МИНИСТЕРСТВО ОБРАЗОВАНИЯ КАЛИНИНГРАДСКОЙ ОБЛАСТИ

ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ УЧРЕЖДЕНИЕ

КАЛИНИНГРАДСКОЙ ОБЛАСТИ

ПРОФЕССИОНАЛЬНАЯ ОБРАЗОВАТЕЛЬНАЯ ОРГАНИЗАЦИЯ

«КОЛЛЕДЖ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И СТРОИТЕЛЬСТВА»

(ГБУ КО ПОО «КИТиС»)

|  |  |  |
| --- | --- | --- |
| Курсовой проект  допущен к защите  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (дата)  Зам.директора по УМР  \_\_\_\_\_\_\_\_\_\_\_\_\_Павленко Г.Я.  (подпись) |  | Курсовой проект  защищен с оценкой  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (оценка)  Руководитель работы  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись, дата) |

Пояснительная записка к курсовому проекту

по дисциплине: МДК 01.01 Разработка программных модулей

Тема: Разработка и программная реализация модели персонажа игры

Специальность 09.02.07 «Информационные системы и программирование»

|  |  |
| --- | --- |
| Выполнил: | студент 3 курса,  группы ИСп 20-1  Коржилов Михаил Владимирович  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) |
| Руководитель: | Большакова-Стрекалова Анна Викторовна, преподаватель ГБУ КО ПОО КИТиС  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) |

Калининград

2022

Аннотация

Тема курсового проекта: «Разработка и программная реализация модели персонажа игры»

Курсовая работа состоит из введения, описания предметной области, описания разработки приложения, тестирования и установки приложения, заключения. К ней прилагаются списки литературы и приложения.

Главный тезис курсовой работы: Компьютерные игры с момента создания и до настоящего времени сохраняют свою популярность, поэтому разработка новых компьютерных игр происходит повсеместно. Считается, что игры, в которых геймер может руководить персонажем, более популярны, чем игры без персонажей, поэтому создание персонажа, который может быть встроен в компьютерную игру, является актуальным.

Ссылка на GitHub [7] - https://github.com/1mishakor/Korzhilov\_Kursovaya.

Содержание

[Введение 4](#_Toc133889920)

[1 Описание предметной области 5](#_Toc133889921)

[1.1 Аналоги разрабатываемого приложения 5](#_Toc133889922)

[1.2 Техническое задание 6](#_Toc133889923)

[1.3 Описание структуры приложения 7](#_Toc133889924)

[2 Описание разработки приложения 9](#_Toc133889925)

[2.1 Обоснование средств разработки 9](#_Toc133889926)

[2.2 Разработка интерфейса 9](#_Toc133889927)

[2.3 Разработка логики работы приложения. Схема взаимодействия компонентов проекта 13](#_Toc133889928)

[2.4 Описание переменных, компонентов, классов и подпрограмм 14](#_Toc133889929)

[3 Тестирование и установка приложения 27](#_Toc133889930)

[Заключение 38](#_Toc133889931)

[Список использованной литературы 39](#_Toc133889932)

[Приложение. 40](#_Toc133889933)

# Введение

Данный курсовой проект посвящён разработке программного модуля персонажа игры и его программной реализации. Для его программной реализации изначально следует определить актуальность данного проекта, его цели и задачи.

Актуальность проекта обусловлена возможностью внедрения модели персонажа в любую игру, с возможностью указания спрайтов и его расположения, а также возможностью его настраивать.

Основной целью данного проекта является создание персонажа компьютерной игры с использованием классов и возможностей java. Результаты выполнения данного курсового проекта планируются для использования в дипломной работе.

Для реализации поставленной цели необходимо выполнение следующих задач:

1. Разработка класса персонажа игры, создание структуры классов для его программной реализации.
2. Разработка спрайтов и картинок для использования в программной реализации персонажа игры.
3. Тестирование реализованного персонажа, устранение возникающих неполадок, например, связанных с расширением экрана, адаптация функционала персонажа к разным размерам дисплея, обеспечение совместимости с различными версиями Windows.

# Описание предметной области

* 1. Аналоги разрабатываемого приложения

Одним из аналогов и в тоже время вдохновителем программной реализации настоящего приложения являются игры «Kingdom: New Lands»[8], показанная на рисунке 1, и «Until we die»[9], показанная на рисунке 2,   
в большей степени рисовка и концепт 2D графики, а также для разработки программной реализации.



Рисунок 1 – «Kingdom: New Lands»



Рисунок 2 – «Until we die»

Достоинства и недостатки «Kingdom: New Lands»

Достоинства «Kingdom: New Lands»:

* приятная глазу 2D графика;
* атмосфера;
* незамысловатый сюжет;
* приятная музыка;

Недостатки «Kingdom: New Lands»:

* невозможность управления своей армией (невозможно перемещения группы персонажей с одной стены на другую).

Достоинства и недостатки «Until we die»

Достоинства «Until we die»:

* приятная глазу 2D графика;
* атмосфера;
* приятная музыка, которая может задать тон игре;
* возможность управления ботами.

Недостатки «Until we die»:

* неудобное в некоторых моментах управление;
* высокая сложность в прохождении.
  1. Техническое задание

1. Общие сведения.

Следует разработать класс персонажа игры и его дальнейшую программную реализацию, для компьютера с операционной системой Windows.

Персонаж игры должен реагировать на нажатие клавиш «A» или «стрелка влево» для перемещения персонажа в левую сторону экрана и «D» или «стрелка вправо» для перемещения персонажа в правую сторону экрана.

Далее для класса Game, который является программной реализацией класса Person, необходимо создать ещё два класса: ObjectGame и Fon. Далее следует нарисовать спрайты персонажа игра, движение налево и направо и то, в какие стороны смотрит персонаж, когда была нажата последняя клавиша, отвечающая за его движение. Следует также изобразить спрайты развевающегося огня и дерева от ветра, нарисовать фоновые изображения и облака.

В программной реализации при движении персонажа, когда персонаж переходит определённую черту, он перестает двигаться (анимация продолжает движение), в то время как фон начинает двигаться, пока фон либо дойдёт до левого конца экрана, либо до правого конца экрана, после чего персонаж перестает «упираться» в эту границу и в дальнейшем персонаж «упрётся» в границы экрана. Всё вышеперечисленное должно создать эффект камеры, следящей за персонажем до границ карты игры.

Данная игра должна быть доступной для компьютера с разным разрешением экрана.

1. Требования к системе.

Вес приложения с учётом файлов ресурсов составляет 176 мегабайт, в архивированном виде весит 64 мегабайт. Минимальная версия Windows версии Windows 95. Приложение доступно для разных разрешений экрана. Доступна только для операционной системы Windows.

1. Представление программы пользователю.

Программа должна быть доступной для многих компьютеров, где нет Java, поэтому программа должна быть в формате exe.

Внешний вид программы должен быть реализован в пиксель арт формате. В программе должны быть изображены: горы, трава с кустами, одиноко стоящее дерево и костёр, который должен гореть возле дерева. При запуске игры игрок должен находиться между деревом и костром. Время суток игры полдень, время суток не меняется.

* 1. Описание структуры приложения

Программная реализация персонажа игры имеет структуру из 4 классов написанные на Java:

* Game.java – класс программной реализации классов представленных ниже, является самой условной игрой.
* ObjectGame.java – класс, который создаёт объекты и имеет возможность их анимировать.
* Fon.java – класс, создающий фон.
* Person.java – класс, реализующий персонаж, анимацию его движений влево и вправо, а также, во время остановки, поворота фигуры в сторону, в которую он двигался.

# Описание разработки приложения

## Обоснование средств разработки

Для создания и программной реализации персонажа игры будет использована среда разработки IntelliJ IDEA [4] для языка программирования Java [3], а для создания спрайтов и прочих картинок будет использоваться PyxelEdit [2].

Язык Java был выбран из-за того, что этот язык хорошо подходит для данного проекта, в данном языке очень хорошо и понятно реализуется объектно-ориентированное программирование (ООП), что подходит для реализации и разработки персонажа игры.

Среда разработки IntelliJ IDEA – это одна из популярных сред программирования для Java. Данная среда разработки была выбрана из-за своего удобства и простоты использования, а также благодаря понятному интерфейсу. Была выбрана редакция Community Edition, так как является бесплатной, конечно, данная редакция не имеет того функционала, который есть в редакции Ultimate, но этого функционала достаточно для разработки программы.

PyxelEdit – это среда для рисования пиксель арт картин. Данная среда выбрана из-за того, что является бесплатной и удобной в рисовании, а также имеет простой и понятный интерфейс. Кроме того, в ней имеется возможность настраивать количество пикселей, а также есть возможность делить поле рисования на несколько частей, что подходит для рисования спрайтов.

## Разработка интерфейса

В начале следует нарисовать персонажа, его перемещение влево и вправо, к примеру, будем использовать персонажа, который изображён на рисунке 3.

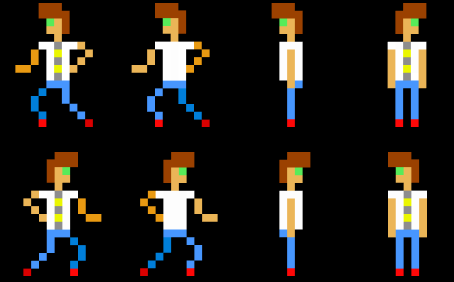


Рисунок 3 - Спрайты персонажа

Далее нужно нарисовать спрайты других объектов и фоны игры. Фонов будет три:

* Небо с солнцем;
* Горы;
* Трава с кустами.

Изображены данные фоновые изображения на рисунке 4 в редакторе PyxelEdit.

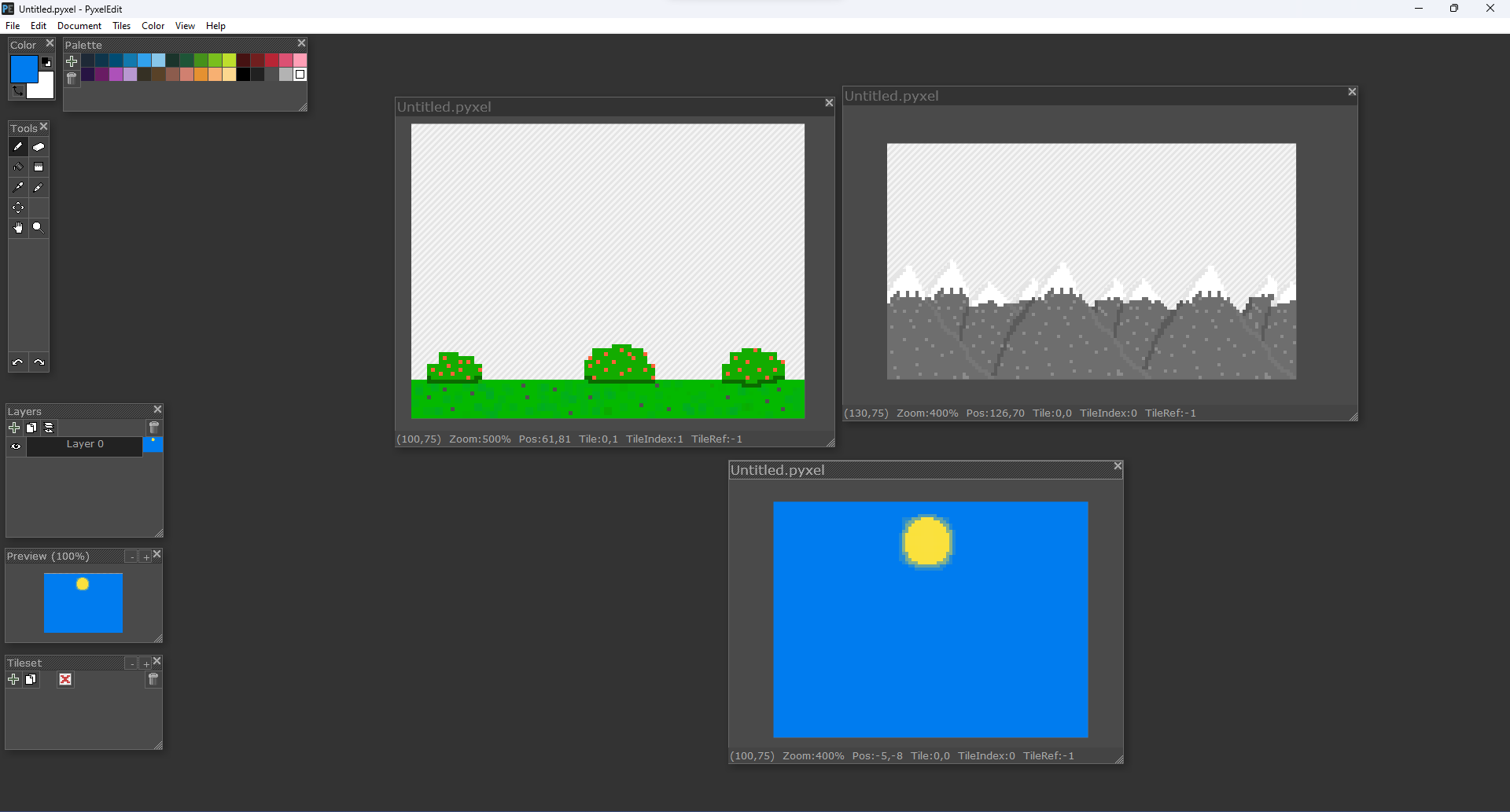


Рисунок 4 - Фоны приложения

При наложении друг на друга должно получится так, как показано на рисунке 5.



Рисунок 5 – Общий фон игры

Далее заметим, что всё выглядит неплохо, но чего-то не хватает. Для этого нужно нарисовать объекты игры. Изобразим простую анимацию для 2 объектов: дерева и костра.



Рисунок 6 – Добавлены дерево и костёр

Как видно из рисунка 6, общий фон преобразился, но чего-то всё еще не хватает, поэтому добавим облака, что позволит придать игре динамики.



Рисунок 7 – Добавление облаков

Теперь, как видно из рисунка 7, фон почти готов, осталось добавить в эту игру персонаж.



Рисунок 8 – Добавление персонажа игры

Теперь, как видно из рисунка 8, всё готово, игра готова к использованию.

## Разработка логики работы приложения. Схема взаимодействия компонентов проекта

Как видно из схемы, изображённой на рисунке 9, имеется три независимых класса: Person, ObjectGame, Fon; которые используются в классе Game.

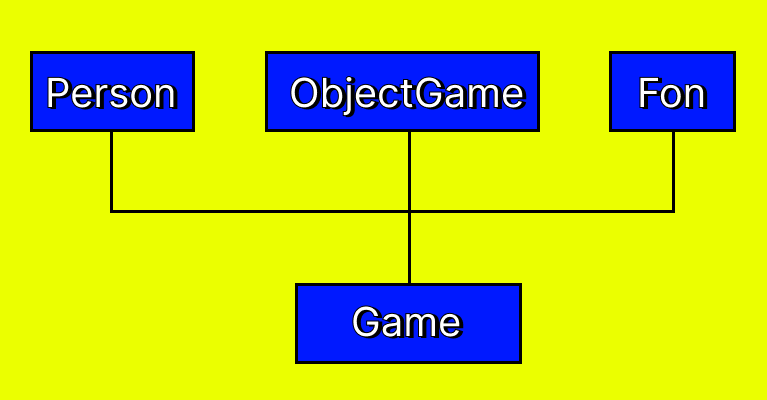


Рисунок 9 – Схема классов игры

Класс Person выполняет задачу управления и отрисовки персонажа, как понятно из сказанного выше, этот класс реагирует на нажатие кнопок клавиатуры, и в зависимости от нажатой кнопки проигрывается разная анимация.

Класс ObjectGame выполняет задачу отрисовки разных объектов программы, данный класс отыгрывает анимацию объекта не зависимо от пользователя.

Класс Fon выполняет задачу фонового изображения, не имеет анимации не реагирует на нажатие клавиш клавиатуры.

Класс Game выполняет задачу программной реализации всех трёх вышеперечисленных классов.

Логика перемещения персонажа в классе Game заключается в том, что движение персонажа игры ограничено препятствиями двух типов, назовём эти препятствия постоянными и переменными. Постоянные препятствия – это края экрана, через которые персонаж не может двигаться, как бы «упираясь» в препятствие границ экрана. Переменными препятствиями являются те, которые на время ограничивают движение персонажа игры. Переменные препятствия располагаются от центра экрана на сто пикселей в левую сторону и в правую сторону, логика работы данных препятствий такова: если левый конец по x подножия персонажа (поверхность под ногами персонажа) равен нулю, и его длина по x не равна ширине экрана, то персонаж передвигаясь, уткнётся в левое переменное препятствие, которое располагается от центра на сто пикселей вправо, тем самым остановившись, и теперь после этого подножие персонажа и объекты, которые находятся в игре, начинают двигаться в противоположном движении от движения персонажа, тем самым создаётся ощущение «передвижения камеры» за персонажем. Когда подножие персонажа дойдёт до конца то препятствие исчезнет и далее за ним будет граница экрана, через которую персонаж не сможет пройти. Тот же принцип работает и в обратную сторону. Если персонаж будет в центре «карты», то будут активны два препятствия сразу, и упираясь в одно из них персонаж будет переставать перемещаться, а фон, объекты и его подножие будет двигаться в противоположном направлении от персонажа. Следует заметить, что анимация не должна останавливаться, иначе эффект будет разрушен. Для справки, переменные препятствие называется скроллинг [5].

### Описание переменных, компонентов, классов и подпрограмм

В таблице 1 показаны классы программы и то, какие классы и интерфейсы они наследуют.

Таблица 1 – Классы программы

|  |  |  |  |
| --- | --- | --- | --- |
| Классы | Наследует класса (extends) | Наследует интерфейса (implements) | Назначение |
|  |  |  |  |

Продолжение таблицы 1

|  |  |  |  |
| --- | --- | --- | --- |
| Person | JComponent | KeyListener, ActionListener | Отображает спрайты персонажа и отвечает за перемещение персонажа |
| ObjectGame | JComponent | ActionListener | Отображает спрайты объектов |
| Fon | - | - | Отображает фоновые изображения |
| Game | JComponent | KeyListener, ActionListener | Программная реализация персонажа игры, отображает все элементы задействованных классов в данной программе |

Переменные класса Person отображены в таблице 2.

Таблица 2 – Переменные класса Person

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование переменной | Тип данных | Модификатор | Назначение |
| bool | boolean | public | Отвечает за поворот персонажа после движения в какую-либо из сторон. |
| i | int | private | Отвечает за 1 массив из 3, который нужно прокручивать при нажатии на определённую клавишу. |
| j | int | private | Отвечает за прокручивание массива выбранный i. |

Продолжение таблицы 2

|  |  |  |  |
| --- | --- | --- | --- |
| sprite | ImageIcon [] [] | - | Отвечает за хранение анимации. Это двумерный массив. В него входят i и j (sprite[i][j]). |
| grass\_sp | Image | - | Служит для отображения одного спрайта из массивов sprite. |
| x | int | public | Отвечает за положение персонажа по оси X. |
| y | int | public | Отвечает за положение персонажа по оси Y. |
| h | int | public | Отвечает за высоту персонажа. |
| w | int | public | Отвечает за ширину персонажа. |
| t | Timer | public | Скорость отображения спрайтов персонажа |
| delay | int | - | Отвечает доли секунд. Используется в переменной t. |
| speed | int | public | Отвечает за скорость персонажа. |
| speedL | int | public | Скорость персонажа в левую скорость, создана для возможности скроллинга. |

Продолжение таблицы 2

|  |  |  |  |
| --- | --- | --- | --- |
| speedR | int | public | Скорость персонажа в правую скорость, создана для возможности скроллинга. |
| g1 | Graphics2D | - | Используется для отрисовки персонажа. |
| sides | Sides | - | Используется для указания направления персонажа. |

Класс Person имеет в себе встроенный класс Sides. Класс Sides имеет методы Left и Right отвечающие за направление движения персонажа, также в этом методе можно заметить переменную bool, а это значит, что этот же класс отвечает за направление персонажа уже после движения. Класс Sides использует такие же переменные, что и в классе Person.

Методы персонажа отображены в таблице 3. Методы внутреннего класса рассмотрены не будут.

Таблица 3 – Методы класса Person

|  |  |  |
| --- | --- | --- |
| Метод | Назначение | Что входит |
| Person | Является конструктором класса. | ImageIcon [] [] sprite, int speed, int delay, int x, int y, int w, int h |
| paint | Этот метод отрисовывает спрайты персонажа игры. | Graphics g |
| actionPerformed | Обновляет спрайты персонажа | ActionEvent e |

Продолжение таблицы 3

|  |  |  |
| --- | --- | --- |
| keyTyped | Этот метод используется, когда клавиша была нажата, а затем отпущена на клавиатуре. В данном классе ничего не делает. | KeyEvent e |
| keyPressed | Этот метод используется, когда клавиша была нажата на клавиатуре. | KeyEvent e |
| keyReleased | Этот метод используется, когда клавиша была отпущена на клавиатуре. | KeyEvent e |
| setX | Устанавливает значение переменной для x. | int x |
| setY | Устанавливает значение переменной для y. | int y |
| setH | Устанавливает значение переменной для h. | int h |
| setW | Устанавливает значение переменной для w. | int w |
| getX | Возвращает значение переменной x. | - |
| getY | Возвращает значение переменной y. | - |
| getH | Возвращает значение переменной h. | - |
| getW | Возвращает значение переменной w. | - |
| getSprite | Возвращает значение переменной sprite. | - |
| setSprite | Устанавливает значение переменной для sprite. | ImageIcon [] [] sprite |

Продолжение таблицы 3

|  |  |  |
| --- | --- | --- |
| getGrass\_sp | Возвращает значение переменной grass\_sp. | - |
| setGrass\_sp | Устанавливает значение переменной для grass\_sp. | Image grass\_sp |
| getSpeed | Возвращает значение переменной speed. | - |
| setSpeed | Устанавливает значение переменной для speed. | int speed |

Переменные объекта игры отображены в таблице 4.

Таблица 4 – Переменные класса ObjectGame

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование переменной | Тип данных | Модификатор | Назначение |
| sprite | ImageIcon [] | - | Отвечает за хранение анимации. Это двумерный массив. В него входят i (sprite[i]). |
| i | int | private | Отвечает за прокручивание массива спрайтов из sprite. |
| grass\_sp | Image | - | Служит для отображения одного спрайта из массивов sprite. |
| x | int | public | Отвечает за положение объекта по оси X. |

Продолжение таблицы 4

|  |  |  |  |
| --- | --- | --- | --- |
| y | int | public | Отвечает за положение объекта по оси Y. |
| h | int | public | Отвечает за высоту объекта. |
| w | int | public | Отвечает за ширину объекта. |
| t | Timer | public | Скорость отображения спрайтов объекта |
| delay | int | - | Отвечает доли секунд. Используется в переменной t. |
| g1 | Graphics2D | - | Используется для отрисовки объекта. |

Методы объекта игры отображены в таблице 5.

Таблица 5 – Методы класса ObjectGame

|  |  |  |
| --- | --- | --- |
| Метод | Назначение | Что входит |
| ObjectGame | Является конструктором класса. | ImageIcon [] sprite, int delay, int x, int y, int w, int h |
| paint | Этот метод отрисовывает спрайты объекта игры. | Graphics g |
| actionPerformed | Обновляет спрайты объекта | ActionEvent e |
| setX | Устанавливает значение переменной для x. | int x |
| setY | Устанавливает значение переменной для y. | int y |

Продолжение таблицы 5

|  |  |  |
| --- | --- | --- |
| setH | Устанавливает значение переменной для h. | int h |
| setW | Устанавливает значение переменной для w. | int w |
| getX | Возвращает значение переменной x. | - |
| getY | Возвращает значение переменной y. | - |
| getH | Возвращает значение переменной h. | - |
| getW | Возвращает значение переменной w. | - |

Переменные фона игры отображены в таблице 6.

Таблица 6 – Переменные класса Fon

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование переменной | Тип данных | Модификатор | Назначение |
| fon | ImageIcon | - | Отвечает за хранение картинки. |
| grass | Image | - | Служит для отображения картинки из fon. |
| x | int | public | Отвечает за положение фона по оси X. |
| y | int | public | Отвечает за положение фона по оси Y. |

Продолжение таблицы 6

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| hf | int | public | | Отвечает за высоту фона. |
| wf | int | public | | Отвечает за ширину фона. |
| g1 | Graphics2D | | - | Используется для отрисовки фона. |

Методы фона игры отображены в таблице 7.

Таблица 7 – Методы класса Fon

|  |  |  |
| --- | --- | --- |
| Метод | Назначение | Что входит |
| Fon | Является первым конструктором класса. | ImageIcon fon, int w, int h |
| Fon | Является вторым конструктором класса. | ImageIcon fon, int w, int h, int x, int y |
| paint | Этот метод отрисовывает картинку фона игры. | Graphics g |
| setFon | Устанавливает значение переменной для fon. | ImageIcon fon |
| getFon | Возвращает значение переменной fon. | - |
| getX | Возвращает значение переменной x. | - |
| getY | Возвращает значение переменной y. | - |
| setY | Устанавливает значение переменной для y. | int y |

Продолжение таблицы 7

|  |  |  |
| --- | --- | --- |
| setX | Устанавливает значение переменной для x. | int x |
| setHf | Устанавливает значение переменной для h. | int hf |
| setWf | Устанавливает значение переменной для w. | int wf |
| getHf | Возвращает значение переменной h. | - |
| getWf | Возвращает значение переменной w. | - |

Переменные игры отображены в таблице 8.

Таблица 8 – Переменные класса Game

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование переменной | Тип данных | Модификатор | Назначение |
| f | JFrame | static | Это основное окно, где будут происходить действия. |
| fon0 | Fon | static | Это фон игры. |
| fon1 | Fon | static | Это фон игры. |
| fon2 | Fon | static | Это фон игры. |
| fon3 | Fon | static | Это фон игры. |
| tree | ObjectGame | static | Отображает спрайты дерева. |
| fier | ObjectGame | static | Отображает спрайты огня. |

Продолжение таблицы 8

|  |  |  |  |
| --- | --- | --- | --- |
| oblako1 | ObjectGame | static | Отображает картинку облака. |
| oblako2 | ObjectGame | static | Отображает картинку облака. |
| oblako3 | ObjectGame | static | Отображает картинку облака. |
| oblako4 | ObjectGame | static | Отображает картинку облака. |
| oblako5 | ObjectGame | static | Отображает картинку облака. |
| oblako6 | ObjectGame | static | Отображает картинку облака. |
| oblako7 | ObjectGame | static | Отображает картинку облака. |
| person | Person | static | Непосредственно сам персонаж игры. |
| t | Timer | public | Время отрисовки компонентов игры. |
| screenSize | Dimension | - | Узнаёт размер экрана компьютера. |
| px | int | static | Определяет середину экрана по оси X. |
| speed | int | static | Скорость персонажа. |
| game | Game | - | Используется для отрисовки и для считывания клавиш клавиатуры при запуске данного класса, применяется в переменной f. |

Продолжение таблицы 8

|  |  |  |  |
| --- | --- | --- | --- |
| sprite | ImageIcon [] [] | - | Спрайты персонажа игры. |
| ZF0 | ImageIcon | - | Используется для хранения изображения неба с солнцем. |
| ZF | ImageIcon | - | Используется для хранения изображения гор. |
| PF | ImageIcon | - | Используется для хранения изображения травы с кустами. |
| oblako | ImageIcon [] | - | Используется для хранения изображения облака. |
| TreeSP | ImageIcon [] | - | Используется для хранения спрайтов дерева. |
| FierSP | ImageIcon [] | - | Используется для хранения спрайтов костра. |
| cursorImg | BufferedImage | - | Используется для считывания курсора мыши. |
| blankCursor | Cursor | - | Убирает курсор мыши. |
| g1 | Graphics2D | - | Используется для отображения всех компонентов игры. |

Методы игры отображены в таблице 9.

Таблица 9 – Методы класса Game

|  |  |  |
| --- | --- | --- |
| Метод | Назначение | Что входит |
| main | Запускает программу. | String [] PascalABC |
| paint | Отрисовывает компоненты программы. Производится логика движения облаков. | Graphics g |
| actionPerformed | Обновляет картину игры. | ActionEvent e |
| keyTyped | Этот метод используется, когда клавиша была нажата, а затем отпущена на клавиатуре. В данном классе ничего не делает. | KeyEvent e |
| keyPressed | Этот метод используется, когда клавиша была нажата на клавиатуре. В нём создана логики скроллинга для персонажа игры. | KeyEvent e |
| keyReleased | Этот метод используется, когда клавиша была отпущена на клавиатуре. Это относится только к персонажу игры. | KeyEvent e |

# Тестирование и установка приложения

Тестирование. Следует протестировать адаптивность интерфейса приложения, то есть, интерфейс приложения должен меняться в зависимости от расширения экрана. Чтобы это проверить нужно зайти в Параметры> Дисплей и в «Разрешения дисплея» изменять параметры экрана и после этого запускать приложение. При первом запуске приложения оно выглядит как показано на рисунке 10, разрешение экрана при этом 1920Х1080.



Рисунок 10 – Приложение при разрешении экрана 1920Х1080

Далее меняем параметры дисплея на 1366Х768 и приложение будет выглядеть как на рисунке 11.



Рисунок 11 – Приложение при разрешении экрана 1366Х768

Ничего особо не поменялось, теперь уменьшим размер до минимального, до 800Х600 и выглядеть это будет как на рисунке 12.

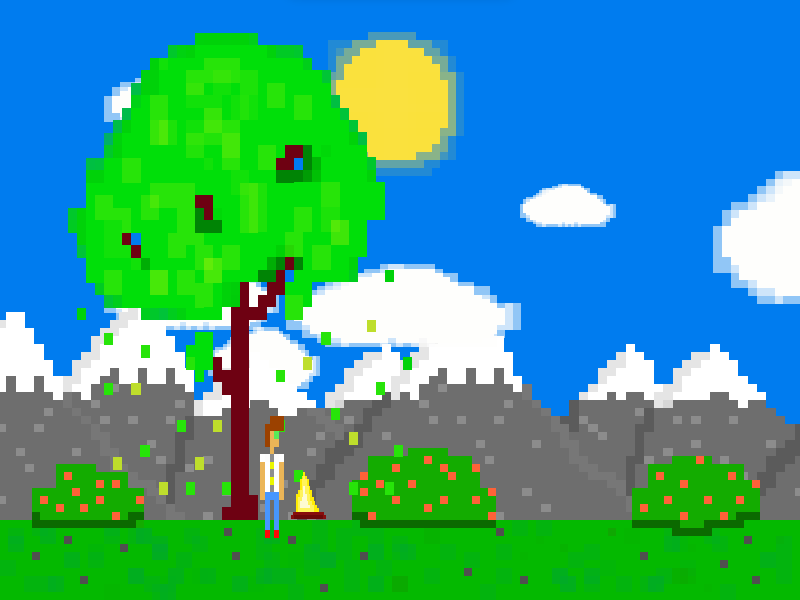


Рисунок 12 – Приложение при разрешении экрана 800Х600

Снова ничего не изменитесь, из этого следует сделать вывод, что приложение подходит для любых экранов с разным разрешением, если бы программа не была адаптирована под разные устройства, то это выглядело примерно, как на рисунке 13.



Рисунок 13 – Приложение, не адаптивное к другим разрешениям экрана

Кстати, при изменении разрешения экрана, когда приложение запущенно, приложение смещается. Для устранения этого, нужно выйти из него и заново зайти.

Этого удалось достигнуть благодаря формуле (1) и формуле (2). Ещё (1) формула используется для скорости персонажа, так как

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | ; | | | (1) |
| где |  | - | адаптированное расположение по x(ширина) объекта; | |
|  |  | - | расположение по x(ширина) объекта; | |
|  |  | - | ширина экрана. | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | ; | | | (2) |
| где |  | - | адаптированное расположение по y(высота) объекта; | |
|  |  | - | расположение по y(высота) объекта; | |
|  |  | - | высота экрана. | |

Если объект должен быть размером с экран по высоте и по ширине, то необходимо указать нынешний размер экрана, как правило это фон.

Далее следует протестировать класс Person, для того нужно поставить курсор на название класса и нажать комбинацию «Alt+Enter». После этого появится окно, как на рисунке 14, и нужно нажать «Create Test».

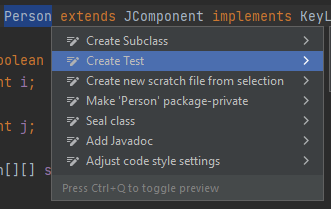


Рисунок 14 – Создание теста

После этого должно появиться окно, в котором указываются методы для их дальнейшего тестирования, как на рисунке 15.

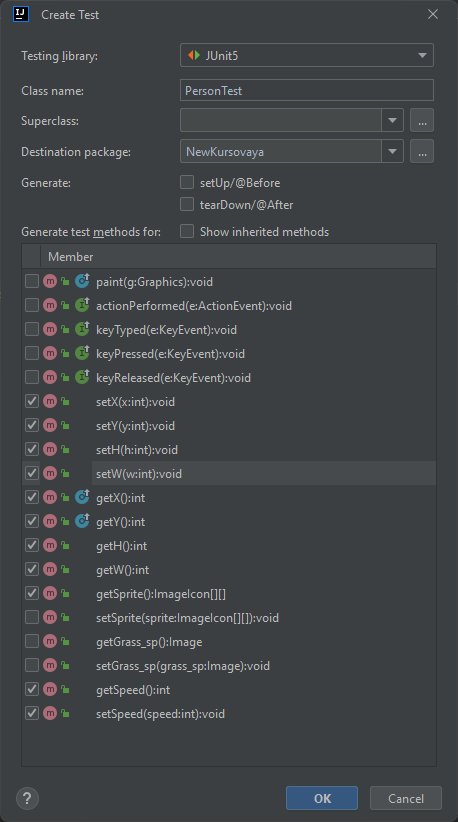


Рисунок 15 – Выбираем методы для тестирования

Далее нажимается «ОК» и после этого создастся класс с методами теста, в них вписывается код для проверки. Для начала создаётся массив спрайтов и вызывается класс Person вне методов тестирования, потом используется переменная класса Person уже непосредственно в методах тестирования. В методах будет проводиться тестирование работы Set-ов и Get-ов и то, как они влияют на класс.

Листинг юнит теста [6] есть в приложении. На рисунке 16 представлен результат тестирования после запуска теста, и как видно, программа прошла тестирование успешно.

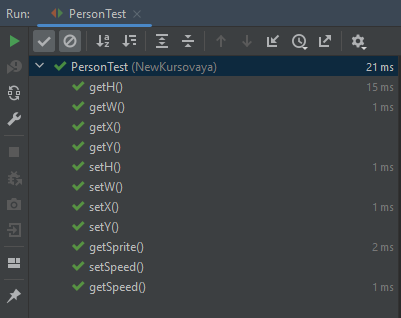


Рисунок 16 – Итог тестирования класса Person

Установка приложения. Для начала установки приложения необходимо java-файл преобразовать в jar-файл. Для этого нужно зайти в File> Project Structure, как показано на рисунке 17.

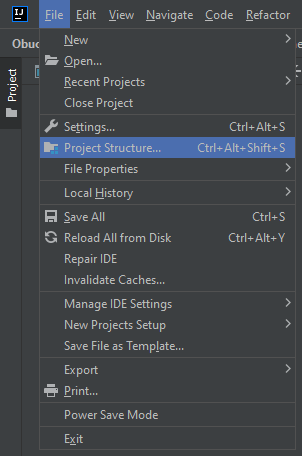


Рисунок 17 – Заходим в Project Structure через File

Перед нами откроется окно Project Structure, в нём нужно зайти в Artifacts, далее нужно нажать на плюсик и в появившейся области выбрать JAR> From modules with dependencies..., как указано на рисунке 18.

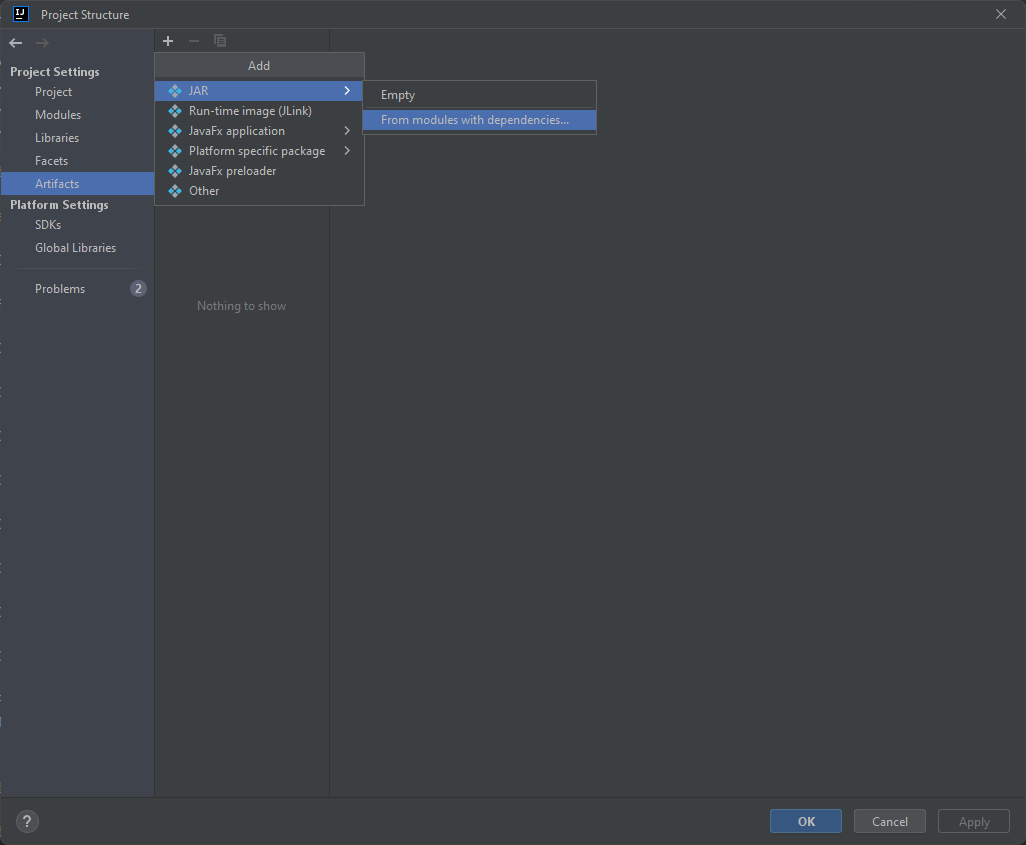


Рисунок 18 – Project Structure

Далее появится окно Create JAR from Modules, как на рисунке 19.

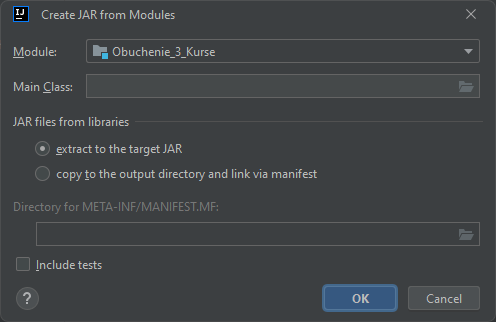


Рисунок 19 – Create JAR from Modules

Далее указывается класс Game.java в Main Class, для этого нужно нажать на  и в появившемся окне выбрать этот класс, как на рисунке 20.

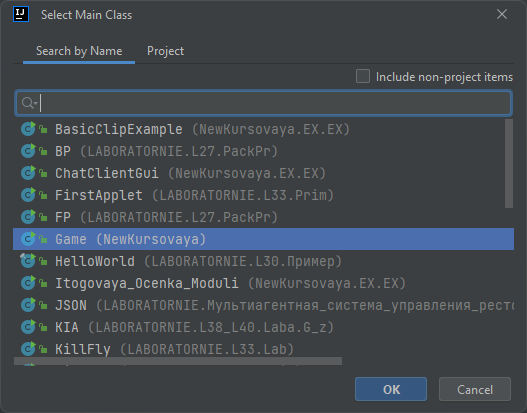


Рисунок 20 – Указываем класс Game

После этого нужно нажать «ОК» и также в Create JAR from Modules нажать «ОК». В Project Structure необходимо выбрать «Apply» после чего тоже нажать «ОК». Далее требуется зайти в Build> Build Artifacts, как показано на рисунке 21. Это нужно для создания исполняемого jar-файла.

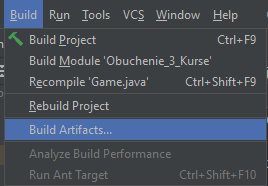


Рисунок 21 – Build Artifacts

В появившемся окне необходимо выбрать Build, как показано на рисунке 22.

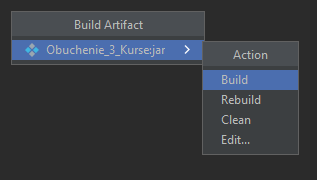


Рисунок 22 – Создаем jar-файл

Теперь всё готово, jar-файл создан, находится данный файл в out> artifacts> Obuchenie\_3\_Kurse\_jar, выглядеть это должно так, как показано на рисунке 23.

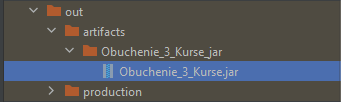


Рисунок 23 – Расположение jar-файл

После запуска приложения будет показан белый экран, это означает, что jar-файл не может найти ресурс, то есть, спрайты. Для решения этой проблемы нужно перекинуть необходимую папку туда, где находится jar-файл.

Эта проблема происходит, если путь к ресурсу был указан так: «res\\Sprite\\Person\\S\_R\_2.png», хотя это более правильно, так как даёт возможность свободно переносить папку с jar-файлом и с его ресурсами между компьютерами, где можно использовать данный формат. Но тут появляется другая проблема, что формат jar-файлов не везде доступен, поэтому следует jar-файл перевести в формат exe. Для этого используем программу launch4j [1]. В поле «\*jar:» требуется указать расположение jar-файла, а поле «\*Output file:» указывается куда перенести файл в формате exe (в названии файла в конце следует дописать «.exe», иначе выдаст ошибку), также можно указать иконку данного приложения в поле «Icon:», как показано на рисунке 24.

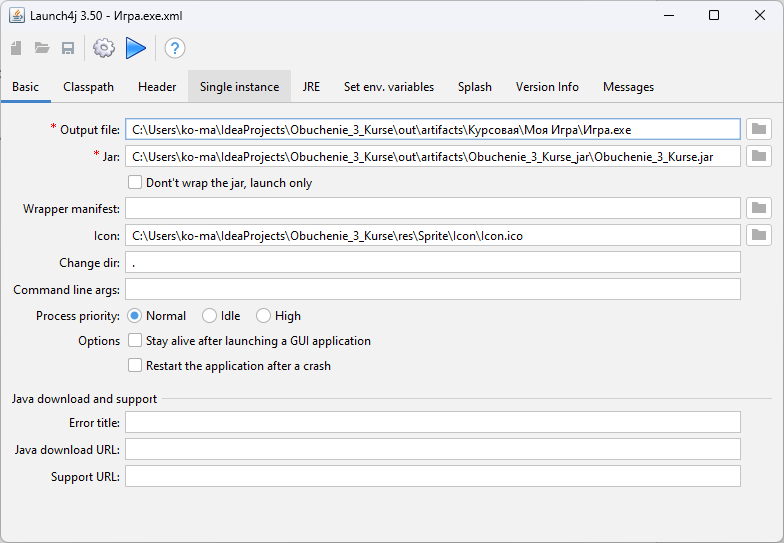


Рисунок 24 – Указываем расположение нужных нам файлов

Иконку, которая изображена на рисунке 25, можно создать из уже сделанных картинок, для этого можно использовать Figma [10].



Рисунок 25 – Иконка приложения

Далее для следующего действия нужно найти папку, где располагается Java, и копировать в ту папку, где будет создан exe-файл, данная папка должна выглядеть как на рисунке 26.

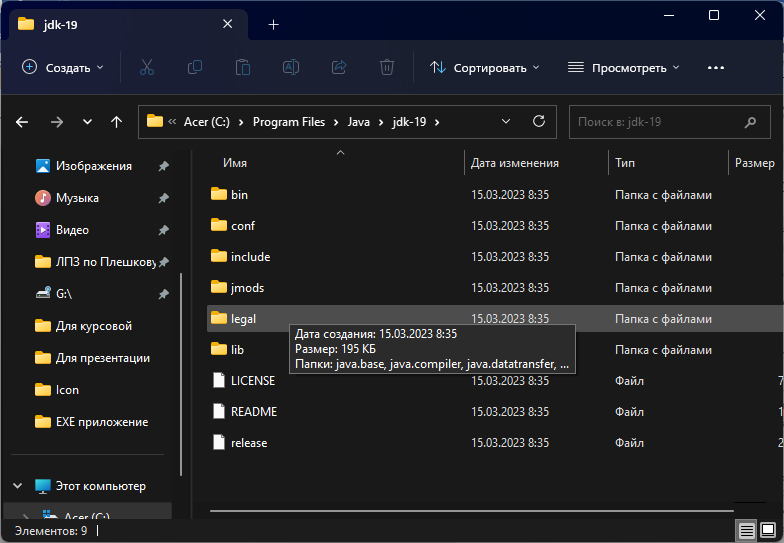


Рисунок 26 – Файлы Java

Перемещённые файлы Java помещаются в папку «JRE», заранее созданную. Далее нажимаем на вкладку JRE в программе launch4j, и в поле «JRE Paths:» указываем имя созданного файла, то есть. «JRE», как показано на рисунке 27.

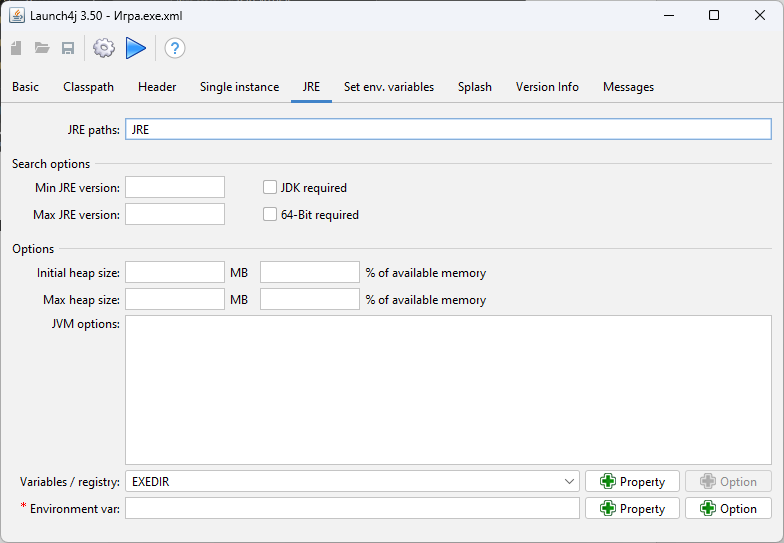


Рисунок 27 – Указываем папку JRE

Далее необходимо кликнуть на  , после чего появится окно, как на рисунке 28, в котором тоже нужно указать путь и наименование приложения, после этого нужно сохранить нажатием на «Save».

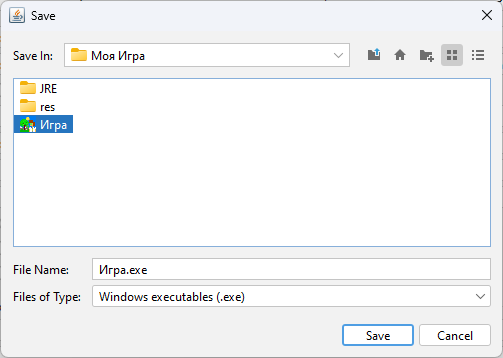


Рисунок 28 – Сохраняем наше приложение

После проделанной работы, приложение появится в той папке, в которую был указан путь.

Главное для установки на другой компьютер или переносе приложения на жёстком носителе (флешке, диске) нужно расположить в одной папке папку с Java и папку с картинками, иначе приложение работать не будет. Также можно саму папку с приложением архивировать для перемещения по сетям интернет (например, прислать программу другу через Телеграм).

# Заключение

В процессе выполнения курсового проекта на тему «Разработка и программная реализация модели персонажа игры» были достигнуты следующие результаты:

1. Был разработан персонаж игры, которого можно внедрить в почти любой игровой проект. В процессе разработки персонажа игры, также была разработана демонстрационная игра, где были продемонстрированы возможности данного класса.
2. Были сгенерированы следующие особенности программной реализации персонажа игры:
3. данная игра доступна на компьютерах с любым разрешением экрана;
4. был реализован скроллинг персонажа благодаря открытым переменным данного класса.
5. На основе проделанной работы были разработаны следующие рекомендации:
6. Разработать персонажу возможность прыжка;
7. Сгенерировать возможность выбирать в какие стороны данный персонаж может двигаться (или AD, или WASD, или WS).
8. Создать встроенный скроллинг.
9. Создать возможность персонажу влиять на объекты игры.

Таким образом, на основе вышеизложенного можно сделать вывод, что программа в целом выполнена функционально, дизайн программы прост и понятен пользователю, тем не менее, данную программу можно улучшить, добавив новые возможности персонажу и исправив некоторые недочёты.

# Список использованной литературы

1. https://sourceforge.net/projects/launch4j/files/launch4j-3/3.50/ - Ссылка на скачивание launch4j;
2. https://ru.freedownloadmanager.org/Windows-PC/PyxelEdit-FREE.html - Ссылка на скачивание PyxelEdit;
3. https://www.oracle.com/java/technologies/downloads/ - Ссылка на скачивание Java;
4. https://www.jetbrains.com/ru-ru/idea/ - Ссылка на скачивание IntelliJ IDEA;
5. https://ru.wikipedia.org/wiki/Скроллинг#:~:text=Скроллинг%20—%20форма%20представления%20информации%2C,изменяет%20содержимое%2C%20но%20передвигает%20«камеру» - определение скроллинга;
6. https://www.youtube.com/playlist?list=PL7ZzXmLk6CYXCsJTwtClt42Xtn4lMDABZ – модульное тестирование (Unit Testing);
7. https://www.youtube.com/watch?v=JfpCicDUMKc - Изучение GitHub в одном видео уроке за 15 минут;
8. https://en.wikipedia.org/wiki/Kingdom:\_New\_Lands – Описание игры «Kingdom: New Lands»;
9. https://thelastgame.ru/until-we-die/#:~:text=Until%20We%20Die%20(Пока%20не,удалось%20выжить%20после%20всемирной%20катастрофы - Описание игры «Until we die»;
10. https://www.figma.com/?fuid= - сайт для использования Figma.

Приложение.

Класс PersonTest.java (Юнит тест)

import org.junit.jupiter.api.Test;  
import javax.swing.\*;  
import static org.junit.jupiter.api.Assertions.\*;  
  
class PersonTest {  
 ImageIcon[][] sprite = new ImageIcon[][]{  
 {new ImageIcon("res\\Sprite\\Person\\S\_R\_2.png"),  
 new ImageIcon("res\\Sprite\\Person\\S\_R\_4.png"),  
 new ImageIcon("res\\Sprite\\Person\\S\_R\_3.png"),  
 new ImageIcon("res\\Sprite\\Person\\S\_R\_4.png"),  
 new ImageIcon("res\\Sprite\\Person\\S\_R\_2.png"),  
 new ImageIcon("res\\Sprite\\Person\\S\_R\_4.png"),  
 new ImageIcon("res\\Sprite\\Person\\S\_R\_3.png"),  
 new ImageIcon("res\\Sprite\\Person\\S\_R\_4.png")},  
 {new ImageIcon("res\\Sprite\\Person\\S\_L\_2.png"),  
 new ImageIcon("res\\Sprite\\Person\\S\_L\_4.png"),  
 new ImageIcon("res\\Sprite\\Person\\S\_L\_3.png"),  
 new ImageIcon("res\\Sprite\\Person\\S\_L\_4.png"),  
 new ImageIcon("res\\Sprite\\Person\\S\_L\_2.png"),  
 new ImageIcon("res\\Sprite\\Person\\S\_L\_4.png"),  
 new ImageIcon("res\\Sprite\\Person\\S\_L\_3.png"),  
 new ImageIcon("res\\Sprite\\Person\\S\_L\_4.png")},  
 {new ImageIcon("res\\Sprite\\Person\\S\_R\_1.png"),  
 new ImageIcon("res\\Sprite\\Person\\S\_L\_1.png")}};  
 Person person = new Person(sprite, 20,30,50,50,170,220);  
 @Test  
 void setX() {  
 person.setX(100);  
 assertEquals(100, person.getX());  
 }  
  
 @Test  
 void setY() {  
 person.setY(120);  
 assertEquals(120, person.getY());  
 }  
  
 @Test  
 void setH() {  
 person.setH(200);  
 assertEquals(200, person.getH());  
 }  
  
 @Test  
 void setW() {  
 person.setW(185);  
 assertEquals(185, person.getW());  
 }  
  
 @Test  
 void getX() {  
 assertEquals(50, person.getX());  
 }  
  
 @Test  
 void getY() {  
 assertEquals(50, person.getY());  
 }  
  
 @Test  
 void getH() {  
 assertEquals(220, person.getH());  
 }  
  
 @Test  
 void getW() {  
 assertEquals(170, person.getW());  
 }  
  
 @Test  
 void getSpeed() {  
 assertEquals(20, person.getSpeed());  
 }  
  
 @Test  
 void setSpeed() {  
 person.setSpeed(50);  
 assertEquals(50, person.getSpeed());  
 }  
  
 @Test  
 void getSprite() {  
 assertEquals(sprite, person.getSprite());  
 }  
}

Класс Person.java

import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.awt.event.KeyEvent;  
import java.awt.event.KeyListener;  
  
import static java.awt.event.KeyEvent.\*;  
  
public class Person extends JComponent implements KeyListener, ActionListener {  
 public boolean bool = true;  
 private int i;  
 private int j;  
 ImageIcon[][] sprite;  
 Image grass\_sp;  
 public int x;  
 public int y;  
 public int h;  
 public int w;  
 int delay;  
 public int speed;  
 public int speedL = speed;  
 public int speedR = speed;  
 public Timer t;  
  
 public Person(ImageIcon[][] sprite, int speed, int delay, int x, int y, int w, int h) {  
 this.sprite = sprite;  
 this.speed = speed;  
 this.delay = delay;  
 this.x = x;  
 this.y = y;  
 this.h = h;  
 this.w = w;

t = new Timer(delay, this);  
 grass\_sp = sprite[2][0].getImage();  
 }  
  
 public void paint(Graphics g) {  
 super.paint(g);  
 Graphics2D g1 = (Graphics2D) g;  
 g1.drawImage(grass\_sp, x, y, w, h, null);  
 }  
  
 class Sides {  
 public void Left() {  
 bool = false;  
 x -= speedL;  
 i = 1;  
 j = j + 1;  
 if (j > sprite[i].length - 1) {  
 j = 0;  
 }  
 grass\_sp = sprite[i][j].getImage();  
 }  
  
 public void Right() {  
 bool = true;  
 x += speedR;  
 i = 0;  
 j = j + 1;  
 if (j > sprite[i].length - 1) {  
 j = 0;  
 }  
 grass\_sp = sprite[i][j].getImage();  
 }  
 }  
  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 repaint();  
 }  
  
 @Override  
 public void keyTyped(KeyEvent e) {  
 }  
  
 @Override  
 public void keyPressed(KeyEvent e) {  
 Sides sides = new Sides();  
 if ((e.getKeyCode() == VK\_RIGHT || e.getKeyCode() == KeyEvent.VK\_D)) {  
 sides.Right();  
 } else if ((e.getKeyCode() == KeyEvent.VK\_LEFT || e.getKeyCode() == KeyEvent.VK\_A)) {  
 sides.Left();  
 }  
  
 }  
  
 @Override  
 public void keyReleased(KeyEvent e) {  
 if (bool && (e.getKeyCode() == VK\_RIGHT || e.getKeyCode() == KeyEvent.VK\_D)) {  
 j = 0;  
 grass\_sp = sprite[2][0].getImage();  
 } else if (!bool && (e.getKeyCode() == VK\_LEFT || e.getKeyCode() == KeyEvent.VK\_A)) {  
 j = 0;  
 grass\_sp = sprite[2][1].getImage();  
 }  
 }  
  
 public void setX(int x) {  
 this.x = x;  
 }  
  
 public void setY(int y) {  
 this.y = y;  
 }  
  
 public void setH(int h) {  
 this.h = h;  
 }  
  
 public void setW(int w) {  
 this.w = w;  
 }  
  
 public int getX() {  
 return x;  
 }  
  
 public int getY() {  
 return y;  
 }  
  
 public int getH() {  
 return h;  
 }  
  
 public int getW() {  
 return w;  
 }  
  
 public ImageIcon[][] getSprite() {  
 return sprite;  
 }  
  
 public void setSprite(ImageIcon[][] sprite) {  
 this.sprite = sprite;  
 }  
  
 public Image getGrass\_sp() {  
 return grass\_sp;  
 }  
  
 public void setGrass\_sp(Image grass\_sp) {  
 this.grass\_sp = grass\_sp;  
 }  
  
 public int getSpeed() {  
 return speed;  
 }  
  
 public void setSpeed(int speed) {  
 this.speed = speed;  
 }  
}

Класс ObjectGame.java

import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
  
public class ObjectGame extends JComponent implements ActionListener {  
 public int i = 0;  
 ImageIcon[] sprite;  
 Image grass\_sp;  
 public int x;  
 public int y;  
 public int h;  
 public int w;  
 int delay;  
 public Timer t;  
  
 public ObjectGame(ImageIcon[] sprite, int delay, int x, int y, int w, int h) {  
 this.delay = delay;  
 this.x = x;  
 this.y = y;  
 this.h = h;  
 this.w = w;  
 t = new Timer(delay, this);  
 this.sprite = sprite;  
 grass\_sp = sprite[0].getImage();  
 }  
  
 public void paint(Graphics g) {  
 Graphics2D g1 = (Graphics2D) g;  
 g1.drawImage(grass\_sp, x, y, w, h, null);  
  
 if (i > sprite.length - 1) {  
 i = 0;  
 }  
 grass\_sp = sprite[i].getImage();  
 i++;  
 t.start();  
 }  
  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 repaint();  
 }  
  
 public void setX(int x) {  
 this.x = x;  
 }  
  
 public void setY(int y) {  
 this.y = y;  
 }  
  
 public void setH(int h) {  
 this.h = h;  
 }  
  
 public void setW(int w) {  
 this.w = w;  
 }  
  
 public int getX() {  
 return x;  
 }  
  
 public int getY() {  
 return y;  
 }  
  
 public int getH() {  
 return h;  
 }  
  
 public int getW() {  
 return w;  
 }  
}

Класс Fon.java

import javax.swing.\*;  
import java.awt.\*;  
  
public class Fon {  
 Image grass;  
 ImageIcon fon;  
  
 public int x = 0;  
 public int y = 0;  
 public int hf;  
 public int wf;  
  
 Fon(ImageIcon fon, int w, int h) {  
 this.fon = fon;  
 wf = w;  
 hf = h;  
 grass = fon.getImage();  
 }  
  
 Fon(ImageIcon fon, int w, int h, int x, int y) {  
 this.fon = fon;  
 this.x = x;  
 this.y = y;  
 wf = w;  
 hf = h;  
 grass = fon.getImage();  
 }  
  
 public void paint(Graphics g) {  
 Graphics2D g1 = (Graphics2D) g;  
 g1.drawImage(grass, x, y, wf, hf, null);  
 }  
  
 public void setFon(ImageIcon fon) {  
 this.fon = fon;  
 }  
  
 public ImageIcon getFon() {  
 return fon;  
 }  
  
 public void setHf(int hf) {  
 this.hf = hf;  
 }  
  
 public void setWf(int wf) {  
 this.wf = wf;  
 }  
 public int getHf() {  
 return hf;  
 }  
  
 public int getWf() {  
 return wf;  
 }  
  
 public void getX(int x) {  
 this.x = x;  
 }  
  
 public void getY(int y) {  
 this.y = y;  
 }  
 public void setX(int x) {  
 this.x = x;  
 }  
  
 public void setY(int y) {  
 this.y = y;  
 }  
}

Класс Game.java

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

import java.awt.image.BufferedImage;

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

public class Game extends JComponent implements KeyListener, ActionListener {

static JFrame f = new JFrame();

static Fon fon0, fon1, fon2, fon3;

static ObjectGame tree, fier;

static ObjectGame oblako1;

static ObjectGame oblako2;

static ObjectGame oblako3;

static ObjectGame oblako4;

static ObjectGame oblako5;

static ObjectGame oblako6;

static ObjectGame oblako7;

static Person person;

public Timer t = new Timer(60, this);

static int px;

static int speed;

public static void main(String[] PascalABC) {

Game game = new Game();

ImageIcon[][] sprite = new ImageIcon[][]{

{new ImageIcon("res\\Sprite\\Person\\S\_R\_2.png"),

new ImageIcon("res\\Sprite\\Person\\S\_R\_4.png"),

new ImageIcon("res\\Sprite\\Person\\S\_R\_3.png"),

new ImageIcon("res\\Sprite\\Person\\S\_R\_4.png"),

new ImageIcon("res\\Sprite\\Person\\S\_R\_2.png"),

new ImageIcon("res\\Sprite\\Person\\S\_R\_4.png"),

new ImageIcon("res\\Sprite\\Person\\S\_R\_3.png"),

new ImageIcon("res\\Sprite\\Person\\S\_R\_4.png")},

{new ImageIcon("res\\Sprite\\Person\\S\_L\_2.png"),

new ImageIcon("res\\Sprite\\Person\\S\_L\_4.png"),

new ImageIcon("res\\Sprite\\Person\\S\_L\_3.png"),

new ImageIcon("res\\Sprite\\Person\\S\_L\_4.png"),

new ImageIcon("res\\Sprite\\Person\\S\_L\_2.png"),

new ImageIcon("res\\Sprite\\Person\\S\_L\_4.png"),

new ImageIcon("res\\Sprite\\Person\\S\_L\_3.png"),

new ImageIcon("res\\Sprite\\Person\\S\_L\_4.png")},

{new ImageIcon("res\\Sprite\\Person\\S\_R\_1.png"),

new ImageIcon("res\\Sprite\\Person\\S\_L\_1.png")}};

ImageIcon ZF0 = new ImageIcon("res\\Zad\_Fon\_0.png");

ImageIcon ZF = new ImageIcon("res\\Zad\_Fon.png");

ImageIcon PF = new ImageIcon("res\\Per\_Fon.png");

ImageIcon[] oblako = new ImageIcon[]{new ImageIcon("res\\Oblako.png")};

ImageIcon[] TreeSP = new ImageIcon[]{

new ImageIcon("res\\Sprite\\Tree1.png"),

new ImageIcon("res\\Sprite\\Tree1.png"),

new ImageIcon("res\\Sprite\\Tree2.png"),

new ImageIcon("res\\Sprite\\Tree2.png")

};

ImageIcon[] FierSP = new ImageIcon[]{new ImageIcon("res\\Sprite\\Fier\\Fier1.png"),

new ImageIcon("res\\Sprite\\Fier\\Fier1.png"),

new ImageIcon("res\\Sprite\\Fier\\Fier1.png"),

new ImageIcon("res\\Sprite\\Fier\\Fier2.png"),

new ImageIcon("res\\Sprite\\Fier\\Fier2.png"),

new ImageIcon("res\\Sprite\\Fier\\Fier2.png")};

Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();

px = (int) (screenSize.getWidth() / 2 - 170 + 100);

speed = (int) (20 / (1920 / screenSize.getWidth()));

f.setExtendedState(JFrame.MAXIMIZED\_BOTH);

f.setUndecorated(true);

person = new Person(sprite, speed, 0, (int) (570 / (1920 / screenSize.getWidth())), (int) (750 / (1080 / screenSize.getHeight())), (int) (170 / (1920 / screenSize.getWidth())), (int) (220 / (1080 / screenSize.getHeight())));

tree = new ObjectGame(TreeSP, 0, (int) (120 / (1920 / screenSize.getWidth())), (int) (36 / (1080 / screenSize.getHeight())), (int) ((200 \* 3 + 270) / (1920 / screenSize.getWidth())), (int) ((200 \* 3 + 300) / (1080 / screenSize.getHeight())));

fier = new ObjectGame(FierSP, 1, (int) (700 / (1920 / screenSize.getWidth())), (int) (851 / (1080 / screenSize.getHeight())), (int) (85 / (1920 / screenSize.getWidth())), (int) (85 / (1080 / screenSize.getHeight())));

oblako1 = new ObjectGame(oblako, 0, (int) (250 / (1920 / screenSize.getWidth())), (int) (555 / (1080 / screenSize.getHeight())), (int) (270 / (1920 / screenSize.getWidth())), (int) (175 / (1080 / screenSize.getHeight())));

oblako2 = new ObjectGame(oblako, 0, (int) (15 / (1920 / screenSize.getWidth())), (int) (470 / (1080 / screenSize.getHeight())), (int) (255 / (1920 / screenSize.getWidth())), (int) (122 / (1080 / screenSize.getHeight())));

oblako3 = new ObjectGame(oblako, 0, (int) (0 / (1920 / screenSize.getWidth())), (int) (400 / (1080 / screenSize.getHeight())), (int) (421 / (1920 / screenSize.getWidth())), (int) (234 / (1080 / screenSize.getHeight())));

oblako4 = new ObjectGame(oblako, 0, (int) (1000 / (1920 / screenSize.getWidth())), (int) (307 / (1080 / screenSize.getHeight())), (int) (231 / (1920 / screenSize.getWidth())), (int) (123 / (1080 / screenSize.getHeight())));

oblako5 = new ObjectGame(oblako, 0, (int) (314 / (1920 / screenSize.getWidth())), (int) (430 / (1080 / screenSize.getHeight())), (int) (564 / (1920 / screenSize.getWidth())), (int) (237 / (1080 / screenSize.getHeight())));

oblako6 = new ObjectGame(oblako, 0, (int) (1213 / (1920 / screenSize.getWidth())), (int) (195 / (1080 / screenSize.getHeight())), (int) (634 / (1920 / screenSize.getWidth())), (int) (423 / (1080 / screenSize.getHeight())));

oblako7 = new ObjectGame(oblako, 0, (int) (0 / (1920 / screenSize.getWidth())), (int) (55 / (1080 / screenSize.getHeight())), (int) (564 / (1920 / screenSize.getWidth())), (int) (237 / (1080 / screenSize.getHeight())));

fon0 = new Fon(ZF0, (int) screenSize.getWidth(), (int) screenSize.getHeight());

fon1 = new Fon(ZF, (int) screenSize.getWidth() + 420, (int) screenSize.getHeight());

fon1.x = -50;

fon2 = new Fon(PF, (int) screenSize.getWidth(), (int) screenSize.getHeight());

fon3 = new Fon(PF, (int) screenSize.getWidth(), (int) screenSize.getHeight(), fon2.wf, 0);

f.setResizable(false);

f.setDefaultCloseOperation(WindowConstants.EXIT\_ON\_CLOSE);

BufferedImage cursorImg = new BufferedImage(16, 16, BufferedImage.TYPE\_INT\_ARGB);

Cursor blankCursor = Toolkit.getDefaultToolkit().createCustomCursor(cursorImg, new Point(0, 0), "blank cursor");

f.getContentPane().setCursor(blankCursor);

f.getContentPane().add(game);

f.addKeyListener(game);

f.setVisible(true);

}

public void paint(Graphics g) {

Graphics2D g1 = (Graphics2D) g;

fon0.paint(g1);

oblako1.paint(g1);

if (oblako1.x >= 1920) {

oblako1.setX(-oblako1.w);

} else {

oblako1.setX(oblako1.x + 2);

}

oblako2.paint(g1);

if (oblako2.x >= 1920) {

oblako2.setX(-oblako2.w);

} else {

oblako2.setX(oblako2.x + 3);

}

oblako3.paint(g1);

if (oblako3.x >= 1920) {

oblako3.setX(-oblako3.w);

} else {

oblako3.setX(oblako3.x + 2);

}

fon1.paint(g1);

fon2.paint(g1);

fon3.paint(g1);

oblako4.paint(g1);

if (oblako4.x >= 1920) {

oblako4.setX(-oblako4.w);

} else {

oblako4.setX(oblako4.x + 2);

}

oblako5.paint(g1);

if (oblako5.x >= 1920) {

oblako5.setX(-oblako5.w);

} else {

oblako5.setX(oblako5.x + 3);

}

oblako6.paint(g1);

if (oblako6.x >= 1920) {

oblako6.setX(-oblako6.w);

} else {

oblako6.setX(oblako6.x + 4);

}

oblako7.paint(g1);

if (oblako7.x >= 1920) {

oblako7.setX(-oblako7.w);

} else {

oblako7.setX(oblako7.x + 2);

}

tree.paint(g1);

fier.paint(g1);

person.paint(g1);

t.start();

}

@Override

public void actionPerformed(ActionEvent e) {

fier.actionPerformed(e);

person.actionPerformed(e);

repaint();

}

@Override

public void keyTyped(KeyEvent e) {

}

@Override

public void keyPressed(KeyEvent e) {

person.keyPressed(e);

if (e.getKeyCode() == VK\_RIGHT || e.getKeyCode() == KeyEvent.VK\_D || e.getKeyCode() == VK\_LEFT || e.getKeyCode() == KeyEvent.VK\_A) {

if (person.x <= 0) {

person.speedL = 0;

} else if (person.x + person.w >= fon2.wf) {

person.speedR = 0;

} else if (person.x >= px + 100 && fon3.x + fon3.wf > fon2.wf) {

fon2.x -= speed;

fon3.x -= speed;

fon1.x -= speed/15;

fier.x -= speed;

tree.x -= speed;

person.speedR = 0;

} else if (person.x <= px - 100 && fon2.x < 0) {

fon2.x += speed;

fon3.x += speed;

fon1.x += speed/15;

fier.x += speed;

tree.x += speed;

person.speedL = 0;

} else {

person.speedL = speed;

person.speedR = speed;

}

}

if (e.getKeyCode() == VK\_ESCAPE) {

f.dispatchEvent(new WindowEvent(f, WindowEvent.WINDOW\_CLOSING));

}

}

@Override

public void keyReleased(KeyEvent e) {

person.keyReleased(e);

}

}