Белорусский государственный университет информатики и радиоэлектроники

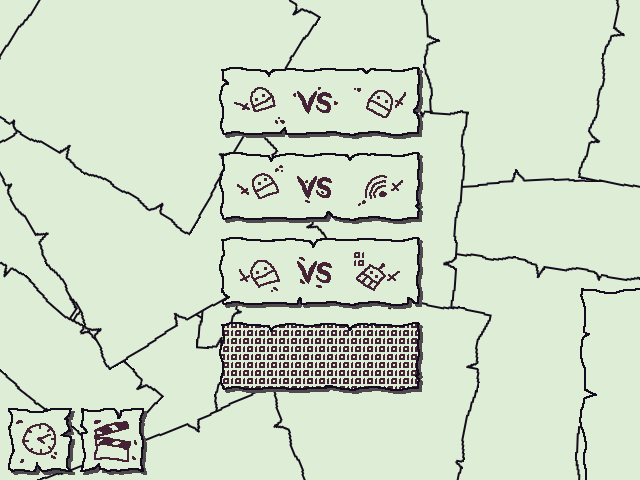
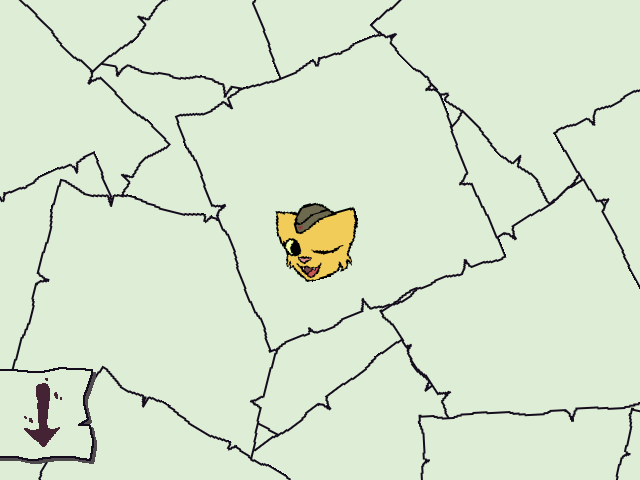
Отчет по курсу КПП на тему:

«Bomb ‘n Shovel»

|  |  |
| --- | --- |
| Выполнил:  Савин Д. В. | Проверил:  Кухарчук И.В.  Лектор:  Искра Н.А. |

Минск

2017

**Лабораторная работа №1.   
Разработка кроссплатформенного интерфейса**Управление осуществляется мышью. Главное меню содержит четыре режима игры: Локальный мультиплеер, Сетевой мультиплеер, Матч против бота и Автономный режим. В нижнем левом углу находится кнопка включения\выключения лимита времени. Рядом с ней – кнопка просмотра реплея предыдущего матча.   
  
  
Рис. 1.1  
  
При выборе сетевой игры игрок попадает в лобби, где ждёт, пока сервер соединит его с другим игроком. В левом нижнем углу имеется кнопка выхода обратно в главное меню, в центре – милый котик.  
  
  
Рис. 1.2  
При соединении с другим игроком или выбрав один из других режимов игры, начинается собственно геймплей. В левом нижнем углу находится таймер, если выбран режим игры на время, в правом – кнопки обзора и завершения хода.  
  
  
Рис. 1.3  
  
В начале игры после рандомной генерации карты каждому игроку даётся четыре юнита. Юнит может ходить по траве и мостикам, рубить деревья и убивать вражеских юнитов. Цель игры – убить всех юнитов врага.

**Лабораторная работа №2. Программирование алгоритмов с использованием механизмов объектно-ориентированного программирования**

В ходе выполнения лабораторной работы было разработано полностью работоспособное приложение «2048».

Предусмотрено 2 вида сложности:

1. Easy – обычный (после каждого хода в случайную пустую клетку поля добавляется «двойка»).
2. Hard – усложненный (с вероятностью в 30% приложение будет неправильно реагировать на действия игрока; например: игрок нажал клавишу «ВНИЗ», а поле сдвинулось «ВЛЕВО»).

Добавлен «БОТ», который случайным образом генерирует введенное количество ходов.

Спроектированная диаграмма классов представлена на рис. 2.

Блок-схема основного алгоритма (смещение поля вверх) представлена на рис.3.

Сгенерированная на основе комментариев документация находится в приложенном html-файле [2048\dist\javadoc](file:///C:\Users\люба\Проекты%20Java\2048\dist\javadoc\index.html) (рис.4-5)



рис.2

Для построения приложения были реализованы четыре класса:

1. Class Main – отрисовка приложения.
2. Class ThreadForReplay – замедленная отрисовка для загрузки из файла или ходов бота
3. Class NewThread – логика приложения.
4. Enum Direction – перечисление для унификации в коде и в аннотации

Главный класс **Main** отвечает за отрисовку каждого элемента приложения, а также начальную инициализацию переменных Сначала в методе *start* происходит отрисовка первоначального окна меню (добавление фона, кнопок, прикрепление к кнопкам методов, выбор уровня сложности). После выбора необходимого уровня сложности вызывается метод *newGame*, в котором выполняется отрисовка основного поля игрового процесса (добавление фона, кнопок, поля для ввода, игровой таблицы, обработка нажатия клавиш). По нажатию клавиши запускается метод *buttonController*, вызывающий соответствующий ей метод из класса **NewThread** и выполняется перерисвка игрового поля (*drowGrindPane*). В случае полного заполнения поля (*check\_lost*) или получение числа 2048 (*check\_won*), вызывается соответствующий метод для вывода сообщения о победе или проигрыше.

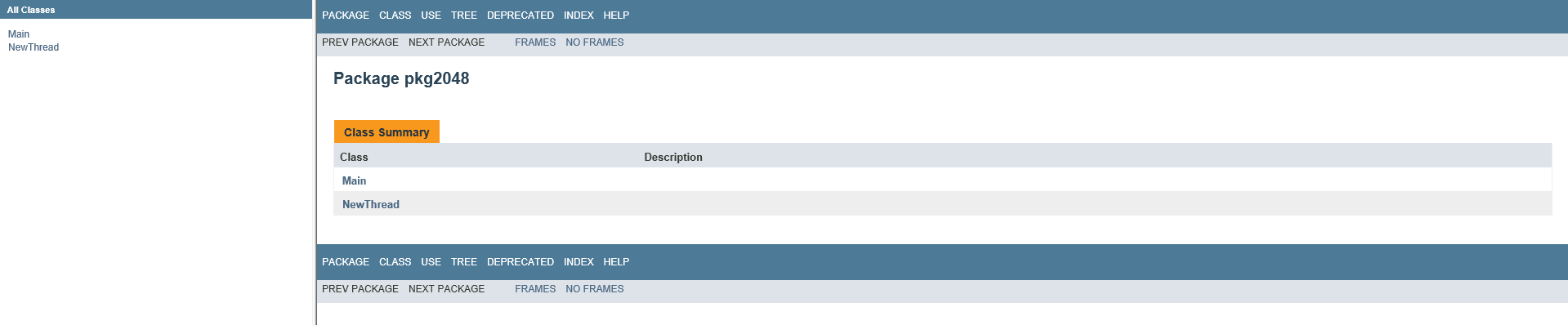
При загрузке сохранения из файла или вызове бота возникает необходимость задерживать отрисовку поля (для демонстрации пошагового действия). Для этой цели был создан класс **ThreadForReplay,** унаследованный от потока, единственный метод которого (*run*) вызывается в классе **Main** методом *autoGame*. Он при помощи таймера осуществляет sleep тем самым позволяя пользователю проверить правильность выполняемых действий.

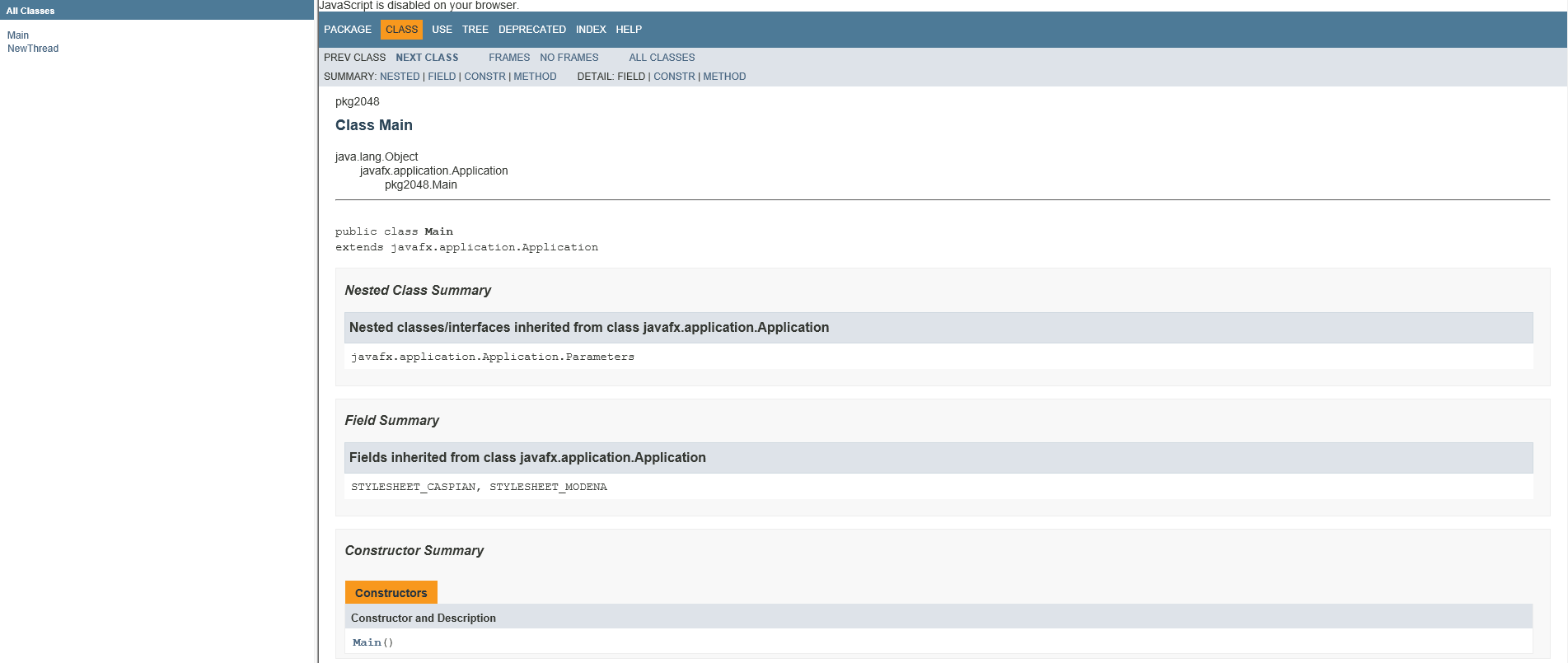
Класс **NewThread** отвечает за математическую составляющую приложения. Он наследуется от потока, что обеспечивает параллельность выполнения отрисовки и логики игры. В этом классе описаны методы, необходимые для пересчета игровой таблицы после действий пользователя. Эти методы вызываются в классе **Main** при обработке нажатий на клавиши «стрелок».

Перечисление **Direction** необходимо для упрощения понимания кода и аннотации.



рис. 3





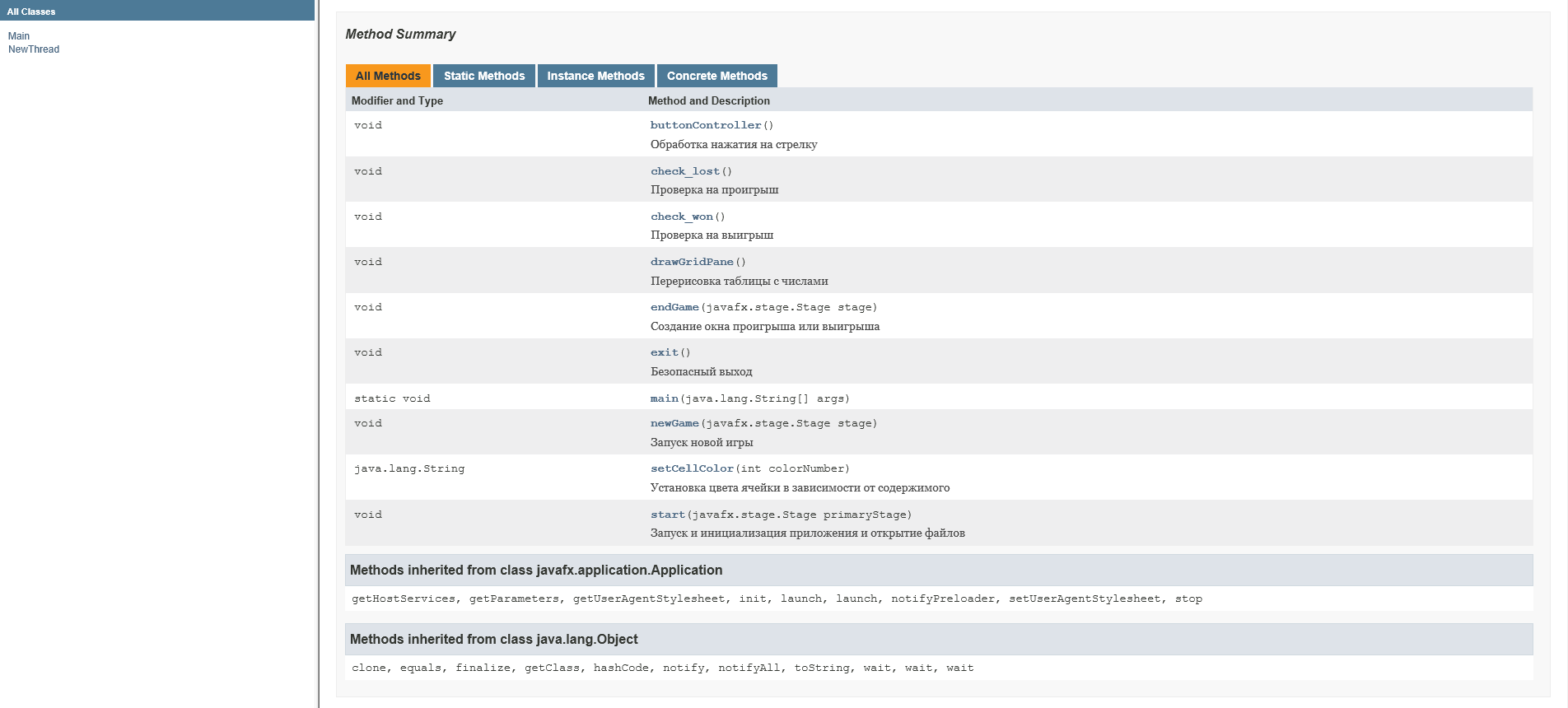
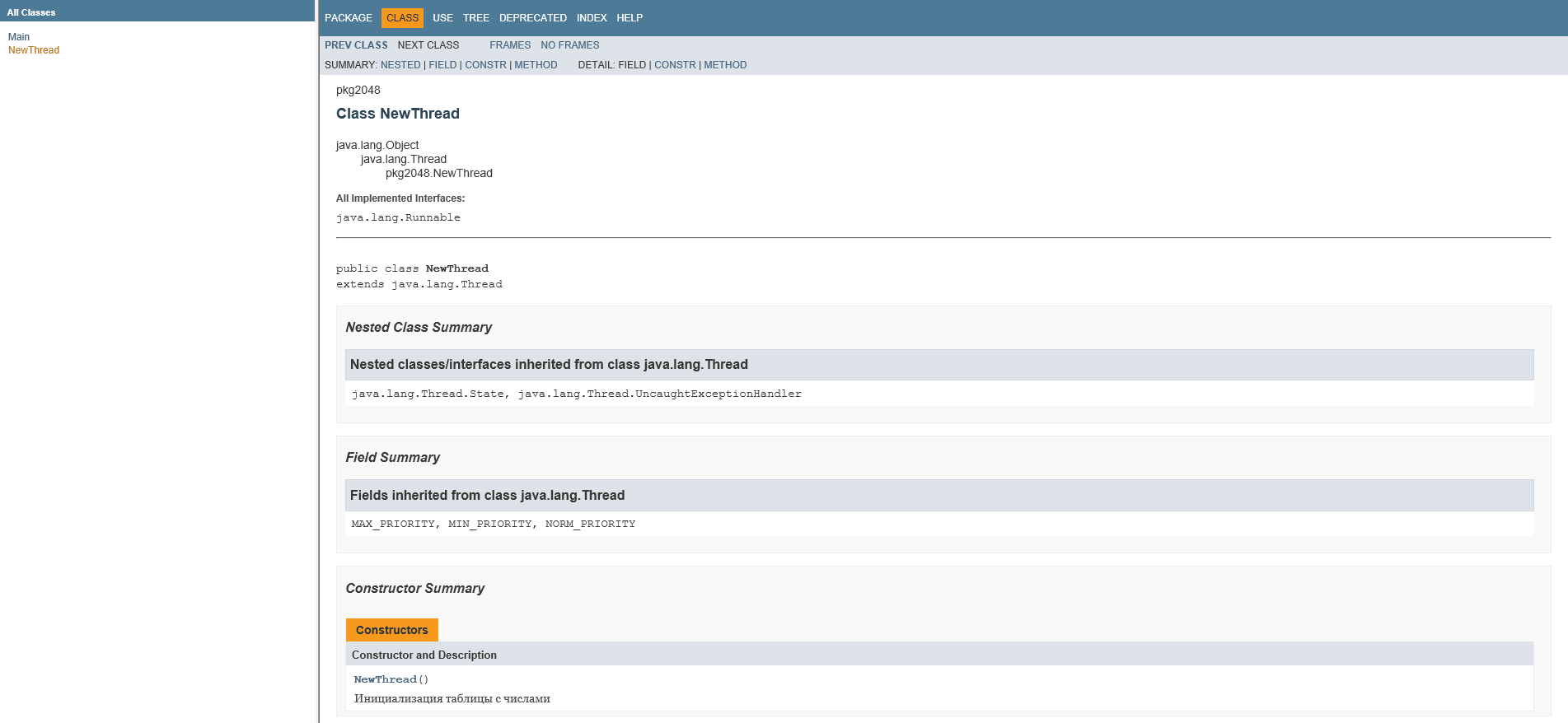


рис. 4



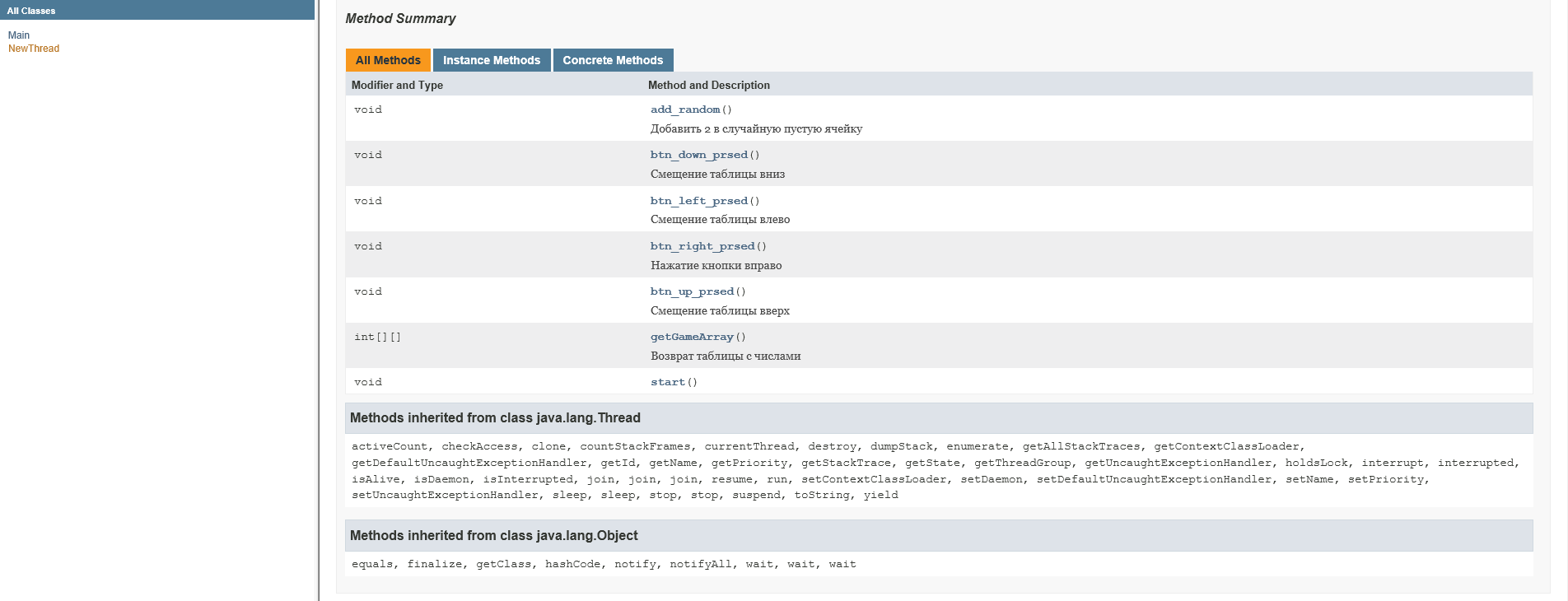


рис. 5

**Лабораторная работа №3. Разработка программ с использованием модульного подхода**

Для реализации сохранения и загрузки игры я использовала нотацию продемонстрированную в Таблице 1.

*Таблица 1.*

|  |  |  |
| --- | --- | --- |
| Клавиша | Интерпретация | Сохранение в файле |
| ↑ | UP | 0 |
| ↓ | DOWN | 1 |
| ← | LEFT | 2 |
| → | RIGHT | 3 |

Я считаю, что данная нотация упрощает реализацию приложения и является наиболее понятной.

Сохранение процесса игры я реализовала путем записи в файл каждого хода игрока\бота. К сожаление данный способ не может в полной мере повторить предыдущий прогресс, однако сохранение всех изменений поля стало бы по моему мнению излишне ресурсозатратным.

Сохранение происходит после нажатия кнопки «Exit». В случае если в нем нет необходимости, следует закрыть окно приложения на «Крестик».

**Лабораторная работа №4. Разработка многопоточных программ, использование стандартных примитивов синхронизации**