Санкт-Петербургский национальный исследовательский университет
информационных технологий, механики и оптики

Факультет информационных технологий и программирования

Кафедра компьютерных технологий

Н.И. Макаров

Объектно-ориентированное программирование с явным выделением состояний на языке *Eiffel*

Курсовая работа

Санкт-Петербург

2012

Оглавление

Введение 3
1. Постановка задачи 4
1.1. Библиотека для поддержки автоматного программирования на языке *Eiffel* 4
1.2. Примеры использования библиотеки 4
2. Реализация 5
2.1. Библиотека 5
2.2. Пример игры *Reversi* 6
Заключение 9
Источники 10

Введение

В настоящее время существует несколько приемов создания исходного кода для сущностей со сложным поведением. Программирование с явным выделением состояний (автоматное программирование) допускает использование ООП для выполнения этой задачи. Такое совмещение парадигм объединяет преимущества каждой из них.

Бертран Мейер разработал объектно-ориентированный язык программирования *Eiffel*. Этот язык лишен некоторых недостатков традиционных языков программирования, поэтому он вполне подходит для реализации парадигмы автоматного программирования – создания системы, которая позволяет использовать эту парадигму.

1. Постановка задачи

1.1. Библиотека для поддержки автоматного программирования на языке *Eiffel*

Парадигма автоматного программирования могут быть реализованы с помощью циклов, операторов `if` или `switch`. Однако синтаксис объектно-ориентированного языка программирования *Eiffel* позволяет использовать многие эффективные методы разработки: парадигму ООП, проектирование по контракту, исключения, множественное наследование, универсализация, управление памятью и т. д. При этом среда разработки *EiffelStudio* реализует этот потенциал. Поэтому можно объединить преимущества автоматного и объектно-ориентированного программирования.

Целью данной курсовой работы является создание библиотеки на языке *Eiffel*, которая позволяет программировать сущности с явным выделением состояний, задавать их «сложное поведение» (зависящее от состояния). Эта библиотека должна поддерживать использование «состояний», «функций переходов» (параметрами к которым являются «входные воздействия»), «функций выходов» (и «выходные воздействия», которые они вызывают).

1.2. Примеры использования библиотеки

Для того чтобы можно было наглядно убедиться в преимуществах совместного использования автоматного и объектно-ориентированного подходов, приведем пример использования разработанной библиотеки при решении конкретной задачи: создание программы, поддерживающей популярную игру *Reversi* для двух человек.

2. Реализация

2.1. Библиотека

В рамках библиотеки, приведенной в приложении 1, реализованы следующие классы:

- `STATE` – «состояние», для него можно задать имя в виде строки, кроме того объект данного класса может быть использован в виде индекса в хэш-таблице;
- `STATE_DEPENDENT_PROCEDURE` – процедура для смены состояния сущности. Можно добавлять различные сценарии поведения для разных состояний и входных воздействий (параметров), а специальный метод – «вызов процедуры» в качестве результата выдает определенное состояние, в которое должна перейти сущность. Таким образом, фактически реализована «функция переходов»;
- `STATE_DEPENDENT_FUNCTION` отличается от «`STATE_DEPENDENT_PROCEDURE`» тем, что вместо смены состояния лишь выдает результат, зависящий от «входных воздействий» и «состояния». Таким образом она представляет из себя «функцию выходов»;
- `AUTOMATED` – сущность с автоматным поведением. В каждый момент времени она «находится» в каком-то состоянии. В итоге именно этот класс «объединяет» предыдущие, представляя собой конечный автомат, у которого имеются необходимые составляющие: «текущее состояние», «функции переходов», «функции выходов», результат выполнения которых зависит, в том числе, и от «входных воздействий».

2.2. Пример игры *Reversi*

С использованием данной библиотеки была реализована игра *Reversi* двух человек на квадратной доске с фишками, окрашенными в белый цвет с одной стороны, и в черный – с другой (рис. 1).

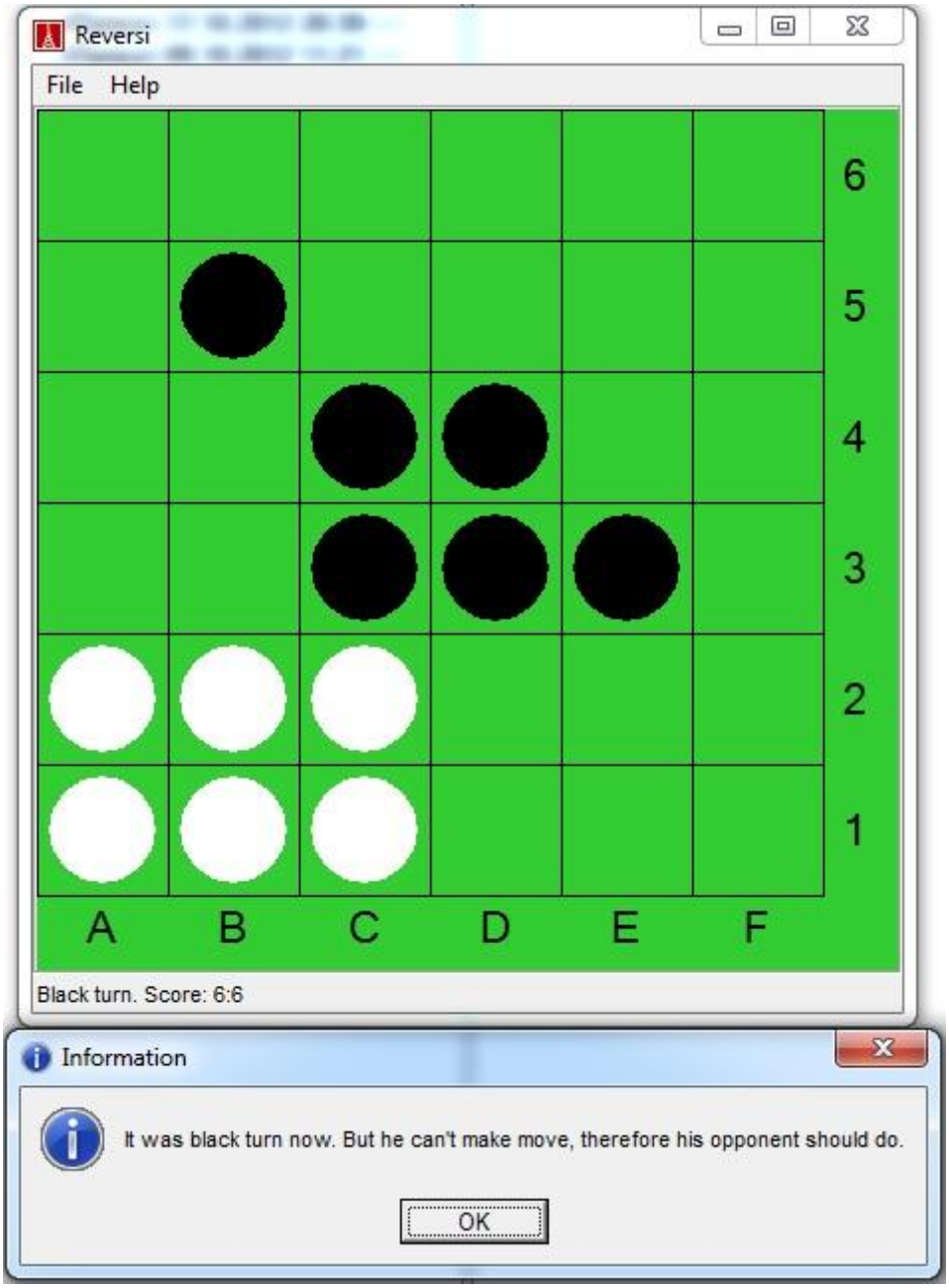

Рис. 1. Скриншот для игры *Reversi*

Это свойство сразу приводит к интуитивному решению: создать сущность «фишка» с двумя состояниями.

В данном примере были реализованы следующие автоматные сущности из предметной области:

- MARKER – поле доски с возможной фишкой на нем, может быть в следующих состояниях: «White», «Black», «Empty» (пока на этом поле нет никакой фишки), «White_can» (на это поле можно поставить белую фишку), а также «Black_can» (на это поле можно поставить черную фишку). Также были введены функции и процедуры, результат выполнения которых зависит от состояния, например, «перевернуть фишку», признак «белая ли фишка», «показать/убрать подсказку» (поля, в которые игрок, обладающий ходом, может положить новую фишку), «сделать ход» (игрок выбрал именно это поле для хода, возможно только в состояниях «White_can» и «Black_can»). В результате был разработан автомат, представленный на рис.2.

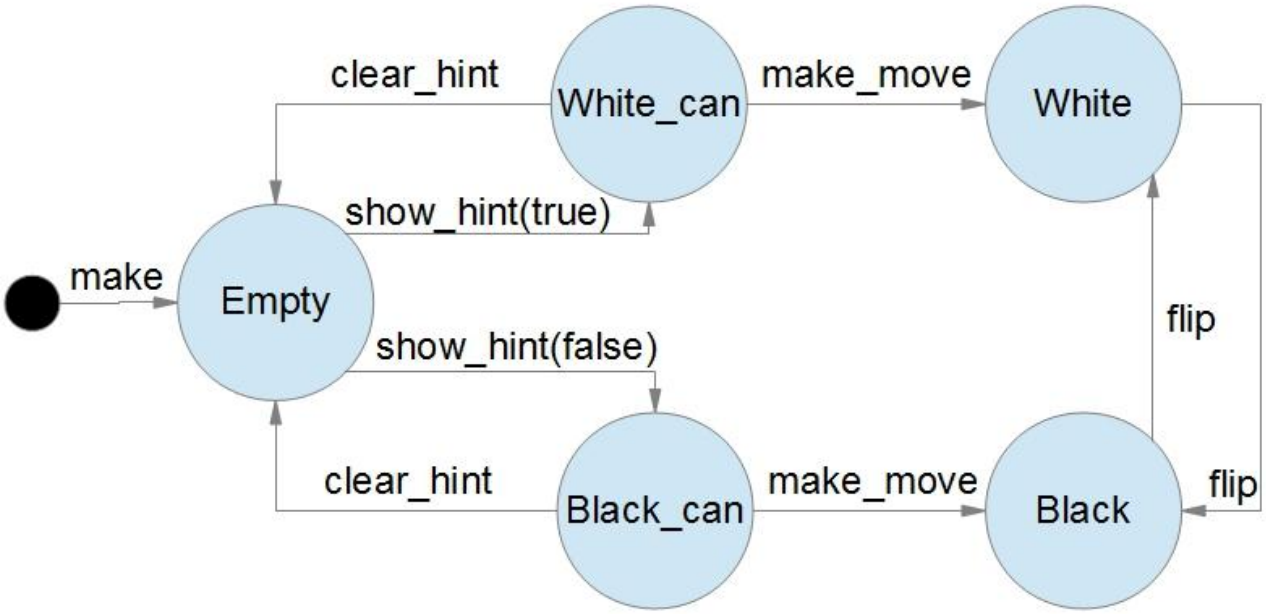

Рис. 2. Диаграмма состояний автомата *MARKER*

- GAME_MANAGER – класс, который содержит информацию о текущем ходе игры. В частности, состояния определяют, ходит ли сейчас игрок с белыми фишками (состояние «White_turn»), его соперник («Black_turn»), либо на доске создалась ситуация, когда кто-то не имеет возможности походить, либо вообще игра окончена («Game_over»). Вся эта информация сохраняется с помощью соответствующего объекта класса STATE, содержащего в себе состояние текущего хода игры («turn_state»). Также введены функции перезапуска игры, служебная функция «игрок сделал ход» для смены состояния и другие. В результате был разработан автомат, представленный на рис. 3.

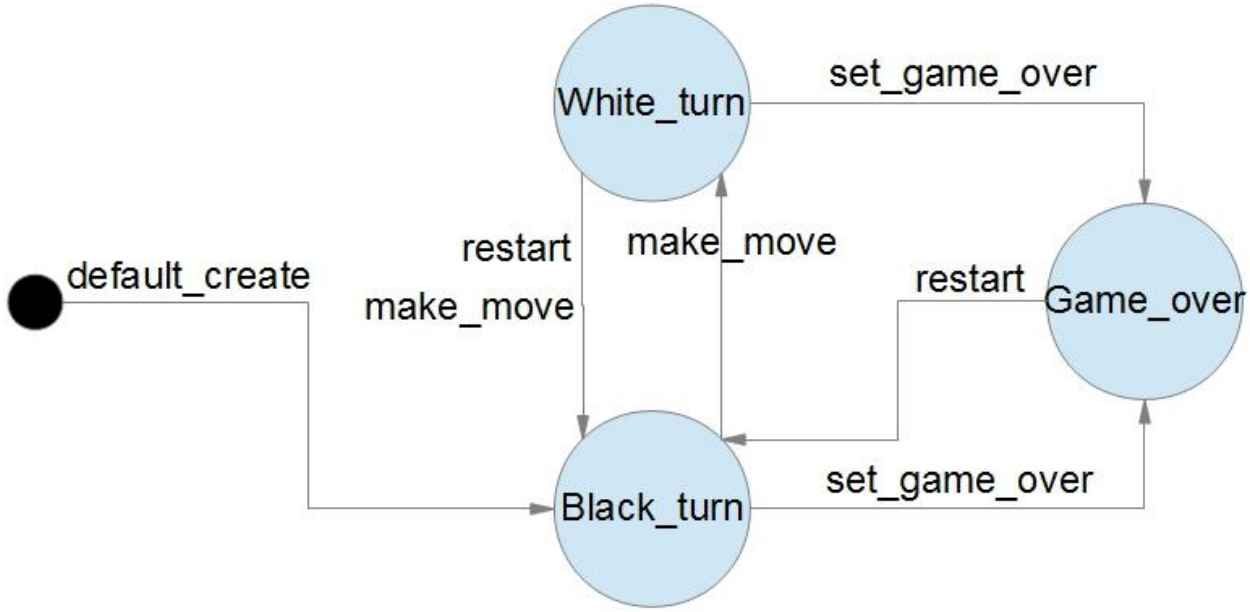

Рис. 3. Диаграмма состояний *GAME_MANAGER*

После проектирования игры с использованием построенных автоматов приступим к ее реализации. Здесь потребовался класс `STATE`, который был создан в предложенной библиотеке для поддержки автоматного программирования на языке *Eiffel*. Как видно из диаграммы классов (рис. 4), классы `GAME_MANAGER` и `MARKER` используют библиотечный класс «`STATE`».

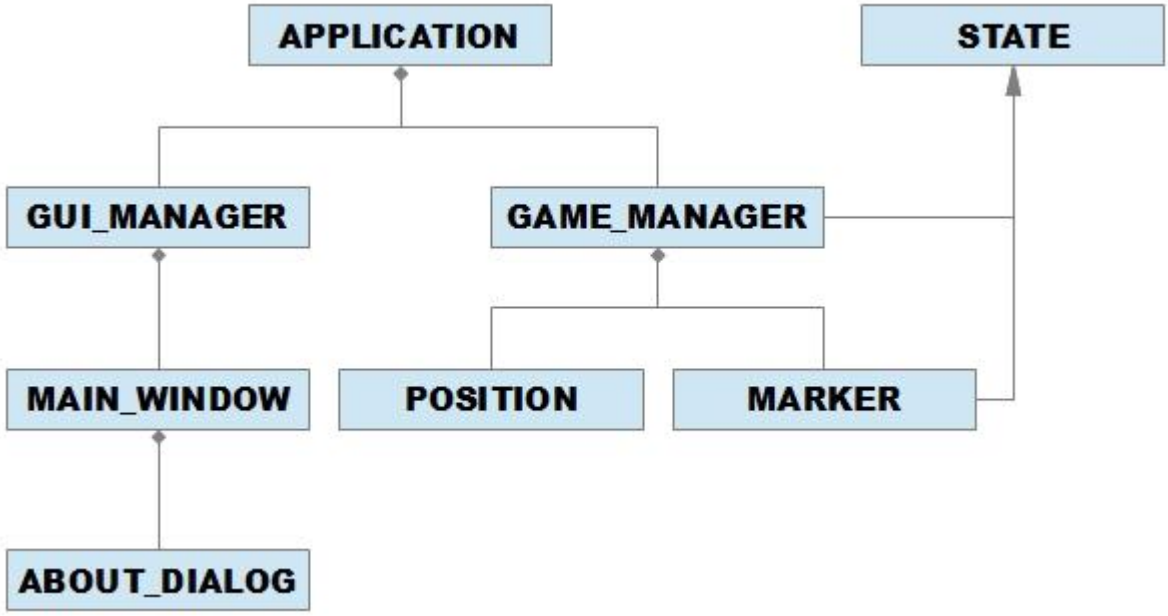

Рис. 4. Диаграмма классов

Заключение

Данная библиотека открывает возможность для единообразного создания автоматных сущностей с явным выделением состояний, вводит необходимые классы для их программирования. Она позволяет объединить преимущества ООП и автоматного программирования. С ее помощью значительно упростилась реализация игры *Reversi*, так как этот пример способствует использованию автоматов, а библиотека ускоряет написание исходного кода, не внося побочных сложностей благодаря инкапсуляции, наследованию и полиморфизму.

Источники

[1] Meyer B. Object-Oriented Software Construction, 2000.

[2] Поликарпова Н. И., Шалыто А. А. Автоматное программирование. СПб.: Питер, 2009.

[3] http://en.wikipedia.org/wiki/Automata-Based_Programming

[4] Шалыто А. А., Туккель Н. И. Программирование с явным выделением состояний // Мир ПК. 2001. № 8, 9. http://is.ifmo.ru (раздел «Статьи»).

[5] Шопырин Д. Г., Шалыто А. А. Объектно-ориентированный подход к автоматному программированию //Информационно-управляющие системы. 2003. №5. http://is.ifmo.ru (раздел «Статьи»).

[6] Раер М. Г. Автоматное расширение языка C#. СПбГУ ИТМО. 2006. http://is.ifmo.ru (раздел «Дипломы»).