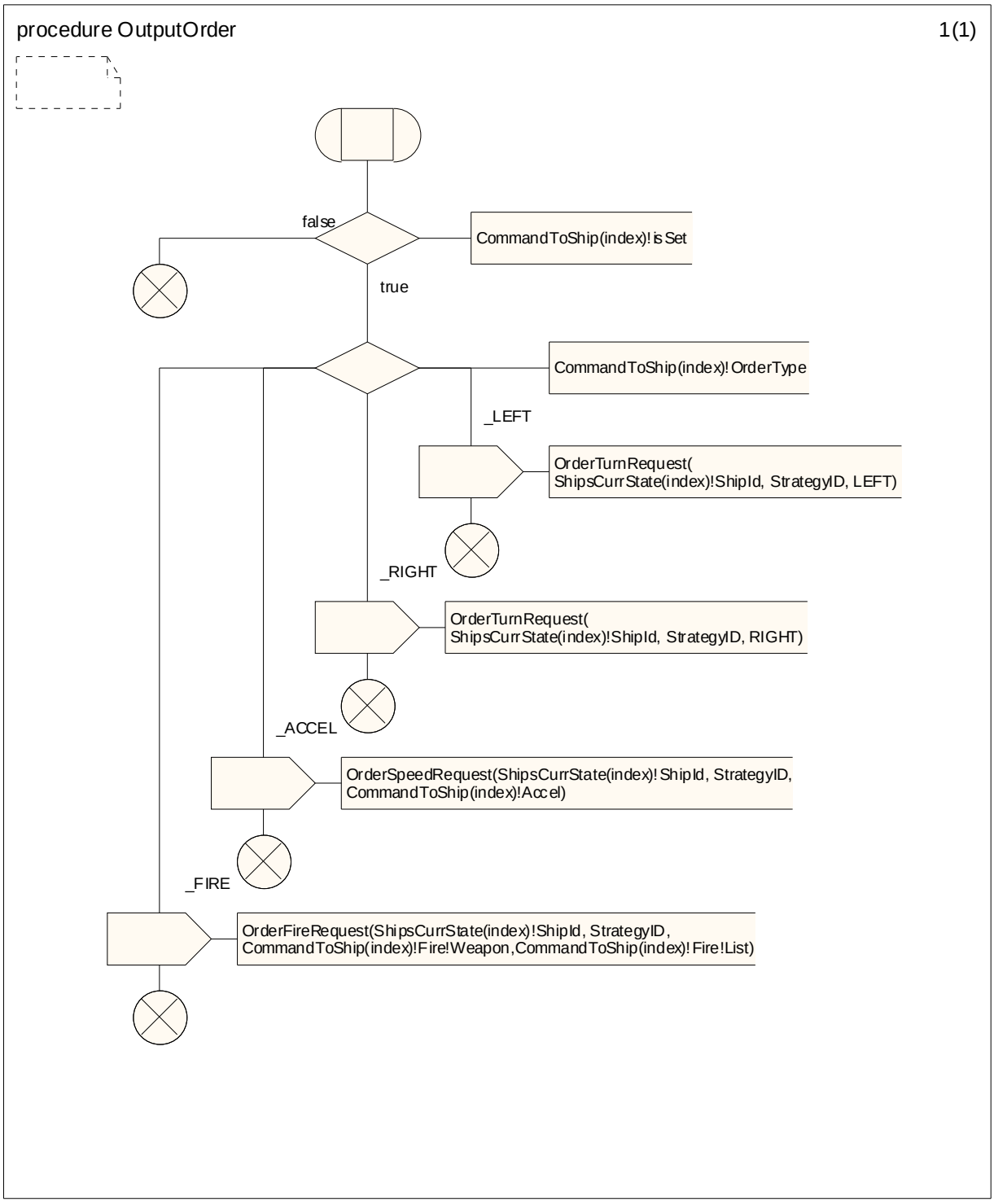

Рисунок 137. Процедура OutputOrder.

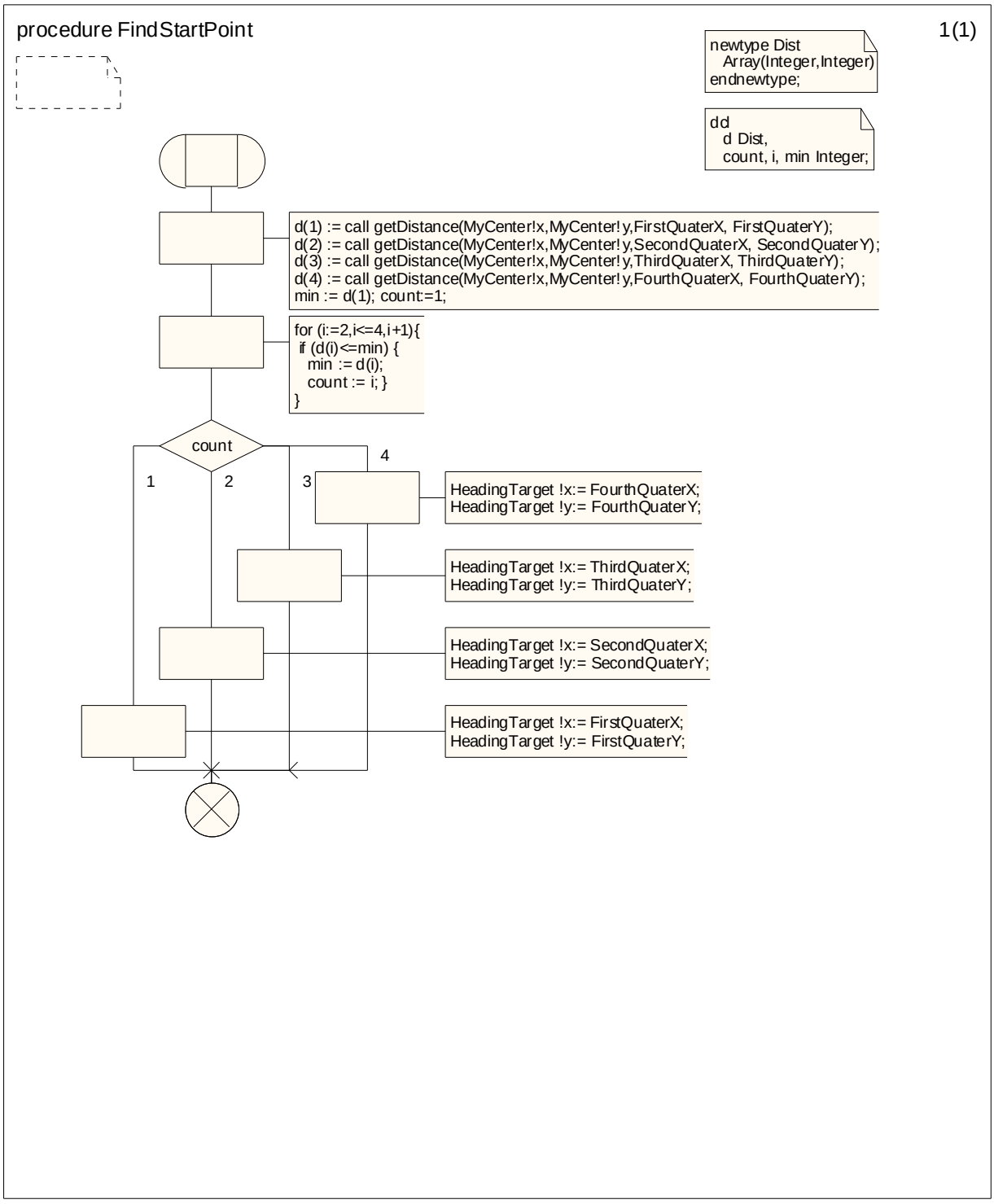

Рисунок 138. Процедура FindStartPoint.

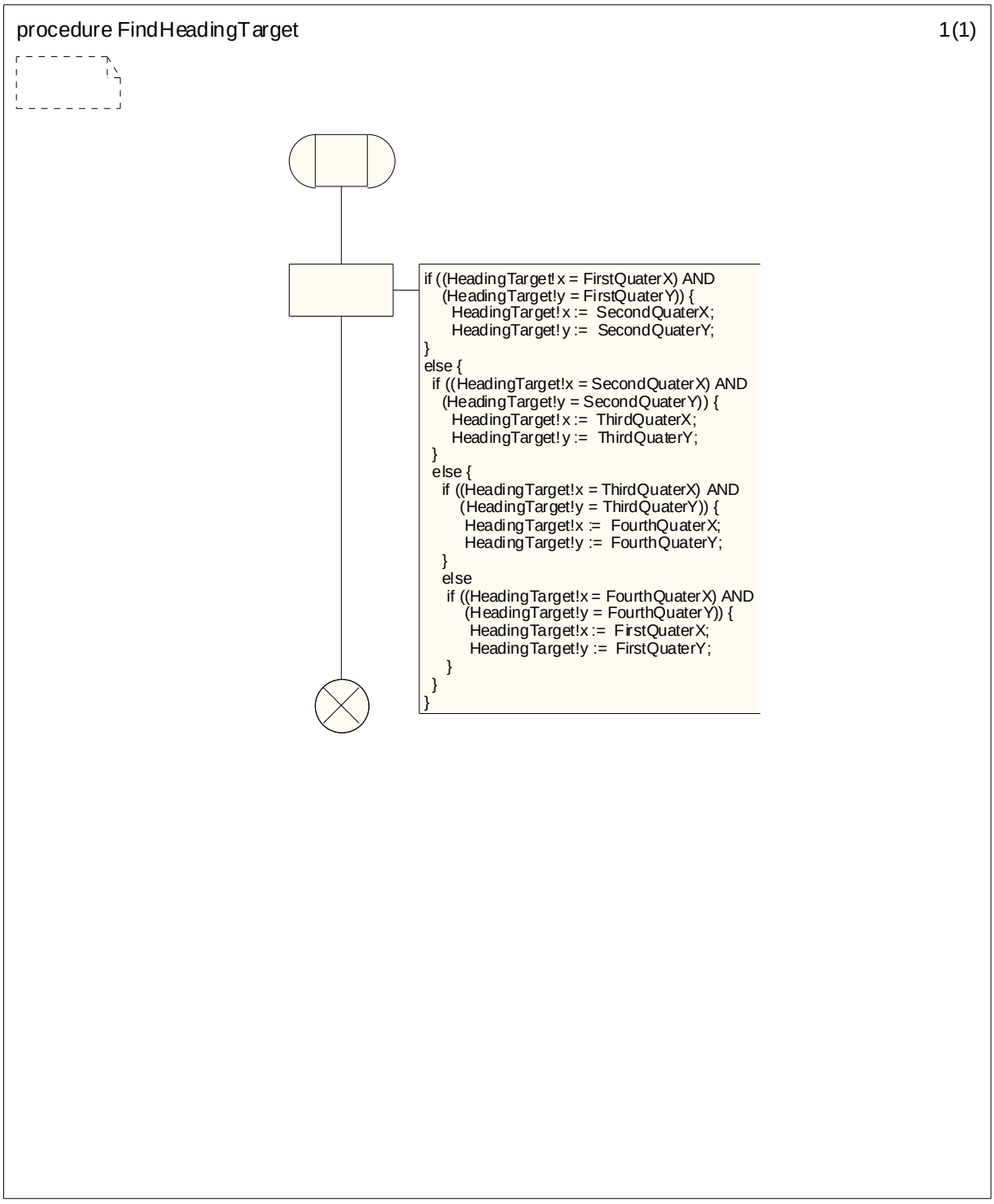

Рисунок 139. Процедура FindHeadingTarget.

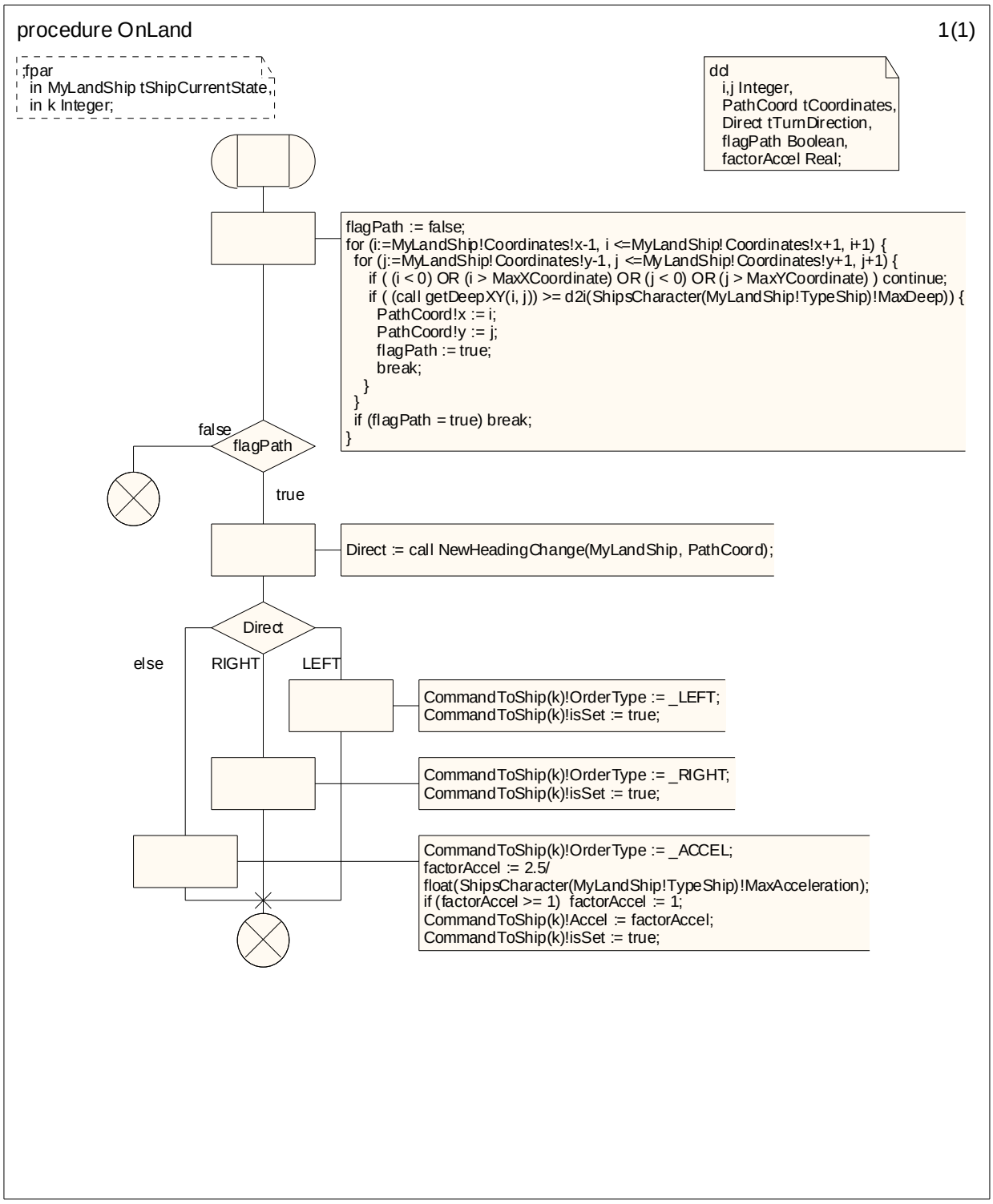

Рисунок 140. Процедура OnLand.

Приложение F -- Внешние функции.

```
#include <math.h>
#include <math.h>

#include <scttypes.h>
```

Рисунок 141. Описание типов данных, переменных и констант.

```
SDL_Integer getDistance(SDL_Integer xps,SDL_Integer yps,
                        SDL_Integer xpf,SDL_Integer ypf)
{
  return (SDL_Integer)floor(sqrt(pow((xpf - xps),2) + pow((ypf - yps),2)));
}
```

Рисунок 142. Функция getDistance.

```
SDL_Integer sdl_sqrt(SDL_Integer x)
{
  return (SDL_Integer)floor(sqrt(x));
}
```

Рисунок 143. Функция sdl_sqrt.

```
SDL_Integer getHeading(void)
{
    return (SDL_Integer)(rand()%8);
}
```

Рисунок 144. Функция getHeading.

Приложение G -- Код функций для работы с картой.

```
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <string.h>

#define MAX_FLEETS 8

typedef struct tagMap {
  int MaxX;
  int MaxY;
  char * MapVal;
} stMap;

static stMap * Map;
static const char * MapPath = "./../../Maps/map1.map";
static int FleetCoords [MAX_FLEETS][2];
static int MapReaded = 0;
static int CoordsCalculated = 0;
```

Рисунок 145. Описание типов данных, переменных и констант.

```
char * getPath()
{
  int StrSize;
  char * PathStr;

  StrSize = strlen(MapPath);
  PathStr = (char *)malloc((StrSize + 1) * sizeof(char));
  return strcpy(PathStr,MapPath);
}
```

Рисунок 146. Функция getPath.

```
int readMap()
{
  FILE * MapFile = NULL;
  int MapSize[2];

  Map = (stMap *)malloc(sizeof(stMap));
```

```
  if (Map == NULL)
    return 0;
  Map->MapVal = NULL;
  Map->MaxX = 0;
  Map->MaxY = 0;

  MapFile = fopen(MapPath,"r");
  if (MapFile == NULL)
    return 0;

  if (fread(MapSize,sizeof(int),2,MapFile) < 2)
    return 0;

  Map->MaxX = MapSize[0];
  Map->MaxY = MapSize[1];

  Map->MapVal = (char *)malloc(Map->MaxX * Map->MaxY * sizeof(char));
  if (fread(Map->MapVal,Map->MaxX * Map->MaxY * sizeof(char),1,MapFile) < 1)
    return 0;

  fclose(MapFile);
  MapReaded = 1;
  return 1;
}
```

Рисунок 147. Функция readMap.

```
int checkMap(void)
{
  if (!MapReaded)
    return readMap();
  return 1;
}
```

Рисунок 148. Функция checkMap.

```
int getDeepXY(int x, int y)
{
  if (!MapReaded)
    if (!readMap())
      return -1;

  if ((x > Map->MaxX)||(y > Map->MaxY)||(x <= 0)||(y <= 0))
```

```
    return -1;
  return Map->MapVal[Map->MaxX * (y - 1) + x - 1];
}
```

Рисунок 149. Функция getDeepXY.

```
int generatePoints(int Radius, int Deep, int NumOfStrat)
{
  int i = 0;
  int j = 0;
  int k = 0;
  int x = 0;
  int y = 0;
  int NumGeneratedFleets = 0;
  int Generated = 0;
  int BankFound = 0;
  int IncorrectCoords = 0;
  int Count = 0;
  float XCoef;
  float YCoef;

  if (!MapReaded)
    if (!readMap())
      return 0;

  if ((Deep > 4)||(Deep < 1)||(Radius < 0)||(NumOfStrat < 2))
    return 0;

  CoordsCalculated = 1;
  XCoef = (float)Map->MaxX/RAND_MAX;
  YCoef = (float)Map->MaxY/RAND_MAX;

  for (NumGeneratedFleets = 0; NumGeneratedFleets < NumOfStrat; NumGeneratedFleets++ )
  {
    Generated = 0;
    Count = 0;
    srand((unsigned)time(NULL));
    while (!Generated) {
      if (Count > 1000000)
        return 0;
      else
        Count++;
```

```
      BankFound = 0;
      IncorrectCoords = 0;
      x = (int)((float)XCoef * rand());
      y = (int)((float)YCoef * rand());

      for (k = 0;k < NumGeneratedFleets;k++)
        if((abs(x - FleetCoords[k][0]) <= (MAX_FLEETS - NumOfStrat + 2) * Radius )&&
           (abs(y - FleetCoords[k][1]) <= (MAX_FLEETS - NumOfStrat + 2) * Radius )) {
          IncorrectCoords = 1;
          break;
        }

      if (!IncorrectCoords) {
        if ((x - Radius >= 1)&&(x + Radius < Map->MaxX - 1)&&
            (y - Radius >= 1)&&(y + Radius < Map->MaxY - 1)) {
          for (j = y - Radius;j <= y + Radius;j++) {
            for (i = x - Radius;i <= x + Radius;i++)
              if (Map->MapVal[(j - 1) * Map->MaxX + i] < Deep) {
                BankFound = 1;
                break;
              }
            if (BankFound)
              break;
          }
          if (!BankFound) {
            FleetCoords[NumGeneratedFleets][0] = x;
            FleetCoords[NumGeneratedFleets][1] = y;
            Generated  = 1;
          }
        }
      }
    }
  }
  return 1;
}
```

Рисунок 150. Функция generatePoints.

```
int getFleetPointX(int StratNum)
{
  if ((StratNum < 0 )||(StratNum > MAX_FLEETS))
```

```
    return -1;
 if (!CoordsCalculated)
    return -1;
  return FleetCoords[StratNum - 1][0];
}
```

Рисунок 151. Функция getFleetPointX.

```
int getFleetPointY(int StratNum)
{
  if ((StratNum < 0 )||(StratNum > MAX_FLEETS))
    return -1;
  if (!CoordsCalculated)
    return -1;
  return FleetCoords[StratNum - 1][1];
}
```

Рисунок 152. Функция getFleetPointY.

```
void FreeAll()
{
    if (Map != NULL)
    {
        if (Map ->MapVal != NULL)
            free(Map->MapVal);
        free(Map);
    }
}
```

Рисунок 153. Функция FreeAll.

Приложение H -- Код функций для работы с GUI.

```
#ifndef _ZSOCKETDATA_H_
#define _ZSOCKETDATA_H_

//Windows implementation
#include <winsock2.h>

typedef SOCKET tSockID;

class ZSocketData {
public:
  ZSocketData();
  virtual ~ZSocketData();

//Platform depended static data
public:
  static bool EnvReady;
  static int RefCounter;

//Platform depended data

  SOCKET GetSockID();
  void SetSockID(tSockID sSockID);

protected:
  tSockID SockID;
};

#endif
```

Рисунок 154. Заголовочный файл «ZSocketData.h».

```
#ifndef _ZSOCKET_H_
#define _ZSOCKET_H_

#include "ZSocketData.h"
#include <string>
using namespace std;

class ZSocket : public ZSocketData {
//Exceptions
```

```
private:
  class WSAFailed {};
  class CreateFailed {};
  class ConnectionFailed {};
  class ReceiveFailed {};
  class SendFailed {};
  class WrongIPString {};
  class WaitingFailed {};

//Interface
public:
  ZSocket();
  virtual ~ZSocket();

  void ConnectToDomain(const std::string strInetAddr,unsigned short int Port);
  void ConnectToIP(const std::string strIPAddr,unsigned short int Port);
  int Pop(char *pBuffer, unsigned int MaxSize);
  void Push(const char *pCHBuffer, int PushSize);
  void Push(u_short *pSIBuffer, int PushSize);
  void Push(u_long *pIBuffer, int PushSize);

//Additional functions
protected:
  sockaddr_in GetSockAddr(const std::string strIPAddr,unsigned short int Port);
  sockaddr_in GetSockAddr(unsigned short int Port);

//Inner implementation
private:
  void Init();
  void Destroy();
  void Create();
};

#endif
```

Рисунок 155. Заголовочный файл «ZSocket.h».

```
#ifndef _ZSSOCKET_H_
#define _ZSSOCKET_H_

#include "ZSocket.h"
#include "ZCSocket.h"
```

```
class ZSSocket: public ZSocket {
public:
  ZSSocket() {};
  ~ZSSocket() {};

  void BindToPort(unsigned short int Port);
  void WaitForCall(struct sockaddr *CallerAddress);
  void SetClient(ZCSocket *pCSocket);

  int PopUntil(char* pBuffer, int BufSize);

protected:
  ZSocket toClientSocket;
};

#endif
```

Рисунок 156. Заголовочный файл «ZSSocket.h».

```
#include "ZSocketData.h"

//========================================================================
//Static members definitions ==============================================

bool ZSocketData::EnvReady=false;
int ZSocketData::RefCounter=0;

//End of Static members definitions
//========================================================================

ZSocketData::ZSocketData()
{
}

ZSocketData::~ZSocketData()
{
}

SOCKET ZSocketData::GetSockID()
{
  return SockID;
```

```
}
void ZSocketData::SetSockID(SOCKET sSockID)
{
  SockID=sSockID;
}
```

Рисунок 157. Файл «ZSocketData.cpp».

```
#include "ZSocket.h"
#include "winsock2.h"

//====================================================================
//=Constructor========================================================
ZSocket::ZSocket()
{
  Init();
  Create();
}
//=End of Constructor=================================================
//====================================================================

//====================================================================
//=Destructor=========================================================
ZSocket::~ZSocket()
{
  Destroy();
}
//=End of Destructor==================================================
//====================================================================

//============================================================================
//==Init======================================================================
void ZSocket::Init()
{
  if (!EnvReady)    {
    WSADATA wsaData;
    if (WSAStartup(MAKEWORD(1,1),&wsaData)) {
      //throw WSAFailed();
      fprintf(stderr,"WinSock DLL initialization failed.");
    }
    EnvReady=true;
  }
```

```
  RefCounter++;
}
//==End of Init=================================================================
//==============================================================================

//==============================================================================
//==Create======================================================================
void ZSocket::Create()
{
  SockID=socket(AF_INET,SOCK_STREAM,0);
  if (SockID==INVALID_SOCKET) {
    //throw CreateFailed();
    fprintf(stderr,"Socket initialization failed.\n");
  }
}
//==End of Create===============================================================
//==============================================================================

//==============================================================
//==Destroy=====================================================
void ZSocket::Destroy()
{
  closesocket(SockID);
  RefCounter--;
  if (RefCounter==0)    {
    WSACleanup();
  }
}
//=End of Destroy==============================================
//==============================================================

//==============================================================================
//==ConnectToDomain=============================================================
void ZSocket::ConnectToDomain(const std::string strInetAddr,unsigned short int Port)
{
  sockaddr_in cs_addr;
  cs_addr.sin_family=AF_INET;
  //cs_addr.sin_addr.S_un.S_addr=inet_addr(strIPAddr.c_str());
  cs_addr.sin_port=htons(Port);

  if (SOCKET_ERROR==connect(SockID,(sockaddr*)&cs_addr,sizeof(cs_addr))) {
    //throw ConnectionFailed();
```

```
    fprintf(stderr,"Socket's connecting failed.");
  }
}
//==End of ConnectToDomain=====================================================
//=============================================================================

//=============================================================================
//==ConnectToIP================================================================
void ZSocket::ConnectToIP(const string strIPAddr,unsigned short int Port)
{
  sockaddr_in cs_addr;
  cs_addr.sin_family=AF_INET;
  cs_addr.sin_addr.S_un.S_addr=inet_addr(strIPAddr.c_str());
  cs_addr.sin_port=htons(Port);

  if (SOCKET_ERROR==connect(SockID,(sockaddr*)&cs_addr,sizeof(cs_addr))) {
    //throw ConnectionFailed();
    fprintf(stderr,"Socket's connecting failed.\n");
  }
}
//==End of ConnectToIP=========================================================
//=============================================================================

//=============================================================================
//==GetSockAddr================================================================
sockaddr_in ZSocket::GetSockAddr(const std::string strIPAddr,unsigned short int Port)
{
  sockaddr_in res_addr;
  res_addr.sin_family=AF_INET;
  if (res_addr.sin_addr.S_un.S_addr=inet_addr(strIPAddr.c_str())==-1) {
    //throw WrongIPString();
    fprintf(stderr,"Wrong IP address string: %s\n",strIPAddr.c_str());
  }
  res_addr.sin_port=htons(Port);
  return res_addr;
}
//==End of GetSockAddr=========================================================
//=============================================================================

//=============================================================================
//==GetSockAddr================================================================
sockaddr_in ZSocket::GetSockAddr(unsigned short int Port)
```

```
{
  sockaddr_in res_addr;
  memset(&res_addr,0,sizeof(res_addr));
  res_addr.sin_family=AF_INET;
  res_addr.sin_addr.S_un.S_addr=htonl(INADDR_ANY);
  res_addr.sin_port=htons(Port);
  return res_addr;
}
//==End of GetSockAddr============================================================
//================================================================================

//================================================================================
//==Pop===========================================================================
int ZSocket::Pop(char* pBuffer, unsigned int MaxSize)
{
  int len;
  len=recv(SockID,pBuffer,MaxSize,0);
  if (len==SOCKET_ERROR) {
    //throw ReceiveFailed();
    fprintf(stderr,"Receiving data error.\n");
  }
  return len;
}
//==End of Pop====================================================================
//================================================================================

//================================================================================
//==Push==========================================================================
void ZSocket::Push(const char* pCHBuffer, int PushSize)
{
  int res;
  res=send(SockID,pCHBuffer,PushSize,0);
  if (res==SOCKET_ERROR) {
    //throw SendFailed();
    fprintf(stderr,"Sending data error.\n");
  }
}
//==End of Push===================================================================
//================================================================================

//================================================================================
//==Push==========================================================================
```

```
void ZSocket::Push(u_long* pIBuffer, int PushSize)
{
  int i;
  union {
    u_long ulValue;
    char chBuf[4];
  } transULNS;

  for (i=0; i<PushSize; i++) {
    transULNS.ulValue=htonl(pIBuffer[i]);
    Push(transULNS.chBuf,4);
  }
}
//==End of Push=====================================================================
//==================================================================================

//==================================================================================
//==Push============================================================================
void ZSocket::Push(u_short *pSIBuffer, int PushSize)
{
  int i;
  union {
    u_short usValue;
    char chBuf[2];
  } transUSNS;

  for (i=0; i<PushSize; i++) {
    transUSNS.usValue=htons(pSIBuffer[i]);
    Push(transUSNS.chBuf,2);
  }
}
//==End of Push=====================================================================
//==================================================================================
```

Рисунок 158. Файл «ZSocket.cpp».

```
#include "ZSSocket.h"

//==================================================================================
//=BindToIP========================================================================
void ZSSocket::BindToPort(unsigned short int Port)
{
```

```
  sockaddr_in cs_addr = GetSockAddr(Port);

  if (SOCKET_ERROR==bind(SockID,(sockaddr*)&cs_addr,sizeof(cs_addr))) {
    //throw ConnectionFailed();
    fprintf(stderr,"Socket's binding failed.\n");
  }
  if (SOCKET_ERROR==listen(SockID,5)) {
    //throw ListenFailed();
    fprintf(stderr,"(%i) Socket listen failed.\n",::WSAGetLastError());
  }
}
//=End of BindToIP=============================================================
//=============================================================================

//=============================================================================
//==WaitForCall================================================================
void ZSSocket::WaitForCall(struct sockaddr *CallerAddress)
{
  int *CASize = 0;

  SOCKET res_acc = accept(SockID,CallerAddress,CASize);
  if ( res_acc == INVALID_SOCKET) {
    //throw WaitingFailed;
    fprintf(stderr,"Socket accepting failed\n");
  } else    {
    toClientSocket.SetSockID(res_acc);
  }
}
//==End of WaitForCall=========================================================
//=============================================================================

//=============================================================================
//==PopUntil===================================================================
int ZSSocket::PopUntil(char* pBuffer, int BufSize)
{
  int res,tmpSize;
  tmpSize=0;
  while (tmpSize<BufSize) {
    res = recv(toClientSocket.GetSockID(),pBuffer,BufSize,0);
    if (res == 0) {
      //Connection closed
      return 0;
```

```
    }
    if (res == SOCKET_ERROR) {
      //throw ReceiveFailed();
      fprintf(stderr,"(ERR:%i) Receiving data error.\n",WSAGetLastError());
      return -1;
    }
    tmpSize += res;
    pBuffer += res;
  }
  return tmpSize;
}
//==End of PopUntil=============================================================
//==============================================================================

//==============================================================================
//==============================================================================
void ZSSocket::SetClient(ZCSocket *pCSocket)
{
  pCSocket->SetSockID(toClientSocket.GetSockID());
}
//==============================================================================
//==============================================================================
```

Рисунок 159. Файл «ZSSocket.cpp».

```
#ifndef _ZSWRAP_H_
#define _ZSWRAP_H_

#include <stdio.h>

const size_t gc_BufferSize = 100;
const unsigned short int GUIPort = 22881;
const int EC_OK = 0;
const int EC_FAULT = -1;

extern "C" int InitConnection(void);
extern "C" void PushParam(int pushParam);
extern "C" void PushMap(void);
extern "C" void PushFixedShipPart(
             int ShipID,
             int StrategyID,
             int TypeShip,
```

```
            int Speed,
            int Heading,
            int Resource,
            int CoordinatesX,
            int CoordinatesY,
            int MissleAmount,
            int TorpedoAmount,
            int ShellAmount,
            int RepairAmount);
extern "C" void PushVariableShipPart(
            int Visible);
extern "C" void PushShot(
            int TypeWeaponry,
            int CurrentCoordinatesX,
            int CurrentCoordinatesY,
            int TargetCoordinatesX,
            int TargetCoordinatesY);
extern "C" int SendData(void);
extern "C" void PushShipsDummy(void);
extern "C" void PushShotsDummy(void);
#endif
```

Рисунок 160. Заголовочный файл «ZSWrap.h».

```
#include "../ZSocket/ZSSocket.h"
#include "../ZSocket/ZCSocket.h"

#include "ZSWrap.h"
#include <list>

void PushHeader(char ID, int LEN);
void FlushIntBuffer(void);
void FlushShipBuffer(void);

//================================================================================
//=Ship depended part============================================================
typedef struct tagShip {
  u_long fixPart[12];
  std::list<char> varPart;
} StructShip;

std::list<StructShip> shipBuffer;
```

```
StructShip *curShip;
//=End of Ship depended part=============================================================
//=======================================================================================

//=======================================================================================
//=Global Variables======================================================================
enum {
  NonConnected,
  Authorizing,
  Idle,
  PushingParams,
  PushingFixedShipPart,
  PushingVariableShipPart,
  PushingShots
} State = NonConnected;

ZSSocket gSSock;
ZCSocket gCSock;
std::list<char> charBuffer;
std::list<u_long> intBuffer;
int g_NumElements;
int g_ByteSize;
//=End of Global Variables===============================================================
//=======================================================================================

//=======================================================================================
//=Interface Part========================================================================
//=======================================================================================

//=======================================================================================
//=InitConnection========================================================================
int InitConnection()
{
  if (State==NonConnected) {
    char buffer[6];
    memset(buffer,0,6);
    sockaddr_in ClAddr;

    bool Authorized=false;
    gSSock.BindToPort(GUIPort);
//    while (!Authorized) {
    gSSock.WaitForCall((sockaddr*)&ClAddr);
```

```
    gSSock.PopUntil(buffer,5);
    if (!strcmp(buffer,"Hello")) {
      gSSock.SetClient(&gCSock);
      Authorized=true;
      PushHeader(10,0);
    } else {
      return EC_FAULT;
    }
//    }

    State = Idle;
  } else {
    return EC_FAULT; // Already connected
  }
  return EC_OK;
}
//=============================================================================
//=============================================================================

//=============================================================================
//=PushParam===================================================================
void PushParam(int pushParam)
{
  if (State==Idle) {
    State=PushingParams;
    charBuffer.clear();
    intBuffer.clear();
    g_NumElements=0;
  }
  if (State==PushingParams) {
    intBuffer.push_back((u_long)pushParam);
    g_NumElements++;
  }
}
//=End of PushParam============================================================
//=============================================================================

//=============================================================================
//=void PushMap(void)==========================================================
extern "C" char* getPath(void);

void PushMap(void)
```

```
{
  char *MapPath = getPath();
  if (State == Idle) {
    union {
      char chBuf[4];
      u_long ulValue;
    } MaxX,MaxY;

    FILE *MapFile;
    char Buffer[gc_BufferSize];
    size_t tmpReaded = 0;

    MapFile = fopen(MapPath,"r");
    if (!MapFile) {
      fprintf(stderr,"Couldn't open file %s.",MapPath);
      return;
    }

    if (!feof(MapFile) && !ferror(MapFile)) {
      tmpReaded = fread(MaxX.chBuf,sizeof(char),4,MapFile);
    }
    if (!feof(MapFile) && !ferror(MapFile)) {
      tmpReaded = fread(MaxY.chBuf,sizeof(char),4,MapFile);
    }
    //Start pushing
    PushHeader(1,(2*4+MaxX.ulValue*MaxY.ulValue));
    gCSock.Push(&MaxX.ulValue,1);
    gCSock.Push(&MaxY.ulValue,1);
    while (!feof(MapFile)) {
      tmpReaded = fread(Buffer,sizeof(char),100,MapFile);
      if (ferror(MapFile)) {
        break;
      }
      gCSock.Push(Buffer,(int)tmpReaded);
    }
    fclose(MapFile);
  }
}
//=End of PushMap=================================================================
//================================================================================

//================================================================================
```

```
//=PushFixedShipPart============================================================
void PushFixedShipPart(
        int ShipID,
        int StrategyID,
        int TypeShip,
        int Speed,
        int Heading,
        int Resource,
        int CoordinatesX,
        int CoordinatesY,
        int MissleAmount,
        int TorpedoAmount,
        int ShellAmount,
        int RepairAmount
        )
{
  if (State == Idle) {
    State = PushingFixedShipPart;
    charBuffer.clear();
    intBuffer.clear();

    shipBuffer.clear();
    g_NumElements=0;
    g_ByteSize=0;
  }
  if (State == PushingVariableShipPart) {
    State = PushingFixedShipPart;
  }
  if (State == PushingFixedShipPart) {
    g_NumElements++;
    g_ByteSize+=12*4;
    StructShip ss;
    ss.fixPart[0]=(u_long)ShipID;
    ss.fixPart[1]=(u_long)StrategyID;
    ss.fixPart[2]=(u_long)TypeShip;
    ss.fixPart[3]=(u_long)Speed;
    ss.fixPart[4]=(u_long)Heading;
    ss.fixPart[5]=(u_long)Resource;
    ss.fixPart[6]=(u_long)CoordinatesX;
    ss.fixPart[7]=(u_long)CoordinatesY;
    ss.fixPart[8]=(u_long)MissleAmount;
    ss.fixPart[9]=(u_long)TorpedoAmount;
```

```
    ss.fixPart[10]=(u_long)ShellAmount;
    ss.fixPart[11]=(u_long)RepairAmount;
    ss.varPart.clear();
    shipBuffer.push_back(ss);
    curShip=&shipBuffer.back();
  }
}
//=End of PushFixedShipPart=============================================================
//======================================================================================

//======================================================================================
//=PushVariableShipPart=================================================================
void PushVariableShipPart(int Visible)
{
  if (State == PushingFixedShipPart) {
    State = PushingVariableShipPart;
  }
  if (State == PushingVariableShipPart) {
    (*curShip).varPart.push_back( char(Visible) );
    g_ByteSize++;
  }
}
//=End of PushVariableShipPart==========================================================
//======================================================================================

//======================================================================================
//=PushShipsDummy=======================================================================
void PushShipsDummy(void)
{
  if (State == Idle) {
    PushHeader(2,4);
    u_long tmpULong = 0;
    gCSock.Push((u_long*)&tmpULong,1);
  }
}
//=End of PushShipsDummy================================================================
//======================================================================================

//======================================================================================
//=PushShot=============================================================================
void PushShot(
        int TypeWeaponry,
```

```
        int CurrentCoordinatesX,
        int CurrentCoordinatesY,
        int TargetCoordinatesX,
        int TargetCoordinatesY
        )
{
  if (State==Idle) {
    State = PushingShots;
    charBuffer.clear();
    intBuffer.clear();
    g_NumElements=0;
  }
  if (State == PushingShots) {
    intBuffer.push_back((u_long)TypeWeaponry);
    intBuffer.push_back((u_long)CurrentCoordinatesX);
    intBuffer.push_back((u_long)CurrentCoordinatesY);
    intBuffer.push_back((u_long)TargetCoordinatesX);
    intBuffer.push_back((u_long)TargetCoordinatesY);
    g_NumElements++;
  }
}
//=End of PushShot=================================================================
//=================================================================================

//=================================================================================
//=PushShotsDummy==================================================================
void PushShotsDummy(void)
{
  if (State == Idle) {
    PushHeader(3,4);
    u_long tmpULong = 0;
    gCSock.Push((u_long*)&tmpULong,1);
  }
}
//=End of PushShotsDummy===========================================================
//=================================================================================

//=================================================================================
//=SendData========================================================================
int SendData(void)
{
  if (State == PushingParams) {
```

```
    PushHeader(0,(int)(4*intBuffer.size()));
    FlushIntBuffer();
    State=Idle;
    return EC_OK;
  }
  if (State == PushingShots) {
    PushHeader(3,(int)(4 * (intBuffer.size()+1)));
    gCSock.Push((u_long*)&g_NumElements,1);
    FlushIntBuffer();
    State=Idle;
    return EC_OK;
  }
  if (State == PushingFixedShipPart || State == PushingVariableShipPart) {
    PushHeader(2,(int)(4+g_ByteSize));
    gCSock.Push((u_long*)&g_NumElements,1);
    FlushShipBuffer();
    State=Idle;
    return EC_OK;
  }
  if (State != Idle) {
    return EC_FAULT;
  }
  return EC_OK;
}
//=End of SendData=================================================================
//=================================================================================

//=================================================================================
//=End of Interface Part===========================================================
//=================================================================================

//=================================================================================
//=Routine Part====================================================================
//=================================================================================

//=================================================================================
//=PushHeader======================================================================
void PushHeader(char ID, int LEN)
{
  gCSock.Push(&ID,1);
  gCSock.Push((u_long*)&LEN,1);
}
```

```
//=End of PushHeader==============================================================
//================================================================================

//================================================================================
//=FlushIntBuffer=================================================================
void FlushIntBuffer(void)
{
  std::list<u_long>::const_iterator runIBuf;
  runIBuf=intBuffer.begin();
  u_long tmpUL;
  while (runIBuf!=intBuffer.end()) {
    tmpUL=*runIBuf;
    gCSock.Push(&tmpUL,1);
    runIBuf++;
  }
  intBuffer.clear();
}
//=End of FlushIntBuffer==========================================================
//================================================================================

//================================================================================
//=FlushShipBuffer================================================================
void FlushShipBuffer(void)
{
  std::list<StructShip>::const_iterator runShBuf;
  std::list<char>::const_iterator runChBuf;;

  runShBuf=shipBuffer.begin();
  u_long tmpUL;
  while (runShBuf!=shipBuffer.end()) {
    int i;
    for (i=0; i<12; i++) {
      tmpUL=(*runShBuf).fixPart[i];
      gCSock.Push(&tmpUL,1);
    }
    runChBuf=(*runShBuf).varPart.begin();
    while (runChBuf!=(*runShBuf).varPart.end()) {
      gCSock.Push((const char*)&(*runChBuf),1);
      runChBuf++;
    }
    runShBuf++;
  }
```

```
  shipBuffer.clear();
}
//=End of FlushShipBuffer=============================================================
//====================================================================================


//====================================================================================
//=End of Routine Part================================================================
//====================================================================================
```

Рисунок 161. Файл «ZSWrap.cpp».