Модули для создания игр типа «Тетрис» с квадратными элеменентарными блоками

Кочурко Александр

25. 05. 2013

Идея проекта, реализация

Имеется четыре равных квадрата. Сколько различных фигур можно составить путём прикладывания квадратов друг к другу гранями?

Очевидно, что таких фигур будет семь. Советский учёный Алексей Пажитнов, используя эти фигуры, создал всемирно известную игру, за лицензию на выпуск которой на протяжении больше чем десяти лет бились ведущие игровые компании мира.

В качестве проекта я решил по-своему реализовать тетрис. Целью работы, однако является не создание самой игры, а применение на практике идеи полиморфизма классов и тренировка написания универсальных методов (это, кстати главная цель, и я даже немного горжусь тем, что у меня вышло).

Суть такова: объект класса «Field» представляет собой динамическую матрицу размером L на M. Поле обязательно имеет границу. Объект класса «Block» — тоже динамическая матрица, только меньших размеров и с большим числом приватных данных. Блоки ничего не знаю о размерах поля; методы, контролирующие движение и поворот блоков, в своей работе опираются на границу поля и на клетки, занятые другими блоками.

Наследниками блока как раз и являются 7 фигур: «Square», «Line», «StairLeft», «StairRight», «Pedestal», «CornerLeft» и «CornerRight».

Каждый класс (кроме класса интерфейса «ConField») оформлен в отдельном модуле. Все модули, содержащие блоки и поле, являются кроссплатформенными.

Первой созданной фигурой был «Square» - самый лёгкий блок, т. к. все его методы очень простые, а метод «Rotate» и его напарник «StopRotate» вообще не требуются.

Затем была реализована «Line». С ней дела обстояли уже посложнее, т. к. возникла необходимость в реализации метода «Rotate» (у этой фигуры уже две степени поворота). Однако и тут особых проблем в реализации не было (в основном благодаря тому, что, во первых, фигура состоит только из одной строки, и во вторых, все её клетки - «full».

Потом - «StairsLeft» и «StairsRight». Реализация этих фигур потребовала гораздо больше времени и раздумий, т. к. существенно усложнялись все методы. Но и тут фунуции, принадлежащие только этому классу, оказались не очень объёмными по количеству кода (хотя и больше, чем для предыдущих фигур).

И, наконец, мы дошли до фигур, имеющих четыре степени поворота. Расписывать все частные случаи в методах каждой из оставшихся фигур было бы слишком трудоёмко и нерационально. Тогда я решил реализовать все методы так, чтобы они были универсальными. Программа, предположительно, будет

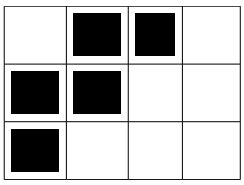
работать медленнее, но выигрыш колоссальный: если бы удалось так сделать, то новую фигуру можно было бы ввести, лишь описав, как она выглядит. Отпадёт потребность в написании методов для каждой из новых фигур.

На реализацию этих методов ушло больше всего времнени. Я считаю, что это была самая интересная задача в моём проекте. Разработанные методы работают для всех фигур, так что отпала необходимость в старых функциях, однако я их оставил (перенёс варианты фигур в отдельную папку), т. к. старые методы для своих фигур могут работать быстрее. Можете, кстати, проверить их — они полностью рабочие.

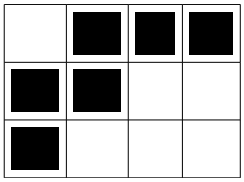
Таким образом, я максимально упростил себе задачу создания новых фигур. Пьедестал и углы были написаны за 10 минут. Также на всякий случай я создал тестовую фигуру и придавал ей разные формы, играл ей. Методы ни разу не дали сбой, даже если фигура разрывна. В варианте, который я вам прислал, тестовой фигурой является знак вопроса, и им тоже можно управлять. Так что фигуру теперь может создать любой желающий, даже не вдумываясь в программу: достаточно граммотно реализовать конструктор и деструктор.

ВАЖНО: при создании фигуры к неё не должно быть окаймляющих строк/столбцов в матрице! Однако допускается, чтобы пустые строки/столбцы не были крайними.

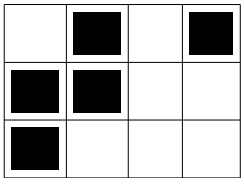
Так нельзя:



А так можно:



И так можно:



Также хотелось бы уделить немного внимания методу «KillLine» классса

«Field». Он хорош тем, что возвращает число одновременно уничтоженных линий.
Благодаря этому удалось сделать интересную вещь (я называю её «комментатор») , с которой вы ознакомитесь в процессе игры :)

Ещё важный момент — условие проигрыша. Иногда бывает, что вы проигрываете, при этом верхняя полоса поля ещё не заполнена. Это нормально! Просто следующая фигура имеет высоту 2 и не может поместиться в полоске высотой 1. Сначала я этого не предусмотрел, и у меня, когда оставалась одна свободная полоса, новая фигура, если её высота 2, перекрывала старые, что противоречит логике игры. Поэтому я изменил условие проигрыша, и теперь с этим проблем нет.

Интерфейс

В программе предусмотрено два консольных интерфейса: простейший (с использованием символов из таблицы UTF-8) и более интересный и сложный (реализован с помощью ncurses). Файлы интерфейсов специально не разбивались на модули, чтобы можно было быстро их замещать.

В первом варианте нет ничего, что заслуживало бы внимания, поэтому перейдём сразу ко второму.

Вообще говоря, я думал, что создание интерфеса окажется более простым заданием. Однако, как обычно, в процессе написания кода я частенько натыкался на подводные камни.

Было создано три дополнительных окна: первое — «infoscr» - окно со статической информацией, её содержимое почти не меняется. Второе окно - «gamescr» - непосредственно отображает нашу матрицу «field». И, наконец, третье - «statusscr» - показывает текущую информацию об игре.

Предусмотрено три уровня сложности, под сложностью подразумевается скорость падения фигур. Также, по сравнению, с первым интерфейсом, добавлена функция «gameBreak» - пауза.

Заключение

Процесс работы над проектом был очень интересным, я, впринципе, доволен своей программой. Теперь частенько играю в свой тетрис в консоли (кстати на лёгком уровне сложности набрал 116 очков :)). В будущем планирую реализовать тетрис на Qt.

Разбираясь с ncurses, я пользовался статьёй «Программирование с помощью ncurses », Автором которой является Андрей Боровский.