МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ

ЛУГАНСКОЙ НАРОДНОЙ РЕСПУБЛИКИ

ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

ЛУГАНСКОЙ НАРОДНОЙ РЕСПУБЛИКИ

«ЛУГАНСКИЙ НАЦИОНАЛЬНЫЙ УНИВЕРСИТЕТ

ИМЕНИ ТАРАСА ШЕВЧЕНКО»

Институт физики, математики и информационных технологий

(полное название института / факультета)

Кафедра информационных технологий и систем

(полное название кафедры)

**КУРСОВАЯ РАБОТА**

по «Программированию для платформы JAVA»

(название дисциплины)

на тему: «Создание 2D игры «Ping-pong» на языке программирования JAVA»

Студентки 3 курса, группы 3-ПИ

специальности «Программная инженерия»

(код, название без кавычек)

форма освоения ООП очная

(очная, заочная)

Нехаевой Дианы Владимировны

(фамилия, имя, отчество)

Руководитель:

старший преподаватель Шкандыбин Ю.А.

(должность, ученое звание, научная степень, фамилия и инициалы)

Национальная шкала \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Количество баллов: \_\_\_\_\_\_ Оценка: ECTS \_\_\_\_

Члены комиссии:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (фамилия и инициалы)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (фамилия и инициалы)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (фамилия и инициалы)

Луганск – 2019 год

**Содержание**

[ВВЕДЕНИЕ 3](#_Toc26953152)

[1. СРЕДА РАЗРАБОТКИ ПРОГРАММЫ 4](#_Toc26953153)

[1.1 Язык программирования JAVA 4](#_Toc26953154)

[1.2 Архитектура платформы Eclipse 6](#_Toc26953155)

[1.3 Объектно-ориентированный подход 8](#_Toc26953156)

[2. СТРУКТУРА ПРОЕКТА 10](#_Toc26953157)

[2.1 История создания игры «ping-pong». 10](#_Toc26953158)

[2.2 Игровой процесс 10](#_Toc26953159)

[2.3 Основные функции программы и их реализация 11](#_Toc26953160)

[2.4 Функции интерфейса 11](#_Toc26953161)

[2.5 Описание алгоритма решения задачи 12](#_Toc26953162)

[ЗАКЛЮЧЕНИЕ 14](#_Toc26953163)

[СПИСОК ЛИТЕРАТУРЫ 15](#_Toc26953164)

[ПРИЛОЖЕНИЕ А 16](#_Toc26953165)

ВВЕДЕНИЕ

В данной курсовой работе реализуется игра «Ping-pong». Для написания кода будет использоваться язык Java.

Целью работы является разработка игры «Ping-pong». Эта игра относится к логическим играм, которые помогают развить реакцию.

Задачей курсовой работы является изучение объектно-ориентированного языка программирования Java, рассмотрение игры пинг-понг и создание программного кода на языке Java.

Смысл игры заключается в стремлении выиграть у противника. Для того, чтобы игровое поле не отвлекало, была использована не яркая графика. Интерфейс понятен и удобен в использовании. Программа должна реагировать при нажатии на клавиши.

1. СРЕДА РАЗРАБОТКИ ПРОГРАММЫ

1.1 Язык программирования JAVA

Java является объектно-ориентированным языком программирования. Если сравнивать в этом смысле Java и C++, то между ними есть существенные различия. C++ тоже является объектно-ориентированным языком. Но он является расширением C, который не является объектно-ориентированным. Из этого, в частности следует, что на C++ можно писать программы как в объектно-ориентированном стиле, так и в обычном. Т.е. на C++ можно программировать, не зная и не понимая объектно-ориентированного подхода. Можно даже смешивать оба подхода в едином продукте или пытаться программировать в объектно-ориентированном стиле, а тогда, когда это не получается, скатываться к традиционному программированию.

В Java так нельзя. В ней нет средств, позволяющих писать не объектно-ориентированные программы.

Программы, созданные на языке программирования Java, подразделяются по своему назначению на две группы.

К первой группе относятся приложения Java, предназначенные для локальной работы под управлением интерпретатора (виртуальной машины) Java.

Вторую группу программ называют апплетами (aplets). Апплеты представляют собой небольшие специальные программы, находящиеся на удаленном компьютере в сети, с которым пользователи соединяются с помощью браузера. Апплеты загружаются в браузер пользователя и интерпретируются виртуальной машиной Java, встроенной практически во все современные браузеры.

Приложения, относящиеся к первой группе, представляют собой обычные локальные приложения. Поскольку выполняются интерпретатором и не содержат машинного кода, то их производительность заметно ниже, чем у обычных компилируемых программ (С++, Delphi).

Апплеты Java можно встраивать в документы HTML и помещать на Web-сервер. Использование в интернет-страницах Java-апплетов придает динамический и интерактивный характер поведению последних. Апплеты берут на себя сложную локальную обработку данных, полученных от Web-сервера или от локального пользователя. Для более быстрого выполнения апплетов в браузере применяется особый способ компиляции – Just-In-Time compilation, что позволяет увеличить скорость выполнения апплета в несколько раз.

Для разработки программ на языке Java нам потребуется специальное программное обеспечение. Самые новые версии системного программного обеспечения, необходимого для поддержки, можно загрузить с сайта компании Sun (http://java.sun.com/): JRE, JDK. Первое приложение JRE – это программа для запуска и исполнения программ (среда выполнения Java) Java Runtime Environment (JRE).

Для разработки программ также требуется комплект разработки программного обеспечения – JDK (Java Development Kit). Он содержит компилятор, стандартные библиотеки и т.п.

В настоящее время имеется три Java-платформы: 1) Java 2 Platform, Standard Edition (J2SE); 2) Java 2 Platform, Enterprise Edition (J2EE); 3) Java 2 Platform, Micro Edition (J2ME). Каждая из этих платформ предназначена для разработки определенного типа программ.

Первая платформа J2SE позволяет разрабатывать обычные (desktop) локальные приложения и апплеты.

Вторая платформа J2EE предназначена для разработки серверных приложений (сервлетов, jsp-страниц, компонентов JavaBeans).

Третья платформа (J2ME) применяется при разработке приложений для мобильных и небольших устройств (сотовых телефонов, карманных компьютеров и др.), которые имеют существенно ограниченные аппаратные ресурсы (емкость оперативной памяти, быстродействие процессора и др.).

Таким образом, в минимальный комплект для разработки программ на Java входят следующие:

* JRE – среда выполнения;
* JDK для соответствующей платформы (J2SE, J2EE, J2ME) – компилятор и библиотеки;
* среда программирования.

Системы программирования на Java состоят из нескольких частей: среда разработки, язык программирования, программный интерфейс приложений (Java API), различные библиотеки классов.

1.2 Архитектура платформы Eclipse

Java Eclipse является одной из лучших сред разработки программ.

Программы Java обычно проходят пять стадий обработки, прежде чем будут выполнены: редактирование, компиляция, загрузка, проверка байт-кода и выполнение.

На первом этапе в редакторе Eclipse вводится программа, а затем исправления в случае необходимости. Файл с программой необходимо сохранить, после окончательного редактирования он имеет стандартное расширение «имя».java. С таким же успехом можно использовать распространенные текстовые редакторы vi, kate, kwrite и emacs. В Eclipse содержится неплохой встроенный редактор и среда программирования, поэтому лучше все действия проводить там.

На следующем втором этапе необходимо откомпилировать программу, что достаточно просто сделать, выбрав команду Run главного меню. Компилятор Java должен выполнить трансляцию программы Java в байт-код, в этой форме она уже будет доступна интерпретатору Java. Если программа успешно откомпилируется, то будет создан файл с именем «имя».class. В данном файле содержатся байт-коды, которые будут интерпретироваться во время выполнения.

Третий этап называется загрузкой. Программа помещается в оперативную память и ей передается управление. Загрузчик классов в Eclipse считывает файл «имя».class и помещает его в оперативную память. Файл может загружаться как с локального диска компьютера, так с удаленного компьютера по сети. Файл «имя».class может содержать программы двух видов: обычные локальные приложения и апплеты, о которых мы говорили в начале введения. Загрузчик загружает в ОП файл и затем программа начинает выполняться интерпретатором Java. Среда Eclipse сама позаботится и вызовет интерпретатор для выполнения приложения.

Загрузчик классов также может вызываться и в том случае, когда интернет-браузер загружает интернет-страницу с встроенным Java-апплетом. Интернет-страница в формате HTML может ссылаться на Java-апплет. Когда браузер загружает такую страницу и начинает ее интерпретировать, то в момент ссылки на апплет он вызывает загрузчик классов и загружает этот апплет. Практически все современные интернет-браузеры поддерживают Java, т.е. имеют встроенный интерпретатор языка Java.

Перед тем как интерпретатор Java, вызываемый в Eclipse или встроенный в браузер, приступит к выполнению байт-кода, последний проверяется верификатором байт-кода на четвертом этапе выполнения программы. Этот этап называют проверкой байт-кода. Успешное прохождение данного этапа гарантирует то, что загруженные классы не нанесут ущерб защите и не содержат ошибок, которые могли бы вызвать сбой работы программы.

На последнем пятом этапе Eclipse интерпретирует программу, последовательно выполняя байт-коды. Программа может сразу не заработать в результате ошибок, вызванных на разных этапах выполнения, тогда следует вернуться к исходному тексту программы и внести необходимые исправления. Писать программы на Java необходимо в простом и ясном стиле. Следует избегать неправильных конструкций и способов употребления языка, тогда программы будут быстро проходить все этапы выполнения.

В настоящее время отмечается большой прогресс в развитии свободно распространяемого программного обеспечения для разработки программ на языке Java. По своему функциональному составу и набору предоставляемых сервисных услуг Eclipse практически не уступает платным программам, поэтому изучение данной среды является перспективным и отвечающим большинству требований, выдвигаемых к современному программному обеспечению, функционирующему на различных аппаратных платформах.

1.3 Объектно-ориентированный подход

Объектно-ориентированное программирование или ООП (object-oriented programming) – методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из который является реализацией определенного типа, использующая механизм пересылки сообщений и классы, организованные в иерархию наследования.

Класс является описываемой на языке терминологии (пространства имён) исходного кода моделью ещё не существующей сущности, т. е. объекта.

Объект – сущность в адресном пространстве вычислительной системы, появляющаяся при создании экземпляра класса (например, после запуска результатов компиляции исходного кода на выполнение).

Инкапсуляция – это принцип, согласно которому любой класс должен рассматриваться как чёрный ящик - пользователь класса должен видеть и использовать только интерфейсную часть класса (т.е. список декларируемых свойств и методов класса) и не вникать в его внутреннюю реализацию. Поэтому данные принято инкапсулировать в классе таким образом, чтобы доступ к ним по чтению или записи осуществлялся не напрямую, а с помощью методов. Принцип инкапсуляции (теоретически) позволяет минимизировать число связей между классами и, соответственно, упростить независимую реализацию и модификацию классов.

Сокрытие данных – неотделимая часть ООП, управляющая областями видимости. Является логическим продолжением инкапсуляции. Целью сокрытия является невозможность для пользователя узнать или испортить внутреннее состояние объекта.

Наследованием называется возможность порождать один класс от другого с сохранением всех свойств и методов класса–предка (прародителя, иногда его называют суперклассом) и добавляя, при необходимости, новые свойства и методы. Набор классов, связанных отношением наследования, называют иерархией. Наследование призвано отобразить такое свойство реального мира, как иерархичность.

Полиморфизмом называют явление, при котором функции (методу) с одним и тем же именем соответствует разный программный код (полиморфный код) в зависимости от того, объект какого класса используется при вызове данного метода. Полиморфизм обеспечивается тем, что в классе-потомке изменяют реализацию метода класса-предка с обязательным сохранением сигнатуры метода. Это обеспечивает сохранение неизменным интерфейса класса-предка и позволяет осуществить связывание имени метода в коде с разными классами из объекта какого класса осуществляется вызов, из того класса и берётся метод с данным именем. Такой механизм называется динамическим (или поздним) связыванием, в отличие от статического (раннего) связывания, осуществляемого на этапе компиляции.

2. СТРУКТУРА ПРОЕКТА

2.1 История создания игры «ping-pong».

«Ping-pong» – одна из ранних аркадных игр. Это теннисная игра с использованием простой двумерной графики, разработанная и выпущенная фирмой Atari в 1972 году. «Ping-pong» называют первой в истории коммерчески успешной видеоигрой, а с ее именем связывают появление индустрии интерактивных развлечений.

Идею пинг-понга для создания игры предложил Нолан Бушнелл своему сотруднику, программисту Аллану Алькорну. Идея пинг-понга для видеоигр в то время уже была реализована в Magnavox Odyssey, и это привело к иску против Atari.

Во время рождественского сезона 1975 года Atari выпустила домашнюю версию «Pong» эксклюзивно для розничных магазинов Sears. Это обернулось коммерческим успехом и привело к появлению ее клонов, таких как Color TV Game 6 от Nintendo, которая стала первой собственной приставкой для нее. Игра была переиздана на домашних и портативных платформах.

2.2 Игровой процесс

«Ping-pong» является простым спортивным симулятором настольного тенниса. Небольшой шар, заменяющий пинг-понговый мячик, двигается по экрану по линейной траектории. Если он ударяется о периметр игрового поля, то его траектория изменяется в зависимости от угла столкновения. Если шарик отбивается ракеткой игрока, то его движение дополнительно зависит от скорости и направления движения ракетки. Периметр игрового поля обозначен краями экрана, а мячик не может покинуть поле через верхний или нижний край. В верхней части поля отображаются очки игроков, у каждого на своей половине экрана.

Игровой процесс состоит в том, что игроки передвигают свои ракетки вертикально для защиты своих ворот. В начале каждого раунда мячик подаётся одному из игроков, и раунд продолжается до тех пор, пока один из игроков не заработает очко. Это происходит тогда, когда его оппонент не может отбить мячик.

Первоначальная версия «Ping-pong» разработана для двух игроков, более поздних версиях игры стал доступен однопользовательский режим, когда одна из ракеток управляется компьютерным игроком.

2.3 Основные функции программы и их реализация

Основными функциями программы являются:

* реализация пользовательского интерфейса, в частности меню программы
* описание предметной области в ключе объектно-ориентированного программирования
* реализация процедуры быстрого рисования объекта (мяча)
* реализация алгоритма управления платформы противника
* реализация механизма поведения мяча при столкновении с препятствиями.

2.4 Функции интерфейса

Пользовательский интерфейс – первое, с чем сталкивается пользователь при запуске программы. Первое впечатление о любой программе у пользователя складывается в первые мгновения работы с ней. И здесь дружелюбность пользовательского интерфейса чрезвычайно важна. Весь человеко-машинный интерфейс при работе с компьютером является интерфейсами операционной системы и программ, с которыми пользователь работает.

Понятие дружелюбного пользовательского интерфейса включает в себя всю совокупность удобства, информативности и интуитивной понятности пользовательского интерфейса. Косвенно, хороший дружественный интерфейс свидетельствует о хорошем уровне программы, ускоряет работу с ней.

Таким образом, важность пользовательского интерфейса ни в коем случае не стоит приуменьшать.

2.5 Описание алгоритма решения задачи

Для решения поставленной задачи необходимо разработать несколько объектов и реализовать их методы.

Базовым объектом – предком является статический объект. Статический объект характеризуется координатами x,y расположения объекта. Динамический объект является наследником от статического. Дополнительными характеристиками динамического объекта являются экранные координаты, на которые он был прорисован последний раз, а также указатель на битовую матрицу экранного изображения объекта.

Основными объектами программы являются: мяч и выбиваемый прямоугольник.

Прямоугольник – объект, наследующий все свойства статического объекта. Дополнительными характеристиками этого объекта являются ширина и высота прямоугольника, а также условная единица показывающая твердость объекта, то есть количество попаданий необходимых для его уничтожения.

Мяч – объект, наследующий все свойства динамического объекта. Дополнительные параметры – величины смещения по обоим осям координат за один такт игры и радиус мяча.

Все объекты содержат процедуру draw, рисующую изображение объекта на экране. При этом динамические объекты рисуются путем предварительного стирания себя на экране копированием себя по маске XOR, а затем уже обычным способом копируют свое изображение в новое положение на экране.

Для объединения всех объектов в одно логическое целое. Разработан класс – игровое поле, содержащее в себе указатели на все объекты игры и реализующий общее управление программой.

Общий алгоритм работы игры следующий. В начале игры запускается метод Start, инициализирующий переменные игры. Затем идет основной игровой цикл. В нем прорисовываются все визуальные объекты игры. Затем производится задержка, необходимая для соразмерения скорости игры со скоростью компьютера. Если не ставить этой задержки, то скорость игры будет слишком быстрой. Далее в цикле опрашивается клавиатура и если нажата какая-либо управляющая кнопка, то обновляется скорость и направление движения платформы игрока. Затем осуществляется один условный шаг игры путем запуска class game(Pong) объекта игрового поля. Внутри данного метода вызываются одноименные методы для кареток игрока и компьютера, а также мяча. Здесь же анализируется попадание мяча во все препятствия. При попадании производится расчет нового направления движения мяча по обеим осям координат. При необходимости производится изменение направления движения по каждой из осей за счет изменения знака величины vx или vy мяча на противоположный. После одного шага игры проверяется текущее ее состояние. Если мяч вылетел за пределы игрового поля, то увеличивается количество пропущенных мячей того из игроков в чью сторону улетел мяч. Также проверяется факт окончания игры. Игра считается оконченной, если у одного из игроков количество пропущенных мячей достигло максимальной величины, либо пользователь отказался от игры. Выход из игрового цикла осуществляется, если такой факт установлен.

ЗАКЛЮЧЕНИЕ

В ходе работы с курсовым проектом были выполнены такие задачи:

1. Рассмотреть возможности языка программирования Java;
2. Описать алгоритм проектирования программы;
3. Реализовать программу.

При написании программы был использован объектно-ориентированный язык Java. В результате была написана игра «Ping-pong».

Приложение разрабатывалось в программе Eclipce.

Код программы можно просмотреть на сайте: https://github.com/diana889/ping-pong.git

СПИСОК ЛИТЕРАТУРЫ

1. Барнет Э. Eclipse IDE Карманный справочник: Пер. с англ. — М.:КУДИЦ-ОБРАЗ, 2006. — 160 с.

2. Хабибуллин И. Ш. Самоучитель Java 2. — СПб.: БХВ-Петербург, 2007. — 720 с.

3. Шилдт Г. Полный справочник по Java. — М.: Вильямс, 2007. — 1040 с.

4. Ноутон П., Шилдт Г. Java 2. — CПб:, BНV-Санкт-Петербург, 2008. — 1072 с. Электронные издания

5. Монахов В.В. Материалы курсов, разработанных в рамках программы Sun Microsystems Teaching Grants в 2006 году — «Язык программирования Java». СПбГУ, 2006.

6. Материалы интернет-сайта http://ru.sun.com

7. Фишер Т. Java. Карманный справочник. — М.: Вильямс, 2008. — 224 с.

8. Хемраджани А. Гибкая разработка приложений на Java с помощью Spring, Hibernate и Eclipse. — М.: Вильямс, 2008. — 352 с.

9. Хабибуллин И. Ш. Самоучитель Java 2. — СПб.: БХВ-Петербург, 2007. — 720 с.

10. Шилдт Г. Swing. Руководство для начинающих. — М.: Вильямс, 2007. — 704 с.

ПРИЛОЖЕНИЕ А

Листинг программы:

**package** pong;

**import** java.awt.BasicStroke;

**import** java.awt.Color;

**import** java.awt.Font;

**import** java.awt.Graphics2D;

**import** java.awt.RenderingHints;

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**import** java.awt.event.KeyEvent;

**import** java.awt.event.KeyListener;

**import** java.util.Random;

**import** javax.swing.JFrame;

**import** javax.swing.Timer;

**public** **class** Pong **implements** ActionListener, KeyListener

{

**public** **static** Pong *pong*;

**public** **int** width = 700, height = 700;

**public** Renderer renderer;

**public** Paddle player1;

**public** Paddle player2;

**public** Ball ball;

**public** **boolean** bot = **false**, selectingDifficulty;

**public** **boolean** w, s, up, down;

**public** **int** gameStatus = 0, scoreLimit = 7, playerWon; //0 = Menu, 1 = Paused, 2 = Playing, 3 = Over

**public** **int** botDifficulty, botMoves, botCooldown = 0;

**public** Random random;

**public** JFrame jframe;

**public** Pong()

{

Timer timer = **new** Timer(20, **this**);

random = **new** Random();

jframe = **new** JFrame("Pong");

renderer = **new** Renderer();

jframe.setSize(width + 15, height + 35);

jframe.setVisible(**true**);

jframe.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);

jframe.add(renderer);

jframe.addKeyListener(**this**);

timer.start();

}

**public** **void** start()

{

gameStatus = 2;

player1 = **new** Paddle(**this**, 1);

player2 = **new** Paddle(**this**, 2);

ball = **new** Ball(**this**);

}

**public** **void** update()

{

**if** (player1.score >= scoreLimit)

{

playerWon = 1;

gameStatus = 3;

}

**if** (player2.score >= scoreLimit)

{

gameStatus = 3;

playerWon = 2;

}

**if** (w)

{

player1.move(**true**);

}

**if** (s)

{

player1.move(**false**);

}

**if** (!bot)

{

**if** (up)

{

player2.move(**true**);

}

**if** (down)

{

player2.move(**false**);

}

}

**else**

{

**if** (botCooldown > 0)

{

botCooldown--;

**if** (botCooldown == 0)

{

botMoves = 0;

}

}

**if** (botMoves < 10)

{

**if** (player2.y + player2.height / 2 < ball.y)

{

player2.move(**false**);

botMoves++;

}

**if** (player2.y + player2.height / 2 > ball.y)

{

player2.move(**true**);

botMoves++;

}

**if** (botDifficulty == 0)

{

botCooldown = 20;

}

**if** (botDifficulty == 1)

{

botCooldown = 15;

}

**if** (botDifficulty == 2)

{

botCooldown = 10;

}

}

}

ball.update(player1, player2);

}

**public** **void** render(Graphics2D g)

{

g.setColor(Color.***BLACK***);

g.fillRect(0, 0, width, height);

g.setRenderingHint(RenderingHints.***KEY\_ANTIALIASING***, RenderingHints.***VALUE\_ANTIALIAS\_ON***);

**if** (gameStatus == 0)

{

g.setColor(Color.***WHITE***);

g.setFont(**new** Font("Arial", 1, 50));

g.drawString("PONG", width / 2 - 75, 50);

**if** (!selectingDifficulty)

{

g.setFont(**new** Font("Arial", 1, 30));

g.drawString("Press Space to Play", width / 2 - 150, height / 2 - 25);

g.drawString("Press Shift to Play with Bot", width / 2 - 200, height / 2 + 25);

g.drawString("<< Score Limit: " + scoreLimit + " >>", width / 2 - 150, height / 2 + 75);

}

}

**if** (selectingDifficulty)

{

String string = botDifficulty == 0 ? "Easy" : (botDifficulty == 1 ? "Medium" : "Hard");

g.setFont(**new** Font("Arial", 1, 30));

g.drawString("<< Bot Difficulty: " + string + " >>", width / 2 - 180, height / 2 - 25);

g.drawString("Press Space to Play", width / 2 - 150, height / 2 + 25);

}

**if** (gameStatus == 1)

{

g.setColor(Color.***WHITE***);

g.setFont(**new** Font("Arial", 1, 50));

g.drawString("PAUSED", width / 2 - 103, height / 2 - 25);

}

**if** (gameStatus == 1 || gameStatus == 2)

{

g.setColor(Color.***WHITE***);

g.setStroke(**new** BasicStroke(5f));

g.drawLine(width / 2, 0, width / 2, height);

g.setStroke(**new** BasicStroke(2f));

g.drawOval(width / 2 - 150, height / 2 - 150, 300, 300);

g.setFont(**new** Font("Arial", 1, 50));

g.drawString(String.*valueOf*(player1.score), width / 2 - 90, 50);

g.drawString(String.*valueOf*(player2.score), width / 2 + 65, 50);

player1.render(g);

player2.render(g);

ball.render(g);

}

**if** (gameStatus == 3)

{

g.setColor(Color.***WHITE***);

g.setFont(**new** Font("Arial", 1, 50));

g.drawString("PONG", width / 2 - 75, 50);

**if** (bot && playerWon == 2)

{

g.drawString("The Bot Wins!", width / 2 - 170, 200);

}

**else**

{

g.drawString("Player " + playerWon + " Wins!", width / 2 - 165, 200);

}

g.setFont(**new** Font("Arial", 1, 30));

g.drawString("Press Space to Play Again", width / 2 - 185, height / 2 - 25);

g.drawString("Press ESC for Menu", width / 2 - 140, height / 2 + 25);

}

}

@Override

**public** **void** actionPerformed(ActionEvent e)

{

**if** (gameStatus == 2)

{

update();

}

renderer.repaint();

}

**public** **static** **void** main(String[] args)

{

*pong* = **new** Pong();

}

@Override

**public** **void** keyPressed(KeyEvent e)

{

**int** id = e.getKeyCode();

**if** (id == KeyEvent.***VK\_W***)

{

w = **true**;

}

**else** **if** (id == KeyEvent.***VK\_S***)

{

s = **true**;

}

**else** **if** (id == KeyEvent.***VK\_UP***)

{

up = **true**;

}

**else** **if** (id == KeyEvent.***VK\_DOWN***)

{

down = **true**;

}

**else** **if** (id == KeyEvent.***VK\_RIGHT***)

{

**if** (selectingDifficulty)

{

**if** (botDifficulty < 2)

{

botDifficulty++;

}

**else**

{

botDifficulty = 0;

}

}

**else** **if** (gameStatus == 0)

{

scoreLimit++;

}

}

**else** **if** (id == KeyEvent.***VK\_LEFT***)

{

**if** (selectingDifficulty)

{

**if** (botDifficulty > 0)

{

botDifficulty--;

}

**else**

{

botDifficulty = 2;

}

}

**else** **if** (gameStatus == 0 && scoreLimit > 1)

{

scoreLimit--;

}

}

**else** **if** (id == KeyEvent.***VK\_ESCAPE*** && (gameStatus == 2 || gameStatus == 3))

{

gameStatus = 0;

}

**else** **if** (id == KeyEvent.***VK\_SHIFT*** && gameStatus == 0)

{

bot = **true**;

selectingDifficulty = **true**;

}

**else** **if** (id == KeyEvent.***VK\_SPACE***)

{

**if** (gameStatus == 0 || gameStatus == 3)

{

**if** (!selectingDifficulty)

{

bot = **false**;

}

**else**

{

selectingDifficulty = **false**;

}

start();

}

**else** **if** (gameStatus == 1)

{

gameStatus = 2;

}

**else** **if** (gameStatus == 2)

{

gameStatus = 1;

}

}

}

@Override

**public** **void** keyReleased(KeyEvent e)

{

**int** id = e.getKeyCode();

**if** (id == KeyEvent.***VK\_W***)

{

w = **false**;

}

**else** **if** (id == KeyEvent.***VK\_S***)

{

s = **false**;

}

**else** **if** (id == KeyEvent.***VK\_UP***)

{

up = **false**;

}

**else** **if** (id == KeyEvent.***VK\_DOWN***)

{

down = **false**;

}

}

@Override

**public** **void** keyTyped(KeyEvent e)

{

}

}

**package** pong;

**import** java.awt.Graphics;

**import** java.awt.Graphics2D;

**import** javax.swing.JPanel;

**public** **class** Renderer **extends** JPanel

{

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

@Override

**protected** **void** paintComponent(Graphics g)

{

**super**.paintComponent(g);

Pong.*pong*.render((Graphics2D) g);

}

}

**package** pong;

**import** java.awt.Color;

**import** java.awt.Graphics;

**public** **class** Paddle

{

**public** **int** paddleNumber;

**public** **int** x, y, width = 50, height = 250;

**public** **int** score;

**public** Paddle(Pong pong, **int** paddleNumber)

{

**this**.paddleNumber = paddleNumber;

**if** (paddleNumber == 1)

{

**this**.x = 0;

}

**if** (paddleNumber == 2)

{

**this**.x = pong.width - width;

}

**this**.y = pong.height / 2 - **this**.height / 2;

}

**public** **void** render(Graphics g)

{

g.setColor(Color.***WHITE***);

g.fillRect(x, y, width, height);

}

**public** **void** move(**boolean** up)

{

**int** speed = 15;

**if** (up)

{

**if** (y - speed > 0)

{

y -= speed;

}

**else**

{

y = 0;

}

}

**else**

{

**if** (y + height + speed < Pong.*pong*.height)

{

y += speed;

}

**else**

{

y = Pong.*pong*.height - height;

}

}

}

} **package** pong;

**import** java.awt.Color;

**import** java.awt.Graphics;

**import** java.util.Random;

**public** **class** Ball

{

**public** **int** x, y, width = 25, height = 25;

**public** **int** motionX, motionY;

**public** Random random;

**private** Pong pong;

**public** **int** amountOfHits;

**public** Ball(Pong pong)

{

**this**.pong = pong;

**this**.random = **new** Random();

spawn();

}

**public** **void** update(Paddle paddle1, Paddle paddle2)

{

**int** speed = 5;

**this**.x += motionX \* speed;

**this**.y += motionY \* speed;

**if** (**this**.y + height - motionY > pong.height || **this**.y + motionY < 0)

{

**if** (**this**.motionY < 0)

{

**this**.y = 0;

**this**.motionY = random.nextInt(4);

**if** (motionY == 0)

{

motionY = 1;

}

}

**else**

{

**this**.motionY = -random.nextInt(4);

**this**.y = pong.height - height;

**if** (motionY == 0)

{

motionY = -1;

}

}

}

**if** (checkCollision(paddle1) == 1)

{

**this**.motionX = 1 + (amountOfHits / 5);

**this**.motionY = -2 + random.nextInt(4);

**if** (motionY == 0)

{

motionY = 1;

}

amountOfHits++;

}

**else** **if** (checkCollision(paddle2) == 1)

{

**this**.motionX = -1 - (amountOfHits / 5);

**this**.motionY = -2 + random.nextInt(4);

**if** (motionY == 0)

{

motionY = 1;

}

amountOfHits++;

}

**if** (checkCollision(paddle1) == 2)

{

paddle2.score++;

spawn();

}

**else** **if** (checkCollision(paddle2) == 2)

{

paddle1.score++;

spawn();

}

}

**public** **void** spawn()

{

**this**.amountOfHits = 0;

**this**.x = pong.width / 2 - **this**.width / 2;

**this**.y = pong.height / 2 - **this**.height / 2;

**this**.motionY = -2 + random.nextInt(4);

**if** (motionY == 0)

{

motionY = 1;

}

**if** (random.nextBoolean())

{

motionX = 1;

}

**else**

{

motionX = -1;

}

}

**public** **int** checkCollision(Paddle paddle)

{

**if** (**this**.x < paddle.x + paddle.width && **this**.x + width > paddle.x && **this**.y < paddle.y + paddle.height && **this**.y + height > paddle.y)

{

**return** 1; //bounce

}

**else** **if** ((paddle.x > x && paddle.paddleNumber == 1) || (paddle.x < x - width && paddle.paddleNumber == 2))

{

**return** 2; //score

}

**return** 0; //nothing

}

**public** **void** render(Graphics g)

{

g.setColor(Color.***WHITE***);

g.fillOval(x, y, width, height);

}

}