Министерство образования Республики Беларусь
Учреждение образования «Институт информационных технологий
Белорусского государственного университета информатики и
радиоэлектроники»

Факультет компьютерных технологий

Кафедра информационных систем и технологий

Дисциплина: Современные платформы программирования

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему

«Игра Тетрис»

БГУИР КП 1-40 01 01 004 ПЗ

Студент: гр. 381064 Данелян П.В.

Руководитель: Бакунова О.М.

Минск 2015

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Основная цель данной курсовой работы – написание программы на языке Java, приобретение знаний и практических навыков самостоятельного программирования задач, а также освоение инструментальных средств отладки и методов программирования.

Для достижения цели была выбрана задача «Игра Tetris». Уровень сложности этой задачи позволяет ознакомиться с основными этапами написания реальных программ, и приобрести определенные навыки программирования. Игра «Tetris» хорошо знакома не только опытным пользователям консольных аркад, но и многим новичкам мира игр.

Тетрис — компьютерная игра, изобретённая в СССР .

Название игры происходит от количества клеток, из которых состоит каждая фигура.

По правилам случайные фигурки тетрамино падают сверху в прямоугольный стакан шириной 10 и высотой 20 клеток. В полёте игрок может поворачивать фигуркывается фигурка, которая будет следовать после текущей — это подсказка, которая позволяет планировать игроку действия. Темп игры постепенно увеличивается. Игра заканчивается, когда новая фигурка не может поместиться в стакан. Игрок получает очки за каждый заполненный ряд, поэтому его задача — заполнять ряды, не заполняя сам стакан (по вертикали) как можно дольше, чтобы таким образом получить как можно больше очков.

Начисление очков в разных версиях «Тетриса» довольно разнообразное. Очки могут начисляться за убранные линии, за сброшенные фигурки, за переход на новую скорость и тому подобное.

При начислении очков за линии количество очков обычно зависит от того, сколько линий убрано за один раз. Например, в китайских «Тетрисах», популярных в СНГ в 1990-х годах, начисление очков обычно было таким: 1 линия — 100 очков, 2 линии — 300 очков, 3 линии — 700 очков, 4 линии (то есть, сделать Тетрис) — 1500 очков. То есть, чем больше линий убирается за один раз, тем больше отношение количества очков к количеству линий. Любопытно, что тетрисом во многих версиях игры также называется действие, после которого исчезает сразу 4 линии. Это можно сделать только одним способом — сбросить «палку» (фигурку, в которой все клетки расположены на одной линии) в «шахту» ширины 1 и глубины как минимум 4.

При начислении очков за сброшенные фигурки могут учитываться высота, на которой остановилась фигурка (например, чем ниже, тем лучше), расстояние, которое пролетела фигурка после «сбрасывания» (ускорения падения). Хотя обычно приоритетом являются линии, а за фигурки начисляется относительно небольшое количество очков.

Игра реализована практически на всех современных компьютерах, включая КПК, мобильные телефоны, игровые видеоприставки, телевизоры (как доп. функция), множество карманных игровых устройств. Есть варианты игры для всех сколько-нибудь распространённых ОС, а также для Java. Есть порт даже для осциллографа[7]. Трудно, если вообще возможно, назвать такую вычислительную платформу, где бы не было этой игры.

Пожалуй, наибольшую популярность приобрела реализация тетриса для игровой консоли Game Boy и видеоприставки NES (и её многочисленных клонов).

Во многих реализациях стакан изначально не пуст. Есть реализации (например, бесплатная Gravytris для Microsoft Windows) с более реалистичными правилами гравитации: например, при пропадании горизонтального ряда блоки, которые выше его, соединяются в связные

области и каждая область падает, пока не наткнётся на блок; это может привести к заполнению новых рядов и новым падениям, и так далее.

Краткое описание разделов:

1 Анализ предметной области – в этом разделе описывается назначение и функции программы, аналоги программ.

2 Проектирование задачи – в этом разделе описывается обоснование инструментов разработки, описание алгоритма решения задачи;

3 Разработка программного средства – в этом разделе описывается обоснование языка программирование, архитектура ПО;

4 Тестирование – в этом разделе идёт описание о самом тестировании, их видах и о том, как была протестирована программа;

5 Методика использования программного средства – в этом разделе идёт описание о назначение программы, требования программы к аппаратным ресурсам ПК и об руководстве пользователя.

1 ПОСТАНОВКА ЗАДАЧИ

Практически на всех рабочих местах сегодня можно встретить компьютер. Электронно-вычислительная техника (ЭВТ) помогает людям в их повседневной деятельности, повышая производительность их труда.

Целью данного проекта является создание игры Тетрис. Игра предназначена для тренировки вснимая, реакции. Корме того данная игра зарекомендовала себя еще с 80х годов прошлого века. Она была портирована на большинство современных платформ.

1.1 ВХОДНАЯ ИНФОРМАЦИЯ

Входная информация является неотъемлемой частью любой программы. Подобное программное средство взаимодействует с пользователем и позволяет автоматизировать некоторые задачи. Входной информацией данного программного средства будет вводимая пользователем, запускающим процессы.

1.2 ВЫХОДНАЯ ИНФОРМАЦИЯ

Выходная информация используется для того, чтобы видеть результаты работы программы. Выходной информацией будет список запущенных процессов и их потоки.

2 ЛОГИЧЕСКАЯ МОДЕЛЬ ДАННЫХ

2.1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

Игра Тетрис предназначается для тренировки памяти, реакции и умения делать расчеты. Однако основной концепцией которой придерживался я в данной работе это запуск программы из окна консольного терминала. Причиной этому послужило то, что как правило на сервера не ставится графическая оболочка и по этой причине, если администратор например запустил долгий процесс, он может отвлечься на пару минут игрой в Тетрис.

2.2 МОДЕЛЬ ПРЕДМЕТНОЙ ОБЛАСТИ

Состоять программное средство будет из окна с пользовательским интерфейсом. Пользователь будет взаимодействовать с программой при помощи посылки сигналов с клавиатуры. Так же в качестве окна используется окно консолького терминала. Это означает что можно пользоваться коммандами консоли для сохранения результата, запуска по рассписание и т.д..

3 ФИЗИЧЕСКАЯ МОДЕЛЬ ДАННЫХ

Программное средство разрабатывалось с помощью технологии и методов построения приложений, предоставляемых, непосредственно, языком Java.

3.1 ВЫБОР И ОБОСНОВАНИЕ СРЕДСТВ РАЗРАБОТКИ

Для написания данного проекта был выбран язык программирования Java.

Программы на Java транслируются в байт-код, выполняемый виртуальной машиной Java (JVM) — программой, обрабатывающей байтовый код и передающей инструкции оборудованию как интерпретатор.

Достоинством подобного способа выполнения программ является полная независимость байт-кода от операционной системы и оборудования, что позволяет выполнять Java-приложения на любом устройстве, для которого существует соответствующая виртуальная машина. Другой важной особенностью технологии Java является гибкая система безопасности, в рамках которой исполнение программы полностью контролируется виртуальной машиной. Любые операции, которые превышают установленные полномочия программы (например, попытка несанкционированного доступа к данным или соединения с другим компьютером), вызывают немедленное прерывание.

Часто к недостаткам концепции виртуальной машины относят снижение производительности. Ряд усовершенствований несколько увеличил скорость выполнения программ на Java:

•применение технологии трансляции байт-кода в машинный код непосредственно во время работы программы (JIT-технология) с возможностью сохранения версий класса в машинном коде,

•широкое использование платформенно-ориентированного кода (native-код) в стандартных библиотеках,

•аппаратные средства, обеспечивающие ускоренную обработку байт-кода (например, технология Jazelle, поддерживаемая некоторыми процессорами фирмы ARM).

По данным сайта shootout.alioth.debian.org, для семи разных задач время выполнения на Java составляет в среднем в полтора-два раза больше, чем для C/C++, в некоторых случаях Java быстрее, а в отдельных случаях в 7 раз медленнее[12]. С другой стороны, для большинства из них потребление памяти Java-машиной было в 10—30 раз больше, чем программой на C/C++. Также примечательно исследование, проведённое компаниейGoogle, согласно которому отмечается существенно более низкая производительность и бóльшее потребление памяти в тестовых примерах на Java в сравнении с аналогичными программами на C++.

Идеи, заложенные в концепцию и различные реализации среды виртуальной машины Java, вдохновили множество энтузиастов на расширение перечня языков, которые могли бы быть использованы для создания программ, исполняемых на виртуальной машине. Эти идеи нашли также выражение в спецификации общеязыковой инфраструктуры CLI, заложенной в основу платформы .NET компанией Microsoft.

4 ОПИСАНИЕ ПРОГРАММЫ

Программа является консольным игровым приложением. Основное назначение — это снять стресс после долгой работы.

4.1 ВЫБОР И ОБОСНОВАНИЕ СРЕДЫ РАЗРАБОТКИ

В данном случае средой разработки приложения была выбрана Eclipse.

Eclipse — свободная интегрированная среда разработки модульных кроссплатформенных приложений. Развивается и поддерживается Eclipse Foundation.

Наиболее известные приложения на основе Eclipse Platform — различные «Eclipse IDE» для разработки ПО на множестве языков (например, наиболее популярный «Java IDE», поддерживавшийся изначально, не полагается на какие-либо закрытые расширения, использует стандартный открытый API для доступа к Eclipse Platform).

Первоначально Eclipse разрабатывалась фирмой IBM как преемник среды разработки IBM VisualAge, в качестве корпоративного стандарта IDE для разработки на разных языках под платформы IBM. По сведениям IBM, проектирование и разработка стоили 40 миллионов долларов. Исходный код был полностью открыт и сделан доступным после того, как Eclipse был передан для дальнейшего развития независимому от IBM сообществу.

В Eclipse 3.0 (2003 год) были выбраны спецификации сервисной платформы OSGi, как архитектура среды исполнения. С версии 3.0 Eclipse перестал быть монолитной IDE, поддерживающей расширения, а сам стал набором расширений. В основе лежат фреймворк OSGi и SWT/JFace, на основе которых разработан следующий слой — RCP (Rich Client Platform,

платформа для разработки полноценных клиентских приложений). RCP служит основой не только для Eclipse, но и для других RCP-приложений, например, Azureus и File Arranger. Следующий слой — сам Eclipse, представляющий собой набор расширений RCP — редакторы, панели, перспективы, модуль CVS и модуль Java Development Tools (JDT).

С 2006 года фонд Eclipse координирует ежегодный общий релиз (Simultaneous Release), который происходит в июне. Каждый выпуск включает в себя платформу Eclipse, а также ряд других проектов Eclipse.

Eclipse служит в первую очередь платформой для разработки расширений, чем он и завоевал популярность: любой разработчик может расширить Eclipse своими модулями. Уже существуют Java Development Tools (JDT), C/C++ Development Tools (CDT), разрабатываемые инженерами QNX совместно с IBM, и средства для языков Ada (GNATbench, Hibachi), COBOL, FORTRAN, PHP, X10 (X10DT) и пр. от различных разработчиков. Множество расширений дополняет среду Eclipse диспетчерами для работы с базами данных, серверами приложений и др.

Eclipse JDT (Java Development Tools) — наиболее известный модуль, нацеленный на групповую разработку: среда интегрирована с системами управления версиями — CVS, GIT в основной поставке, для других систем (например, Subversion, MS SourceSafe) существуют плагины. Также предлагает поддержку связи между IDE и системой управления задачами (ошибками). В основной поставке включена поддержка трекера ошибок Bugzilla, также имеется множество расширений для поддержки других трекеров (Trac, Jira и др.). В силу бесплатности и высокого качества, Eclipse во многих организациях является корпоративным стандартом для разработки приложений.

Eclipse написана на Java, потому является платформо-независимым продуктом, за исключением библиотеки SWT, которая разрабатывается для всех распространённых платформ (см. ниже). Библиотека SWT используется

вместо стандартной для Java библиотеки Swing. Она полностью опирается на нижележащую платформу (операционную систему), что обеспечивает быстроту и натуральный внешний вид пользовательского интерфейса, но иногда вызывает на разных платформах проблемы совместимости и устойчивости приложений.

4.2 ОПИСАНИЕ ИНТЕРФЕЙСА

Графический интерфейс пользователя, графический пользовательский интерфейс, ГИП (англ. graphical user interface, GUI) — разновидность пользовательского интерфейса, в котором элементы интерфейса (меню, кнопки, значки, списки и т. п.), представленные пользователю на дисплее, исполнены в виде графических изображений.

В отличие от интерфейса командной строки, в GUI пользователь имеет произвольный доступ (с помощью устройств ввода — клавиатуры, мыши, джойстика и т. п.) ко всем видимым экранным объектам (элементам интерфейса) и осуществляет непосредственное манипулирование ими. Чаще всего элементы интерфейса в GUI реализованы на основе метафор и отображают их назначение и свойства, что облегчает понимание и освоение программ неподготовленными пользователями.

Графический интерфейс пользователя является частью пользовательского интерфейса и определяет взаимодействие с пользователем на уровне визуализированной информации. Недостатком подобного типа интерфейса является ограниченность изобразительных средств по причине ограниченности количества символов, включённых в состав шрифта, предоставляемого аппаратурой.

В отличие от интерфейса командной строки, пользователь имеет произвольный доступ (с помощью клавиатуры или указательного устройства ввода) ко всем видимым экранным объектам — элементам интерфейса, а на экране реализуется модель мира в соответствии с некоторой метафорой и осуществляется прямое манипулирование.

Разработанное программное средство не обладает обширным функционалом, но имеет дружелюбный графический интерфейс. От пользователей необходимы только ввод имени процесса взаимодействие с кнопками на главном окне.

4.3 ПРОГРАММНО-АППАРАТНЫЕ РЕСУРСЫ ПК

Для корректной работы приложения среда использования должна удовлетворять следующим требованиям:

- операционная система из семейства:

a) Linux;
b) BSD;
c) OS X.

- аппаратные требования:

a) 50 Мб свободного пространства на жестком диске;
b) 50 Мб оперативной памяти;

5 ТЕСТИРОВАНИЕ

Тестирование – процесс подготовки тестов и выполнение программы для этих тестов, с целью доказательства факта наличия в программе ошибки.

Тестирование – это любая деятельность, направленная на обнаружение ошибок в программном продукте. Тестирование проводится для того, чтобы найти ошибки в программе и тем самым повысить ее надежность, а следовательно - ценность. Если мы тестируем программу, то нам нужно окупить затраты на тестирование, каким-либо образом повысив стоимость программы. Это можно сделать, только повысив надежность программы, ради чего тестирование и проводится.

Повысить надежность можно только исправлением ошибок, внесенных в процессе разработки. После тестирования нельзя гарантировать отсутствие ошибок, можно лишь говорить о некотором уровне уверенности в правильности работы системы. Ошибка программы – наличие в программе дефекта, который проявляется в том, что программа не может быть выполнена или результаты работы программы отличаются от ожидаемых.

В ходе тестирования возникают следующие типы ошибок:

- синтаксические - ошибки компиляции (результат незнания языка программирования или невнимательность программирования),обнаруженные на этапе трансляции программы;

- логические - результат грубого просмотра, непонимания алгоритма программы, иногда непонимание смысла некоторых конструкций языка программы. Ошибки в логике программы, но обусловленные внешними по отношению к алгоритму факторами, т.е. алгоритм правильный, а программа составлена неправильно;

- семантические ошибки - ошибки времени выполнения – программы, которая пытается сделать что-нибудь запрещенное во время ее выполнения.

5.1 АНАЛИЗ НАДЁЖНОСТИ АЛГОРИТМА

Строгое математическое доказательство правильности работы алгоритма - обычно очень трудная задача, главным образом из-за того, что трудно доказать правильность работы циклов и рекурсивных процедур. Вместе с тем демонстрация правильной работы алгоритмов на некотором наборе тестов еще не означает, что он всегда будет работать правильно. Следует помнить, что различных комбинаций входных данных бывает, как правило, бесконечно (или "практически" бесконечно) много. Поэтому необходимо сопровождать алгоритм некоторым рассуждением, которое, даже не будучи строгим доказательством, в достаточно полной мере убеждает нас в правильности алгоритма. Конечно, оно не должно быть рассуждением в таком, например, стиле: "алгоритм перебирает все варианты, поэтому он правилен". Ведь тогда возникает вопрос, а как убедиться, что алгоритм действительно перебирает все варианты.

Наилучшим реальным подходом к обоснованию алгоритма является его правильность "по построению", когда используется метод пошаговой разработки. Чтобы получить правильный алгоритм, необходимо следить за правильностью детализации его шагов в ходе такого построения. Но это уже значительно более простая задача: как правило, детализация шага происходит в соответствии с определением того, что он должен делать.

При таком подходе построение алгоритма и его обоснование тесно переплетаются друг с другом. При этом следует, конечно, понимать, что если на этапе анализа задачи был выбран неверный подход к ее решению, то даже самая аккуратная последующая детализация исходной спецификации уже не позволит получить правильный алгоритм.

Обоснование алгоритма будет выглядеть еще более убедительно, если его дополнить индивидуальными доказательствами, позволяющими убедиться в правильной работе хотя бы некоторых циклов.

Используя метод пошаговой разработки, не следует забывать о таком мощном средстве доказательного программирования, как аннотирование программы утверждениями, размещенными в скобках комментариев. Аннотации описывают свойства вычислений в соответствующих точках программы. Они помогают избежать ошибок при шагах детализации и в обосновании их правильности. Кроме того, аннотированная программа может выступать в качестве доказательства своей правильности.

Во время тестирования курсового программного средства были выявлены и устранены существенные ошибки, связанные с соединением с сервером, и проблему получения ответа от клиента.

5.2 СРЕДСТВА ОБНАРУЖЕНИЯ ОШИБОК

Сегодня все большую популярность приобретает test-driven development(TDD), техника разработки ПО, при которой сначала пишется тест на определенный функционал, а затем пишется реализация этого функционала. На практике все, конечно же, не настолько идеально, но в результате код не только написан и протестирован, но тесты как бы неявно задают требования к функционалу, а также показывают пример использования этого функционала.

JUnit используется в двух вариантах JUnit 3 и JUnit 4. Обе версии, так как в старых проектах до сих пор используется 3-я, которая поддерживает Java 1.4.

Для создания теста нужно унаследовать тест-класс от *TestCase*, переопределить методы *setUp* и *tearDown* если надо, ну и самое главное — создать тестовые методы(должны начинаться с *test*). При запуске теста сначала создается экземляр тест-класса(для каждого теста в классе отдельный экземпляр класса), затем выполняется метод *setUp*, запускается сам тест, ну и в завершение выполняется метод *tearDown*. Если какой-либо из

методов выбрасывает исключение, тест считается провалившимся.

5.3 АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ

В ходе выполнения курсового проекта на тему «Игра Тетрис», была выполнена поставленная задача, а именно: создана игра написанная на языке Java поставляемая исполняемыми файлами длдя запуска. Так же с предусмотренным меню помощи.

6 ОПИСАНИЕ ПРИМЕНЕНИЯ

6.1 СПОСОБ УСТАНОВКИ ПРОГРАММЫ

Программа требует предворительной компиляции при помощи утилиты javac.

Javac — оптимизирующий компилятор языка java, включенный в состав многих Java Development Kit (JDK).

Компилятор принимает исходные коды, соответствующие спецификации Java language specification (JLS), и возвращает байт-код, соответствующий спецификации Java Virtual MachineSpecification (JVMS).

Javac написан на Java. Может вызваться непосредственно из java-программ.

Затем после компиляции запускается коммандой Java.

6.2 ДЕМОНСТРАТИВНЫЙ ПРИМЕР РАБОТЫ ПРОГРАММЫ

Для того, чтобы запустить программное средство, необходимо выполнить java tetris.

После запуска откроется окно диспетчера, что будет означать запуск программы (Рисунок 6.2.1).

Рисунок 6.2.1 – окно программы после запуска.

Процесс вызова помощи осуществляется вызовом команды h (Рисунок 6.2.2).

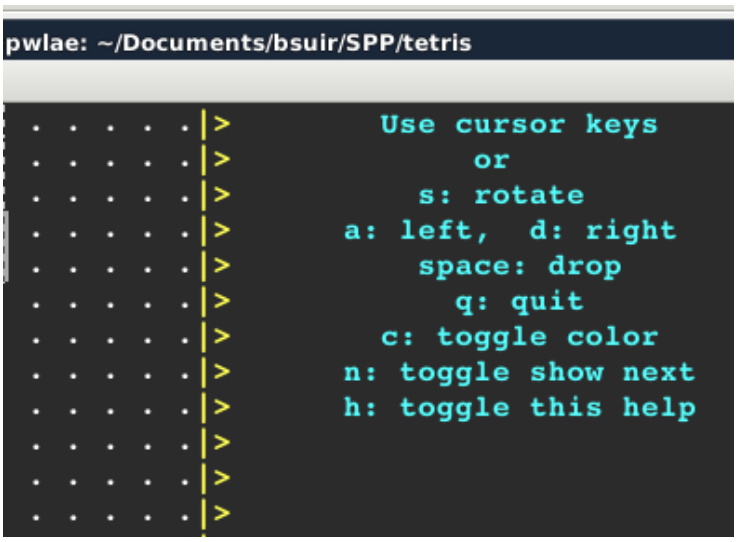

Рисунок 6.2.2 – окно справки.

По нажатию клавиши n мы можем убрать показ следующего элемента. (Рисунок 6.2.3).

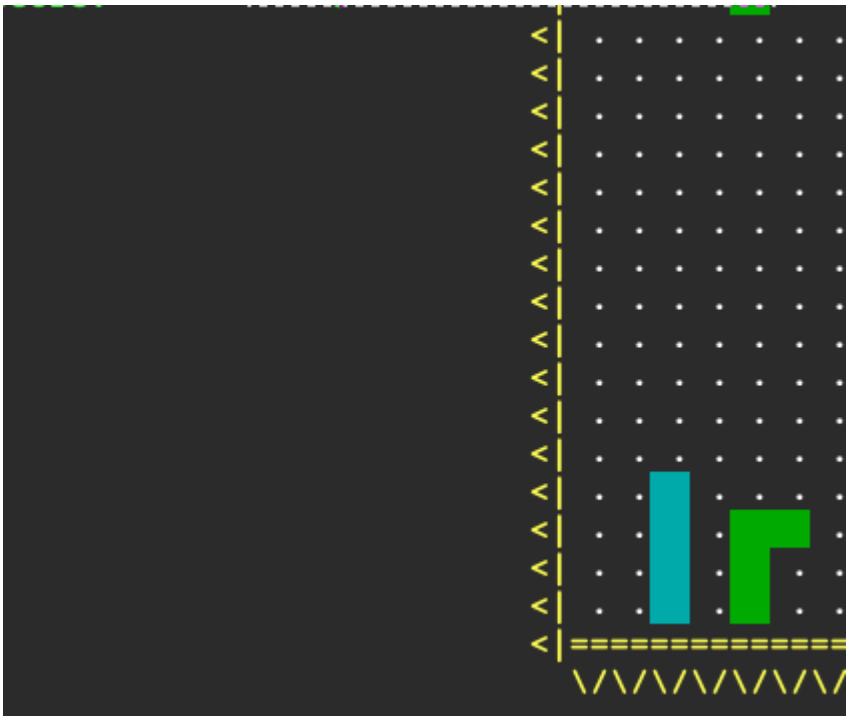

Рисунок 6.2.3 – следующий элемент не показан.

Окончание игры демонстрируется в консоли. (Рисунок 6.2.3).

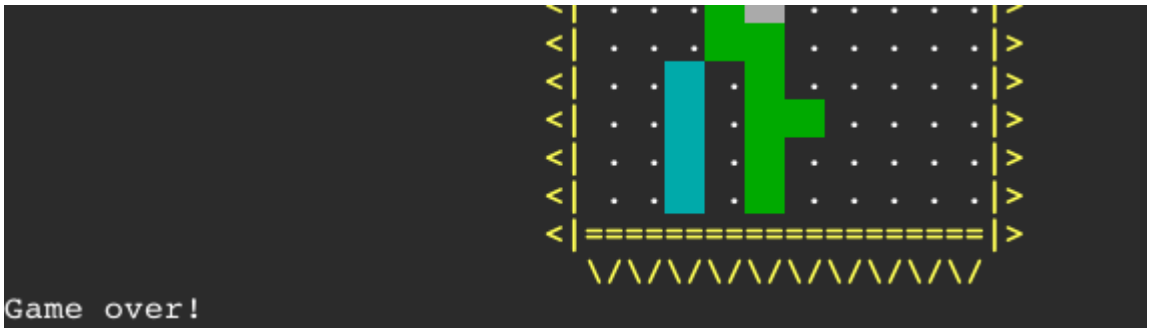

Рисунок 6.2.4 – запуск нового процесса.

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной курсовой работы «Игра Тетрис» были выполнены все функции, которые заранее были заданы перед разработчиком.

Я изучил основы языка программирования Java, получил азы работы с компиляторами, выделением памяти. Так же были изучены принципы работы JVM.

Особо важную роль я отметил в работе сборщиков мусора, Java платформа предоставляет их болшое количество, однако в контексте данного проекта не удалось полностью опробовать их прицип работы.

В перспективе данное приложение может быть портировано изконсолького приложения в WEB application. Что например позволило бы создать таблицу лучших результатов среди пользователей сети.

Так же хорошим тоном была бы привязка к легковесной реляционной или не реляционной баз данных, например sqlite или mongodb для сохранения результатов.

В текущую конфигурацию можно еще добавить поддержку различных режимов. Например чтоб пользователь мог нажатием клавиши изменять фигуру

Достоинства программы является то, что она бесплатная, требует мало ресурсов и запускается в консоли.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. М. Эллис, Б. Строуструп. Справочное руководство по языку Java с комментариями: Пер. с англ. - Москва: Мир, 1992. 445с.
2. Стенли Б. Липпман. Java для начинающих: Пер. с англ. 2тт. - Москва: Унитех; Рязань: Гэлион, 1992, 304-345сс.
3. Бруно Бабэ. Просто и ясно о Eclipse Java: Пер. с англ. - Москва: БИНОМ, 1994. 400с.
4. В.В. Подбельский. Язык Java: Учебное пособие. - Москва: Финансы и статистика, 1995. 560с.
5. Ирэ Пол. Объектно-ориентированное программирование с использованием Java: Пер. с англ. - Киев: НИИПФ ДиаСофт Лтд, 1995. 480с.
6. Т. Фейсон. Объектно-ориентированное программирование на Eclipse Java 4.5: Пер. с англ. - Киев: Диалектика, 1996. 544с.
7. Т. Сван. Освоение Java 4.5: Пер. с англ. - Киев: Диалектика, 1996. 544с.
8. Г. Шилдт. Самоучитель Java: Пер. с англ. - Санкт-Петербург: BHV-Санкт-Петербург, 1998. 620с.
9. У. Сэвитч. Java в примерах: Пер. с англ. - Москва: ЭКОМ, 1997. 736с.
10. К. Джамса. Учимся программировать на языке Java: Пер. с англ. - Москва: Мир, 1997. 320с.
11. В.А. Скляров. Язык C++ и объектно-ориентированное программирование: Справочное издание. - Минск: Вышэйшая школа, 1997. 480с.
12. "Системное программирование в среде Windows" (Д. Харт)
13. Х. Дейтел, П. Дейтел. Как программировать на C++: Пер. с англ. - Москва: ЗАО "Издательство БИНОМ", 1998. 1024с.
14. Гэри Неббет. Справочник по базовым функциям API Windows NT/2000 = Windows NT/2000 Native API Reference. — М.: «Вильямс», 2002. — С. 528. — ISBN 1-57870-199-6.

Приложение А
(Обязательное)
Листинг программы

```
import java.io.Console;
import java.io.IOException;
import java.io.Reader;
import java.util.Random;
import java.util.List;
import java.util.ArrayList;

public class Tetris {
    public static final int PLAYFIELD_W = 10;
    public static final int PLAYFIELD_H = 20;
    public static final int PLAYFIELD_X = 30;
    public static final int PLAYFIELD_Y = 1;
    public static final TetrisColor BORDER_COLOR = TetrisColor.YELLOW;

    public static final int HELP_X = 58;
    public static final int HELP_Y = 1;
    public static final TetrisColor HELP_COLOR = TetrisColor.CYAN;

    public static final int SCORE_X = 1;
    public static final int SCORE_Y = 2;
    public static final TetrisColor SCORE_COLOR = TetrisColor.GREEN;

    public static final int NEXT_X = 14;
    public static final int NEXT_Y = 11;

    public static final int GAMEOVER_X = 1;
    public static final int GAMEOVER_Y = PLAYFIELD_H + 3;

    public static final int INITIAL_MOVE_DOWN_DELAY = 1000;
    public static final double DELAY_FACTOR = 0.8;
    public static final int LEVEL_UP = 20;

    public static final String NEXT_EMPTY_CELL = "  ";
    public static final String PLAYFIELD_EMPTY_CELL = " .";
    public static final String FILLED_CELL = "[]";

    public static final Random RANDOM = new Random();

    public static void main(String[] args) {
        new Tetris().run();
    }

    public void run() {
        try {
            String[] cmd = {"/bin/sh", "-c", "stty raw -echo </dev/tty"};
            Runtime.getRuntime().exec(cmd).waitFor();
        } catch (IOException ioe) {
        } catch (InterruptedException ie) {
        }
        TetrisScreen screen = new TetrisScreen();
        TetrisController tc = new TetrisController(screen);
        Object lock = new Object();
        TetrisTicker tt = new TetrisTicker(tc, lock);
        Console console = System.console();
        Reader reader = console.reader();
        tt.start();
        char key[] = {0, 0, 0};
        while (tc.running) {
            key[2] = key[1];
```

```
            key[1] = key[0];
            try {
               if (key[2] == 27 && key[1] == '[') {
                  key[0] = (char)reader.read();
               } else {
                  key[0] = Character.toLowerCase((char)reader.read());
               }
            } catch (IOException ioe) {
            }
            synchronized(lock) {
               switch(key[0]) {
                  case 3:
                  case 'q':
                     tc.cmdQuit();
                     break;
                  case 'C':
                  case 'd':
                     tc.cmdRight();
                     break;
                  case 'D':
                  case 'a':
                     tc.cmdLeft();
                     break;
                  case 'A':
                  case 's':
                     tc.cmdRotate();
                     break;
                  case ' ':
                     tc.cmdDrop();
                     break;
                  case 'h':
                     tc.cmdToggleHelp();
                     break;
                  case 'n':
                     tc.cmdToggleNext();
                     break;
                  case 'c':
                     tc.cmdToggleColor();
                     break;
                  default:
                     break;
               }
               screen.flush();
            }
         }
      }
   }

   class TetrisTicker extends Thread {
      private static int delay = Tetris.INITIAL_MOVE_DOWN_DELAY;
      private TetrisController tc = null;
      private Object lock = null;

      public TetrisTicker(TetrisController tc, Object lock) {
         this.tc = tc;
         this.lock = lock;
      }

      public static void decreaseDelay() {
         delay *= Tetris.DELAY_FACTOR;
      }

      public void run() {
         while (true) {
            try {
```

```
                Thread.sleep(delay);
            } catch (java.lang.InterruptedException ie) {
            }
            synchronized(lock) {
                tc.cmdDown();
                tc.screenFlush();
            }
        }
    }
}

enum TetrisColor {
    RED (1),
    GREEN (2),
    YELLOW (3),
    BLUE (4),
    FUCHSIA (5),
    CYAN (6),
    WHITE (7);

    public final int value;

    TetrisColor(int value) {
        this.value = value;
    }

    private static final TetrisColor VALUES[] = values();

    public static TetrisColor getRandomColor() {
        return VALUES[Tetris.RANDOM.nextInt(VALUES.length)];
    }
}

class TetrisScreen {
    private boolean useColor = true;
    private StringBuffer sb = new StringBuffer();;

    public void print(String s) {
        sb.append(s);
    }

    public void xyprint(int x, int y, String s) {
        sb.append("\u001B[" + y + ";" + x + "H" + s);
    }

    public void showCursor() {
        sb.append("\u001B[?25h");
    }

    public void hideCursor() {
        sb.append("\u001B[?25l");
    }

    public void setFg(TetrisColor c) {
        if (useColor) {
            sb.append("\u001B[3" + c.value + "m");
        }
    }

    public void setBg(TetrisColor c) {
        if (useColor) {
            sb.append("\u001B[4" + c.value + "m");
        }
    }
```

```
    public void resetColors() {
        sb.append("\u001B[0m");
    }

    public void setBold() {
        sb.append("\u001B[1m");
    }

    public void clearScreen() {
        sb.append("\u001B[2J");
    }

    public void flush() {
        System.out.print(sb.toString());
        sb.setLength(0);
    }

    public void toggleColor() {
        useColor ^= true;
    }
}

abstract class TetrisScreenItem {
    public boolean visible = true;
    protected TetrisScreen screen = null;

    public abstract void draw(boolean visible);

    public TetrisScreenItem(TetrisScreen screen) {
        visible = true;
        this.screen = screen;
    }

    public void show() {
        if (visible) {
            draw(true);
        }
    }

    public void hide() {
        if (visible) {
            draw(false);
        }
    }

    public void toggle() {
        visible ^= true;
        draw(visible);
    }
}

class TetrisHelp extends TetrisScreenItem {
    private final TetrisColor color = Tetris.HELP_COLOR;
    private String[] text = {
        "  Use cursor keys",
        "       or",
        "    s: rotate",
        "a: left,  d: right",
        "    space: drop",
        "      q: quit",
        "  c: toggle color",
        "n: toggle show next",
        "h: toggle this help"
    };
```

```
    public TetrisHelp(TetrisScreen screen) {
        super(screen);
    }

    public void draw(boolean visible) {
        screen.setBold();
        screen.setFg(color);
        for (int i = 0; i < text.length; i++) {
            String s = text[i];
            if (! visible) {
                s = new String(new char[s.length()]).replace("\0", " ");
            }
            screen.xyprint(Tetris.HELP_X, Tetris.HELP_Y + i, s);
        }
        screen.resetColors();
    }
}

class TetrisScore {
    private int score = 0;
    private int level = 1;
    private int linesCompleted = 0;
    private TetrisScreen screen = null;

    TetrisScore(TetrisScreen screen) {
        this.screen = screen;
    }

    public void update(int completeLines) {
        linesCompleted += completeLines;
        score += (completeLines * completeLines);
        if (score > Tetris.LEVEL_UP * level) {
            level += 1;
            TetrisTicker.decreaseDelay();
        }
        show();
    }

    public void show() {
        screen.setBold();
        screen.setFg(Tetris.SCORE_COLOR);
        screen.xyprint(Tetris.SCORE_X, Tetris.SCORE_Y,     "Lines completed: " + linesCompleted);
        screen.xyprint(Tetris.SCORE_X, Tetris.SCORE_Y + 1, "Level:           " + level);
        screen.xyprint(Tetris.SCORE_X, Tetris.SCORE_Y + 2, "Score:           " + score);
        screen.resetColors();
    }
}

class TetrisPlayField {
    private TetrisScreen screen = null;
    private List<List<TetrisColor>> cells = new ArrayList<List<TetrisColor>>();

    public TetrisPlayField(TetrisScreen screen) {
        this.screen = screen;
        for (int i = 0; i < Tetris.PLAYFIELD_H; i++) {
            cells.add(getEmptyRow());
        }
    }

    public List<TetrisColor> getEmptyRow() {
        List<TetrisColor> row = new ArrayList<TetrisColor>();
        for (int i = 0; i < Tetris.PLAYFIELD_W; i++) {
            row.add(null);
        }
        return row;
```

```
    }

    public void show() {
        for (int y = 0; y < cells.size(); y++ ) {
            List<TetrisColor> row = cells.get(y);
            screen.xyprint(Tetris.PLAYFIELD_X, Tetris.PLAYFIELD_Y + y, "");
            for (TetrisColor cell : row) {
                if (cell == null) {
                    screen.print(Tetris.PLAYFIELD_EMPTY_CELL);
                } else {
                    screen.setFg(cell);
                    screen.setBg(cell);
                    screen.print(Tetris.FILLED_CELL);
                    screen.resetColors();
                }
            }
        }
    }

    public void flattenPiece(TetrisPiece piece) {
        for (int[] cell : piece.getCells(null)) {
            int x = cell[0];
            int y = cell[1];
            cells.get(y).set(x, piece.color);
        }
    }

    public int processCompleteLines() {
        List<List<TetrisColor>> newCells = new ArrayList<List<TetrisColor>>();
        for (List<TetrisColor> row : cells) {
            if (row.indexOf(null) != -1) {
                newCells.add(row);
            }
        }
        int completeLines = Tetris.PLAYFIELD_H - newCells.size();
        for (int i = 0; i < completeLines; i++) {
            newCells.add(0, getEmptyRow());
        }
        cells = newCells;
        return completeLines;
    }

    public void drawBorder() {
        screen.setBold();
        screen.setFg(Tetris.BORDER_COLOR);
        for (int y = 0; y < Tetris.PLAYFIELD_H; y++) {
            // 2 because border is 2 characters thick
            screen.xyprint(Tetris.PLAYFIELD_X - 2, y + Tetris.PLAYFIELD_Y, "<|");
            // 2 because each cell on play field is 2 characters wide
            screen.xyprint(Tetris.PLAYFIELD_X + Tetris.PLAYFIELD_W * 2, y + Tetris.PLAYFIELD_Y, "|>");
        }

        screen.xyprint(Tetris.PLAYFIELD_X, Tetris.PLAYFIELD_Y + Tetris.PLAYFIELD_H, new String(new
char[Tetris.PLAYFIELD_W]).replace("\0", "=="));
        screen.xyprint(Tetris.PLAYFIELD_X, Tetris.PLAYFIELD_Y + Tetris.PLAYFIELD_H + 1, new
String(new char[Tetris.PLAYFIELD_W]).replace("\0", "\\/"));
        screen.resetColors();
    }

    public boolean positionOk(TetrisPiece piece, int[] position) {
        for (int[] cell : piece.getCells(position)) {
            int x = cell[0];
            int y = cell[1];
            if (x < 0 || x >= Tetris.PLAYFIELD_W || y < 0 || y >= Tetris.PLAYFIELD_H ||
cells.get(y).get(x) != null) {
```

```
                return false;
            }
        }
        return true;
    }
}

class TetrisPiece extends TetrisScreenItem {
    public TetrisColor color = null;
    public String emptyCell = Tetris.NEXT_EMPTY_CELL;
    public int[] origin = {0, 0};

    private int symmetry = 0;
    private int[] position = {0, 0, 0};
    private int[] data = null;
    // 0123
    // 4567
    // 89ab
    // cdef
    private static int[][] pieceData = {
        {0x1256}, // square
        {0x159d, 0x4567}, // line
        {0x4512, 0x0459}, // s
        {0x0156, 0x1548}, // z
        {0x159a, 0x8456, 0x0159, 0x2654}, // l
        {0x1598, 0x0456, 0x2159, 0xa654}, // inverted l
        {0x1456, 0x1596, 0x4569, 0x4159}  // t
    };

    public TetrisPiece(TetrisScreen screen, int[] origin, boolean visible) {
        super(screen);
        this.origin = origin;
        this.visible = visible;
        color = TetrisColor.getRandomColor();
        data = pieceData[Tetris.RANDOM.nextInt(pieceData.length)];
        symmetry = data.length;
        position = new int[]{0, 0, Tetris.RANDOM.nextInt(symmetry)};
    }

    public int[][] getCells(int[] newPosition) {
        int x = position[0];
        int y = position[1];
        int z = position[2];
        if (newPosition != null) {
            x = newPosition[0];
            y = newPosition[1];
            z = newPosition[2];
        }
        int currentData = data[z];
        int result[][] = {{0, 0}, {0, 0}, {0, 0}, {0, 0}};
        for (int i = 0; i < 4; i++) {
            result[i][0] = x + ((currentData >> 4 * i) & 3);
            result[i][1] = y + ((currentData >> 4 * i + 2) & 3);
        }
        return result;
    }

    public void draw(boolean visible) {
        if (visible) {
            screen.setFg(color);
            screen.setBg(color);
        }
        int ox = origin[0];
        int oy = origin[1];
        for (int[] cell : getCells(null)) {
```

```
            int x = cell[0];
            int y = cell[1];
            screen.xyprint(ox + x * 2, oy + y, visible ? Tetris.FILLED_CELL : emptyCell);
        }
        screen.resetColors();
    }

    public void setPosition(int[] p) {
        position = new int[]{p[0], p[1], p[2] < 0 ? position[2] : p[2]};
    }

    public int[] newPosition(int dx, int dy, int dz) {
        int x = position[0];
        int y = position[1];
        int z = position[2];
        return new int[]{x + dx, y + dy, (z + dz) % symmetry};
    }
}

class TetrisController {
    public boolean running = true;

    private TetrisScreen screen = null;
    private TetrisHelp help = null;
    private TetrisScore score = null;
    private TetrisPlayField playfield = null;
    private boolean nextPieceVisible = true;
    private TetrisPiece nextPiece = null;
    private TetrisPiece currentPiece = null;

    public TetrisController(TetrisScreen screen) {
        this.screen = screen;
        help = new TetrisHelp(screen);
        score = new TetrisScore(screen);
        playfield = new TetrisPlayField(screen);
        getNextPiece();
        getCurrentPiece();
        redrawScreen();
        screen.flush();
    }

    public void getCurrentPiece() {
        nextPiece.hide();
        currentPiece = nextPiece;
        currentPiece.setPosition(new int[]{(Tetris.PLAYFIELD_W - 4) / 2, 0, -1});
        if (! playfield.positionOk(currentPiece, null)) {
            cmdQuit();
            return;
        }
        currentPiece.visible = true;
        currentPiece.emptyCell = Tetris.PLAYFIELD_EMPTY_CELL;
        currentPiece.origin = new int[]{Tetris.PLAYFIELD_X, Tetris.PLAYFIELD_Y};
        currentPiece.show();
        getNextPiece();
    }

    public void getNextPiece() {
        nextPiece = new TetrisPiece(screen, new int[]{Tetris.NEXT_X, Tetris.NEXT_Y},
nextPieceVisible);
        nextPiece.show();
    }

    public void cmdToggleColor() {
        screen.toggleColor();
        redrawScreen();
```

```
}

public void cmdToggleNext() {
    nextPieceVisible ^= true;
    nextPiece.toggle();
}

public void cmdToggleHelp() {
    help.toggle();
}

public void cmdQuit() {
    running = false;
    screen.xyprint(Tetris.GAMEOVER_X, Tetris.GAMEOVER_Y, "Game over!");
    screen.xyprint(Tetris.GAMEOVER_X, Tetris.GAMEOVER_Y + 1, "");
    screen.showCursor();
    screen.flush();
    try {
        String[] cmd = new String[] {"/bin/sh", "-c", "stty sane </dev/tty"};
        Runtime.getRuntime().exec(cmd).waitFor();
    } catch (IOException ioe) {
    } catch (InterruptedException ie) {
    }
    System.exit(0);
}

public void cmdRotate() {
    move(0, 0, 1);
}

public void cmdLeft() {
    move(-1, 0, 0);
}

public void cmdRight() {
    move(1, 0, 0);
}

public boolean cmdDown() {
    if (move(0, 1, 0)) {
        return true;
    }
    getCurrentPiece();
    return false;
}

public void cmdDrop() {
    while (cmdDown()) {};
}

public void redrawScreen() {
    screen.clearScreen();
    screen.hideCursor();
    playfield.drawBorder();
    help.show();
    playfield.show();
    score.show();
    nextPiece.show();
    currentPiece.show();
}

public void processFallenPiece() {
    playfield.flattenPiece(currentPiece);
    int completeLines = playfield.processCompleteLines();
    if (completeLines > 0) {
```

```
            score.update(completeLines);
            playfield.show();
        }
    }

    public boolean move(int dx, int dy, int dz) {
        int[] newPosition = currentPiece.newPosition(dx, dy, dz);
        if (playfield.positionOk(currentPiece, newPosition)) {
            currentPiece.hide();
            currentPiece.setPosition(newPosition);
            currentPiece.show();
            return true;
        }
        if (dy == 0) {
            return true;
        }
        processFallenPiece();
        return false;
    }

    public void screenFlush() {
        screen.flush();
    }
}
```