Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему

Разработка программы «Жизнь» на основе идеи клеточного автомата Джона Конвея

БГУИР КП 1-71 01 02 01 013 ПЗ

Студент группы 710102 Цвирко А. Ю.
Руководитель КП Куприянова Д. В.

Минск 2018

Оглавление

ВВЕДЕНИЕ

Данный курсовой проект посвящен разработке программы клеточного автомата «Жизнь» на основе идеи математика Джона Конвея.

Клеточные автоматы

Клеточный автомат — дискретная модель, изучаемая в математике, теории вычислимости, физике, теоретической биологии и микромеханике. Включает регулярную решётку ячеек, каждая из которых может находиться в одном из конечного множества состояний, таких как 1 и 0. Решетка может быть любой размерности. Для каждой ячейки определено множество ячеек, называемых окрестностью. К примеру, окрестность может быть определена как все ячейки на расстоянии не более 2 от текущей (окрестность фон Неймана ранга 2). Для работы клеточного автомата требуется задание начального состояния всех ячеек и правил перехода ячеек из одного состояния в другое. На каждой итерации, используя правила перехода и состояния соседних ячеек, определяется новое состояние каждой ячейки. Обычно правила перехода одинаковы для всех ячеек и применяются сразу ко всей решётке.

Основное направление исследования клеточных автоматов — алгоритмическая разрешимость тех или иных задач. Также рассматриваются вопросы построения начальных состояний, при которых клеточный автомат будет решать заданную задачу.

История игры «Жизнь» началась, когда Джон Конвей заинтересовался проблемой, предложенной в 1940-х годах известным математиком Джоном фон Нейманом, который пытался создать гипотетическую машину, которая может воспроизводить сама себя. Джону фон Нейману удалось создать математическую модель такой машины с очень сложными правилами. Конвей попытался упростить идеи, предложенные Нейманом, и в конце концов ему удалось создать правила, которые стали правилами игры «Жизнь».

Впервые описание этой игры было опубликовано в октябрьском (1970 год) выпуске журнала Scientific American, в рубрике «Математические игры» Мартина Гарднера (Martin Gardner).

Правила игры «Жизнь»

Место действия этой игры — «вселенная» — это размеченная на клетки поверхность или плоскость — безграничная, ограниченная, или замкнутая (в пределе — бесконечная плоскость).

1. Каждая клетка на этой поверхности может находиться в двух состояниях: быть «живой» (заполненной) или быть «мёртвой» (пустой). Клетка имеет восемь соседей, окружающих её.
2. Распределение живых клеток в начале игры называется первым поколением. Каждое следующее поколение рассчитывается на основе предыдущего по таким правилам:

   2.1. в пустой (мёртвой) клетке, рядом с которой ровно три живые клетки, зарождается жизнь;

   2.2. если у живой клетки есть две или три живые соседки, то эта клетка продолжает жить; в противном случае, если соседей меньше двух или больше трёх, клетка умирает («от одиночества» или «от перенаселённости»).

3. Игра прекращается, если:

   3.1. на поле не останется ни одной «живой» клетки;

   3.2. конфигурация на очередном шаге в точности (без сдвигов и поворотов). повторит себя же на одном из более ранних шагов (складывается периодическая конфигурация)

   3.3. при очередном шаге ни одна из клеток не меняет своего состояния (складывается стабильная конфигурация; предыдущее правило, вырожденное до одного шага назад).

Эти простые правила приводят к огромному разнообразию форм, которые могут возникнуть в игре.

Игрок не принимает прямого участия в игре, а лишь расставляет или генерирует начальную конфигурацию «живых» клеток, которые затем взаимодействуют согласно правилам уже без его участия (он является наблюдателем).

Также в данной программе предусмотрена принадлежность к клетке свойства «цвет», что позволяет каждой из соседних клеток влиять на цвет клеток нового поколения.

Фигуры

Вскоре после опубликования правил было обнаружено несколько интересных шаблонов (вариантов расстановки живых клеток в первом поколении), в частности: r-пентамино и планер (glider).

Некоторые такие фигуры остаются неизменными во всех последующих поколениях, состояние других периодически повторяется, в некоторых

случаях со смещением всей фигуры. Существует фигура (Diehard) всего из семи живых клеток, потомки которой существуют в течение ста тридцати поколений, а затем исчезают.

Конвей первоначально предположил, что никакая начальная комбинация не может привести к неограниченному размножению и предложил премию в 50 долларов тому, кто докажет или опровергнет эту гипотезу. Приз был получен группой из Массачусетского технологического института, придумавшей неподвижную повторяющуюся фигуру, которая периодически создавала движущиеся «планеры». Таким образом, количество живых клеток могло расти неограниченно. Затем были найдены движущиеся фигуры, оставляющие за собой «мусор» из других фигур.

К настоящему времени более-менее сложилась следующая классификация фигур:

1. Устойчивые фигуры: фигуры, которые остаются неизменными.
2. Долгожители: фигуры, которые долго меняются, прежде чем стабилизироваться.
3. Периодические фигуры: фигуры, у которых состояние повторяется через некоторое число поколений.
4. Двигающиеся фигуры: фигуры, у которых состояние повторяется, но с некоторым смещением.
5. Ружья: фигуры, у которых состояние повторяется, но дополнительно появляется двигающаяся фигура.
6. Паровозы: двигающиеся фигуры, которые оставляют за собой следы в виде устойчивых или периодических фигур.
7. Пожиратели: устойчивые фигуры, которые могут пережить столкновения с некоторыми двигающимися фигурами.
8. Фигуры, которые при столкновении с некоторыми фигурами дублируются.

Кроме того, многие закономерности, обнаруженные в игре, имеют свои аналогии в других, подчас совершенно «нематематических» дисциплинах. Вот список наук, теории которых имеют интересные точки соприкосновения с феноменами «Жизни»:

1. Кибернетика. Сама игра является удачной попыткой Конвея доказать существование простых самовоспроизводящихся систем, а также появление некоего «разума» у самовоспроизводящихся систем.
2. Биология. Внешнее сходство с развитием популяций примитивных организмов впечатляет.

3. Бактериология. Некоторые интересные вариации игры с дополнительными условиями могут с точностью повторить размножение бактерий, которые с случайной вероятностью могут мутировать (по условию модификации) и уже новый вид начинает заполнять всё пространство. Данный механизм с точностью повторяет эволюцию бактерий, которые в конце концов эволюционировали до человека.
4. Физиология. Рождение и смерть клеток аналогичны процессу возникновения и исчезновения нейронных импульсов, которые и формируют процесс мышления. А также аналогичны созданию импульсов в нервной системе многоклеточных организмов.
5. Астрономия. Эволюции некоторых сложных колоний удивительным образом схематично повторяют этапы развития спиралевидных галактик.
6. Физика твёрдого тела. Теория автоматов вообще и игра «Жизнь» в частности используются для анализа «явлений переноса» — диффузии, вязкости и теплопроводности.
7. Квантовая физика. Поведение «жизненных» ячеек (рождение новых и взаимное уничтожение) во многом напоминают процессы, происходящие при столкновении элементарных частиц.
8. Наномеханика. Стационарные и пульсирующие колонии являются показательным примером простейших устройств, созданных на основе нанотехнологий.
9. Электротехника. Правила игры используются для моделирования самовосстанавливающихся электрических цепей.
10. Химия. Конфигурации, подобные строящимся в игре, возникают во время химических реакций на поверхности, в частности в опытах М. С. Шакаевой возникают движущиеся молекулярные конструкции аналогичные «жизненному» планеру. Также предпринимаются попытки объяснить периодические химические реакции с помощью многомерных клеточных автоматов. Самоорганизацией элементарных частиц также занимается супрамолекулярная химия.
11. Социология. Процессы доминации, вытеснения, поглощения, сосуществования, слияния и уничтожения популяций во многих аспектах схожи с явлениями, происходящими при взаимодействии больших, средних и малых социальных групп.
12. Философия. Приведённый список примеров в очередной раз подтверждает, что всё во Вселенной развивается по одним и тем же нескольким фундаментальным законам.
13. Возможно, эта игра связана и с другими научными явлениями, в том числе и с теми, о которых современной науке пока неизвестно.

Также возможно, что не открытые на сегодня законы Природы и Общества станут более понятными благодаря «Жизни» и её модификациям.

1. СПИСОК ИСПОЛЬЗУЕМЫХ ОБОЗНАЧЕНИЙ

1. ПО – программное обеспечение.
2. Вселенная – игровая единица, которая представляет из себя матрицу значений и является совокупностью живых и мертвых клеток разом. По ней определяется является ли клетка живой или мертвой и её цвет.
3. Мертвая клетка – игровая единица, которая имеет значение -1 во Вселенной.
4. Живая клетка – игровая единица, имеющая значение в игровой матрице отличное от значения мертвой клетки (-1).
5. Поколение – совокупность живых и мертвых клеток, находящихся во Вселенной и имеющие одинаковый цвет и одинаковое название (Внимание! Все клетки на поле имеют обязательно уникальное имя и цвет). После начала игры, модифицированные клетки (клетки изменившие градиент цвета) уже больше не принадлежат поколению.
6. Сигнал – специальная функция уведомления о том, что произошло какое-либо событие.
7. Слот – специальная функция, которая может выступать в роли обработчика события и реагировать на сигналы.
8. Дамп – структура данных представляющая собой блок значений матрицы Вселенной.
9. Game of Life (GoL) – название игры «Жизнь» на английском языке.
10. Фича – (англ. feature — особенность, необычное свойство) — сленговое обозначение каких-либо необычных признаков какого-либо явления. Иным словом понятие «фича» в обычной речи заменяют созвучным «фишка».
11. Баг – жаргонное слово, обозначающее ошибку или неправильное поведение программы. Довольно часто путается пользователями со словом «фича».
12. RGB (англ. – red green blue) – аддитивная цветовая модель, описывающая способ кодирования цвета с помощью 3 цветов, которых принято называть основными.

2. ОБЗОР ЛИТЕРАТУРЫ

[1]. Andrew Adamatzky. Game of Life Cellular Automata.

Данное издание объединяет результаты сорокалетних исследований по вычислительным, математическим, физическим и инженерным аспектам клеточных автоматов Game of Life. Отдельные темы включают феноменологию и статистическое поведение; пространственно-временная динамика на тиллинговых и гиперболических пространствах Пенроуза; поколение музыки; алгебраические свойства; моделирование финансовых рынков; полуквантовые расширения; прогнозирование появления; анализ на основе двух графов; нечеткое, предельное поведение и пороговое масштабирование; разработка правил перехода между клетками и состояниями; динамика локализации в квазихимических аналогах GoL; самоорганизация в сторону критичности; асинхронные реализации. Объем уникален, потому что он дает исчерпывающее представление о теоретических и экспериментальных основах, передовых вычислительных методах и математическом анализе невероятно сложных, самоорганизующихся и возникающих явлений, определяемых невероятно простыми правилами.

В книге отлично описываются правила клеточного автомата «Жизнь», приведено множество примеров алгоритмов-комбинаций, которые приводят к неограниченному размножению. Также в книге описаны различные модификации первоначальной игры «Жизнь» Джона Конвея.

[2]. Paul Rendell. Turing Machine Universality of the Game of Life.

Данный материал представляет собой доказательство универсальных вычислений в клеточном автомате Game of Life с использованием конструкции машины Тьюринга. В нем содержится вводная информация, включая справочную информацию и расширенный обзор литературы по машинам Тьюринга, встречным машинам и соответствующим шаблонам в игре жизни Конвея, так что предмет доступен для неспециалистов.

Книга содержит описание авторской машины Тьюринга в Conway's Game of Life, в том числе ленту неограниченного хранения, обеспечиваемую растущими структурами стеков, а также представляет собой быструю универсальную машину Тьюринга, предназначенную для демонстрации работы в удобное время.

[3]. Bjarne Stroustrup Tour to C++.

В книге автор, создатель языка C++, описывает, что составляет современный язык C++. Это краткое, автономное руководство охватывает большинство основных языковых функций и основные компоненты стан-

дартной библиотеки - конечно, не очень подробно, но на уровне, который дает программистам значимый обзор языка, некоторые ключевые примеры и практическую помощь в начале.

Страуструп представляет функции C ++ в контексте поддерживаемых ими стилей программирования, таких как объектно-ориентированное и универсальное программирование. Его тур поразительно всеобъемлющий. Охват начинается с основ, а затем широко охватывает более сложные темы, включая многие новые в C ++ 17, такие как семантика перемещения, равномерная инициализация, лямбда-выражения, улучшенные контейнеры, случайные числа и параллелизм. Тур даже охватывает некоторые расширения, создаваемые для C ++ 20, такие как концепции и модули, и заканчивается обсуждением дизайна и развития языка C ++.

[4]. Bjarne Stroustrup Principle and Practice Using C++.

Книга объясняет фундаментальные концепции и методы более глубоко, чем традиционные введения. Этот подход даст вам прочную основу для написания полезного, правильного, поддерживаемого и эффективного кода.

Книга представляет собой введение в программирование в целом, включая объектно-ориентированное программирование и общее программирование. Это также хорошее введение в язык программирования C ++, один из наиболее широко используемых языков для реального программного обеспечения. В книге с самого начала представлены современные методы программирования на C ++, а также представлена стандартная библиотека C ++ для упрощения задач программирования.

В книге предполагается, что в конечном итоге вы нацелены на написание нетривиальных программ, будь то работа в разработке программного обеспечения или в какой-либо другой технической области.

[5]. Chad Fowler Clean Code.

Данная книга преподаст вам основы и принципы, которых придерживаются сеньор-разработчики при разработке программного обеспечения. Книга позволит улучшить читаемость и «чистоту» кода.
Чистый код делится на три части. Первый описывает принципы, шаблоны и методы написания чистого кода. Вторая часть состоит из нескольких тематических исследований возрастающей сложности. Каждое тематическое исследование - это упражнение по очистке кода - преобразование кодовой базы, имеющей некоторые проблемы, в надежную и эффективную. Третья часть - это результат: отдельная глава, содержащая список эвристик, собранных при создании тематических исследований. Результатом является база знаний, которая описывает, как мы думаем, когда пишем, читаем и очищаем код.

[4]. Max Shclee Qt5 Proffesional Programming with C++.

Данный мануал рассказал мне о программировании на фреймворке Qt и разработке программного обеспечения в среде программирования Qt Creator, познакомит вас с основами и надстройками над языком C++.
В этой книге вы найдете целый ряд приемов создания программ для операционных систем Windows, Mac OS X, Linux и мобильных приложений для Android, iOS с использованием библиотеки Qt, версии 5.10 и публикации приложений на Google Play и Apple App Store. В книге подробно рассматриваются интересные возможности этой библиотеки, которые выгодно отличают ее от различных библиотек для C++. Книга описывает интегрированную среду разработки приложений "Qt Creator", создание форм с помощью программы "Qt Designer", работу с технологией "Qt Quick", "Qt Quick Controls", "Qt Quick Controls 2", программирование 2D- и 3D-графики, возможности мультимедиа, модуль "Qt WebEngine" для создания веб-приложений, программирование баз-данных, создание многопоточных программ, использование XML, QML, JavaScript и много другого.

3. СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

Программа написана на языке C++ с использованием возможностей фреймворка Qt версии 5 и выше, с использованием объектно-ориентированной парадигмы программирования. Программа состоит из двух классов: MainWindow и GameWidget. Класс MainWindow отвечает за работу главного окна и содержит указатель на объект GameWidget. GameWidget отвечает за работу виджета и основной логики программы.

3.1 Класс MainWindow

Наследует класс QMainWindow, который в свою очередь наследуется от одного из главных классов Qt – QObject и содержит множество методов для работы с графическим интерфейсом приложений и позволяет нам использовать именитую сигнально-слотовую систему Qt, и состоит из:

1. конструктора класса, который при инициализации связывает множество сигналов из графического интерфейса с методами класса, инициализирует и устанавливает множество свойств и параметров графического интерфейса.
2. методов saveGame(), loadGame(), которые позволяют загружать и сохранять состояния игры.
3. методов addNewGeneration(), renameGeneration(), deleteGeneration() которые в процессе создания класса привязываются к специальным функциям, называемыми сигналами(более усложненный механизм callback-функций) и позволяют обрабатывать сигналы кнопок из графического интерфейса. Данные функции позволяют добавлять, переименовывать и удалять поколения. После начала игры, модифицированные клетки (клетки изменившие градиент цвета) уже больше не принадлежат поколению и могут быть удалены с поля только нажатием в графическом интерфейсе игры кнопки clear.
4. метода setGenerationHints(), который позволяет при нажатии кнопкой мыши или выбору с помощью стрелок вверх/вниз на клавиатуре выводить значения полей поколения в графический интерфейс.
5. метода selectMasterColor(), который позволяет выбирать цвет из имеющихся цветов поколения для дальнейшего рисования им клеток.
6. метода messageBoxWarningExec(const QString&, const QString&), который принимает 2 параметра-строки, один — заголовок, второй — текст сообщения, который мы хотим показать пользователю в графическом интерфейсе. В общем, эта функция служит для показа

различного рода предупреждений и подсказок пользователю, играющему в игру.

7. указателя на xml-документ, описывающий все особенности и свойства графического интерфейса, парсящийся и загружаемый движком ранее описанного класса QCoreApplication с помощью метода setupUi(QWidget*).
8. указателя на основной класс логики программы GameWidget. Об этом классе поподробнее будет написано в следующем заголовке.
9. деструктора, который высвобождает выделенную в конструкторе память и возвращает ресурсы операционной системе.

3.2 Класс GameWidget

Наследуется от класса QWidget, который в свою очередь наследуется всё от того же QObject, который позволяет использовать сигнально-слотовую систему и множество наследуемых функций от этих классов и состоит из:

1) конструктора, в котором задаются начальные параметры игры, например, создается матрица (поле, которое видит игрок), инициализируются контейнеры для хранения цветов поколений, задаются значения отступов, для того, чтобы матрица всегда была в центре экрана и имела не прямоугольные ячейки, а квадратные.
2) переопределенной виртуальной функции-ивента paintEvent(QPaintEvent*), которая отвечает за низкоуровневое рисование на виджете, перерисовывает Вселенную и рисует границы, вызывая функции paintUniverse(QPainter&) и paintGrid(QPainter&) соответственно.
3) переопределенной виртуальной функции-ивента resizeEvent(QResize Event*), которая отвечает за пересчет отступов при изменении размера окна программы.
4) переопределенных функций-ивентов mousePressEvent(QMouseEvent*) и mouseMoveEvent(QMouseEvent*), которые обрабатывают исключение нажатия кнопок вне интерфейса Вселенной, имитируют сигнал об изменении во Вселенной для блокировки некоторых функций интерфейса и вызывает метод countSellAndUpdateIt(const int&, const int&, const bool) для подсчета и внесения изменений в матрицу Вселенной.
5) функций-слотов startGame(), nextGeneration(), stopGame(), clear(), которые позволяют обрабатывать нажатия на соответсвующие

кнопки start, next, stop и clear и выполнять соответсвующие действия.

6) функций-слотов setCellNumber(const int&), setInterval(const int&), setBackgroundColor(const QColor&), которые подвязываются к соответствующим сигналам и позволяют изменять значения размера Вселенной, значение интервала и цвета фона.
7) функций cellNumber(), interval(), getMasterColor(), setMasterColor(), getBackgroundColor(), которые возвращают значения количества клеток на поле, интервал между генерацией новых поколений.
8) функций dump, setDump(const int*), которые позволяют считывать дамп и заносить значения в матрицу и составлять дамп для записи на жесткий диск.
9) функций-слотов paintGrid(QPainter &), paintUniverse(QPainter &), которые отвечают за отрисовку границы каждой из клеток и самой Вселенной соответственно.
10) функции-слота newGeneration(), который вызывает для каждой клетки функцию isAlive(int k, int j) и выполняет проверки на окончание игры.
11) функции invertBackgroundColor(), которая инвертирует значение цвета заднего фона в режиме HSV для лучшей видимости на фоне различных компонентов графического интерфейса.
12) функций convertQColorToInt(const QColor&) и convertIntToQColor(const int&) для многочисленных преобразований цвета в формате RGB и хранении его в переменной типа int и наоборот из переменной типа int в значение цвета в формате RGB.

4. ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

Рассмотрим основные компоненты программы – игры «Жизнь».

Программа начинается с системного вызова main, который создает процесс и регистрирует его на уровне операционной системы. Далее создается
объект класса QApplication для управления ходом выполнения программы и его ресурсов, создается объект нашего класса MainWindow, который отвечает за создание и начальную инициализацию графического интерфейса приложения и его элементов, вызывается его метод show, который уже выводит графический интерфейс приложения на экран. Следующим действием мы вызываем метод exec() класса QCoreApplication, который зацикливает обработку последующих сигналов и ивентов из интерфейса приложения.

```
int main(int argc, char *argv[])
{
QApplication a(argc, argv);
MainWindow w;
w.show();

return a.exec();
}
```

Далее рассмотрим реализацию конструктора класса MainWindow. Сначала в конструктор базового класса передается указатель на родителя, т.е. на главное окно приложения, однако эта функция принимает значение по умолчанию равное nullptr, что соответствует тому, что оно само является главным. На основе xml-документа mainwindow.ui создается объект класса MainWindow и устанавливает его в качестве основного графического интерфейса окна. Далее создаются пиксельные карты. И на основе их создаются объекты иконок. Создаются начальные items в listwidget и привязывается огромное количество сигналов к соответствующим слотам. Также создается объект класса GameWidget – основной логики программы и добавляется на экран в gameLayout.

```
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow),
    game(new GameWidget(this))
{
    ui->setupUi(this);

    QPixmap pixmap(16, 16);      //create pixmap
```

```
    QIcon icon;                  //create icon object

    pixmap.load(":/images/next-button", ".jpg");
                      // load it and use for next button
    icon.addPixmap(pixmap);
    ui->nextButton->setIcon(icon); // and set

    pixmap.fill(Qt::black);
                      // reuse pixmap for color button icon
    icon.addPixmap(pixmap);
    ui->colorButton->setIcon(icon);

    ui->listWidget->addItem("default generation");
// add default generation
    ui->listWidget->setCurrentItem(ui->listWidget->item(0));
// selection flags at designer
    ui->listWidget->currentItem()->setIcon(icon);

    pixmap.fill(Qt::gray);       // reuse for background color
    icon.addPixmap(pixmap);
    ui->backgroundColor->setIcon(icon);

    // connect signals
    connect(ui->startButton, SIGNAL(clicked()), game,
SLOT(startGame()));
    connect(ui->startButton, SIGNAL(clicked(bool)), ui-
>deleteItemButton, SLOT(setDisabled(bool)));
    connect(ui->nextButton, SIGNAL(clicked(bool)), ui-
>deleteItemButton, SLOT(setDisabled(bool)));
    connect(ui->nextButton, SIGNAL(clicked()), game,
SLOT(nextGeneration()));
    connect(ui->stopButton, SIGNAL(clicked()), game,
SLOT(stopGame()));
    connect(ui->clearButton, SIGNAL(clicked()), game,
SLOT(clear()));
    connect(ui->iterInterval, SIGNAL(valueChanged(int)), game,
SLOT(setInterval(int)));
    connect(ui->cellsControl, SIGNAL(valueChanged(int)), game,
SLOT(setCellNumber(int)));
    connect(game, SIGNAL(environmentChanged(bool)),ui-
>cellsControl, SLOT(setDisabled(bool)));
    connect(game, SIGNAL(gameEnds(bool)),ui->cellsControl,
SLOT(setEnabled(bool)));
    connect(game, SIGNAL(gameEnds(bool)),ui->deleteItemButton,
SLOT(setEnabled(bool)));
    connect(ui->colorButton, SIGNAL(clicked()), this,
SLOT(selectMasterColor()));
    connect(ui->backgroundColor, SIGNAL(clicked()), this,
SLOT(setBackgroundColor()));
```

```
    connect(ui->saveButton, SIGNAL(clicked()), this,
SLOT(saveGame()));
    connect(ui->loadButton, SIGNAL(clicked()), this,
SLOT(loadGame()));
    connect(ui->addItemButton, SIGNAL(clicked()), this,
SLOT(addNewGeneration()));
    connect(ui->renameItemButton, SIGNAL(clicked()), this,
SLOT(renameGeneration()));
    connect(ui->deleteItemButton, SIGNAL(clicked()), this,
SLOT(deleteGeneration()));
    connect(ui->listWidget, SIGNAL(itemSelectionChanged()),
this, SLOT(setGenerationHints()));

    ui->gameLayout->addWidget(game);
// and add new widget
}
```

Рассмотрим конструктор класса GameWidget. В конструктор передается параметр класса QWidget, который принимает указатель на родителя, далее создается объект класса QTimer, указателям на матрицу присваивается значение nullptr, что соответствует нулевому указателю и является хорошей практикой программирования, задается начальное значение длины и ширины матрицы Вселенной. Присваиваются начальные цвета, создаются контейнеры для хранения цветов, задаются начальные значения state-переменных, используемые в обработчиках кнопок нажатия мыши. Привязывается сигнал объекта таймера к обработчику срабатывания сигнала.

```
GameWidget::GameWidget(QWidget *parent) :
    QWidget(parent),
    timer(new QTimer(this)),
    universe(nullptr),
    next(nullptr),
    universeSize(50)
{
    m_masterColor = Qt::black;    //set default color
    m_backgroundColor = Qt::gray; //set default background
    timer->setInterval(300);      //set default timer interval
    generationColor.push_back(m_masterColor);
                            //set first color black by default
    prevSelectedCellX = -1;
    prevSelectedCellY = -1;
    connect(timer, SIGNAL(timeout()), this,
SLOT(newGeneration()));
    resetUniverse();
            //set current and next universes and fill them
}
```

Довольно интересной частью программы является система, которая позволяет при изменении окна изменять масштаб клеток, границ и в общем всей Вселенной.

```
void GameWidget::countMargins()
{
    int w = width();
    int h = height();

    if(w < h)
    {
        gridSize = w / universeSize * universeSize;
    } else
    {
        gridSize = h / universeSize * universeSize;
    }
    widthMargins = (w - gridSize) / 2;
    heightMargins = (h - gridSize) / 2;
}
```

Одним из самых интересных фич приложения является система добавления новых поколений, т.е. клеток, которые имеют общее название и цвет.

```
     void MainWindow::addNewGeneration()
     {
         bool ok;
         QString text = QinputDialog::getText(this,
                            tr("Add generation!\n"),
                            tr("New generation name:"),
                            QLineEdit::Normal,
                            QDir::home().dirName(), &ok);
         if (!ok) return;
// if it's not ok
         if(text.isEmpty())
// it cannot be empty string
         {
             messageBoxWarningExec("Warning", "Please, name
generation!");
             return;
         }

         for(int i = 0; i < ui->listWidget->count();i++)
// search in listview for such generations
         {
             if(ui->listWidget->item(i)->text() == text)
             {
```

```
                messageBoxWarningExec("Warning", "Such
generation exists!");
                return;
            }
        }

        QColor color;
// now choose color for generation
        while(1)
        {
            bool isColorValid = true;
            color = QColorDialog::getColor(game-
>getMasterColor(), this, "Select color for such generation");
            if(!color.isValid())
            {
                messageBoxWarningExec("Warning", "Color is
not valid! Try again!");
                continue;
            }

            for(int i = 0; i < game->generationColor.size();
i++)                                    //search in listview
for such generations
            {
                if(color == game->generationColor.at(i))
                {
                    messageBoxWarningExec("Warning", "Such
color exists!\nYou can't have TWO same colors at one time!\
nChoose new color!");
                    isColorValid = false;
                    break;
                }
            }

            if(isColorValid) break;
        }

        ui->listWidget->addItem(text);
// add new item
        ui->listWidget->setCurrentItem(ui->listWidget-
>item(ui->listWidget->count()-1));  //set new current

        QPixmap pixmap(16, 16);
// create pixmap for item icon
        pixmap.fill(color);
        QIcon icon(pixmap);
// fill pixmap with color
        ui->listWidget->currentItem()->setIcon(icon);
// set new icon of black color
```

```
        ui->colorButton->setIcon(icon);
        game->setMasterColor(color);
// set new master color
        game->generationColor.push_back(color);
// color by default push into vector
    }
```

Также стоит отметить очень интересное нововведение – уникальное смешивание цветов клеток при образовании нового поколения, что, несомненно, является проекцией на реальный мир и размножение организмов.

Этот механизм реализован с помощью следующей функций, которая является посредником, считает «силу» клетки, которая определяется из находящихся рядом клеток и собирает градиенты-значения цветов в формате RGB для дальнейшего подсчета каждого из цветов.

```
void GameWidget::givePowerAndExtractColors(int& power, int&
red, int& green, int& blue, const int& k, const int& j)
{
    power++;
    red += extractRed(universe[k*universeSize + j]);
    green += extractGreen(universe[k*universeSize + j]);
    blue += extractBlue(universe[k*universeSize + j]);
}
```

И с помощью следующих трех методов мы можем доставать нужные нам цвета для подсчета из матрицы Вселенной.

```
int GameWidget::extractRed(const int& color)
{
   return (color >> 16) & 0xff;
}

int GameWidget::extractGreen(const int& color)
{
   return (color >> 8) & 0xff;
}

int GameWidget::extractBlue(const int& color)
{
   return color & 0xff;
}
```

А с помощью данного метода мы рассчитываем состояние клетки в следующем поколении, т.е. рассчитываем такие параметры как «сила» клетки,

которая, напомню, рассчитывается на основе рядом стоящих клеток. И если клетка является живой, тогда рассчитываем её цвет.

```
int GameWidget::isAlive(int k, int j)
{
    int power = 0, r = 0, g = 0, b = 0; // power red green
blue
    // give some power to cell and extract colors
    if(universe[(k+1)*universeSize + j] != -1)
givePowerAndExtractColors(power, r, g, b, k + 1, j);
    if(universe[(k-1)*universeSize + j] != -1)
givePowerAndExtractColors(power, r, g, b, k - 1, j);
    if(universe[k*universeSize + (j+1)] != -1)
givePowerAndExtractColors(power, r, g, b, k, j + 1);
    if(universe[k*universeSize + (j-1)] != -1)
givePowerAndExtractColors(power, r, g, b, k, j - 1);
    if(universe[(k+1)*universeSize + (j-1)] != -1)
givePowerAndExtractColors(power, r, g, b, k + 1, j - 1);
    if(universe[(k-1)*universeSize + (j+1)] != -1)
givePowerAndExtractColors(power, r, g, b, k - 1, j + 1);
    if(universe[(k-1)*universeSize + (j-1)] != -1)
givePowerAndExtractColors(power, r, g, b, k - 1, j - 1);
    if(universe[(k+1)*universeSize +  (j+1)] != -1)
givePowerAndExtractColors(power, r, g, b, k + 1, j + 1);
    if (((universe[k*universeSize + j] != -1) && (power == 2))
|| (power == 3))
    {
        r = r / power;          //count average color from
relatives
        g = g / power;
        b = b / power;
        QColor color(r, g, b);
        return convertQColorToInt(color);
    }
    return -1;
}
```

5. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Программа написана на языке C++ с использованием возможностей фреймворка Qt версии 5 и выше, с использованием объектно-ориентированной парадигмы программирования и основных принципов хорошего стиля программирования. Qt – это кроссплатформенный фреймворк для разработки программного обеспечения на различных языках программирования, поэтому программа может быть скомпилирована и запущена на всех популярных операционных системах, таких как: Windows, Linux, Mac OS, Android. Также в папке с проектом находится файл «conway.exe», который является исполняемым файлом в операционной системе Windows.

5.1 Минимальные системные требования

Минимальные системные требования не высоки в связи с тем, что программа написана и оптимизирована на языке C++, который сам по себе является очень ресурс-экономным языком программирования. Итак, для запуска проекта на целевой машине вам потребуется:

1. Операционная система Windows XP.
2. Процессор Intel Pentium IV.
3. Оперативная память 512 мегабайт.
4. Место на жестком диске 50 мегабайт.
5. Наличие мыши, тачпада или другого устройства ввода.

5.2 Структура проекта

После скачивания программы себе на компьютер и зайдя в основную папку проекта вы увидите большое количество файлов с различными расширениями и названиями, однако не пугайтесь, так как все эти файлы нужны лишь для корректной работы программы. Итак, программа состоит из:

1. проектного файла с расширением (.pro). Он содержит всю информацию, нужную применяемой системой сборки qmake, чтобы собрать приложение, библиотеки, плагины, подключаемые в проект файлы, конфигурации, ресурсные файлы и дополнительно подключаемые модули к приложению.
2. файла main с расширением (.cpp), который в свою очередь определяет точку входа в программу, используя системный вызов main.

3. класса MainWindow, который предназначен для работы с функциями графического интерфейса и состоит из 2 частей:
   3.1. mainwindow.h – содержит объявление класса для работы с графичеким интерфейсом программы.
   3.2. mainwindow.cpp – содержит реализацию всех методов класса, объявленных в одноименном заголовочном файле.
   3.3. mainwindow.ui – содержит xml-документ, который описывает графический интерфес приложения.
4. класса GameWidget, который отвечает за всю логику приложения и состоит из следующих частей:
   4.1. gamewidget.h – содержит объявление методов и полей класса для работы с логикой выполнения приложения.
   4.2. gamewidget.cpp — содержит реализацию всех методов класса, объявленных в одноименном заголовочном файле.
5. ресурсного файла с расширением (.qrc), который позволяет вне зависимости от платформы, в которой используется приложение, подключать к проекту различные нужные файлы, например, иконки и изображения.
6. папки maps, в которой располагаются предустановленные карты для начальной постановки параметров приложения. Также в эту папку можно сохранять текущие состояния игры, а потом их успешно загружать. Все файлы в папке имеют свои собственный формат и расширение, которое ассоциируется с этим приложением(.life).
7. в связи с тем, что мною была выполнена динамическая сборка приложения, нужны различные dll-файлы и библиотеки для успешной работы компилятора, Qt модуля и его компонентов. Все эти файлы находятся в папке с проектом.

5.3 Запуск приложения

Для успешного запуска приложения необходимо отыскать папку с проектом, открыть, найти там файл с названием «conway» с расширением .exe, подождать запуска программы и начать работу:

1. При запуске приложения появится вот такое окошко как на рис 1.

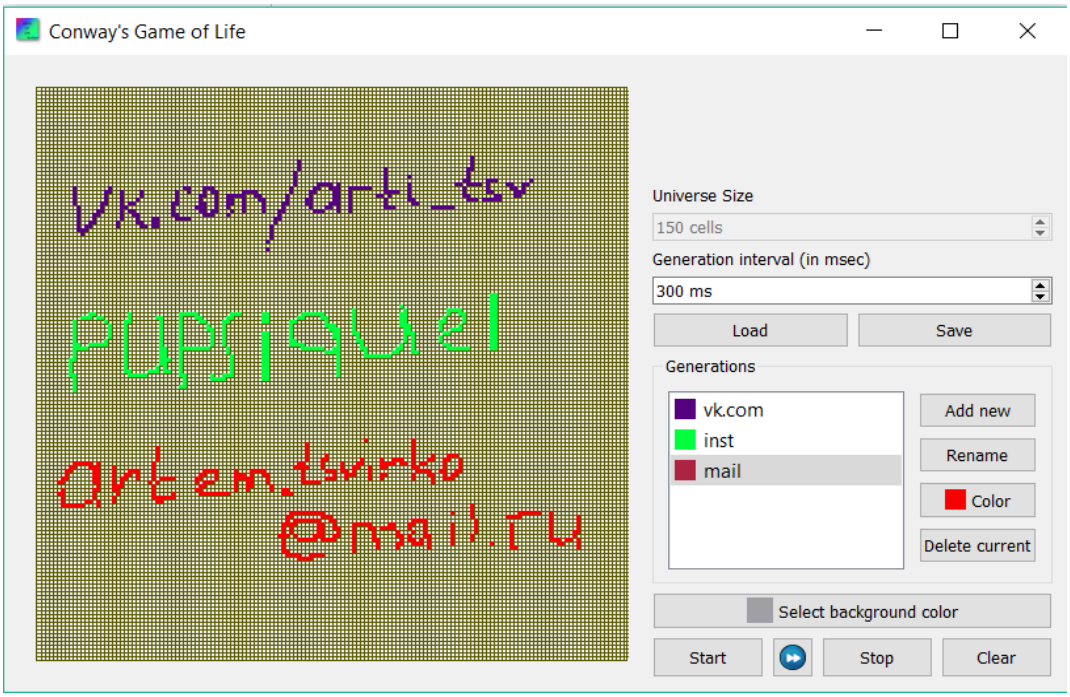

Рисунок 1. Главный графический интерфейс приложения.

2. Чтобы начать «оживлять» клетки – нажмите левую или правую кнопку мыши, проведите, если хотите создать несколько клеток сразу.
3. На лист виджете показывается какое поколение вы сейчас «наносите» на холст. Также там показаны все, созданные вами поколения.

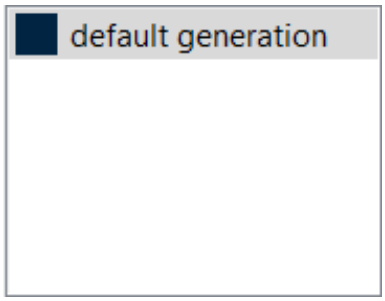

Рисунок 2. Окно поколений.

4. Чтобы создать новое поколение нажмите кнопку «Add new» и заполните соответствующие поля. И помните, что все последующие поколения должны иметь отличные от имеющихся цвета и имена.

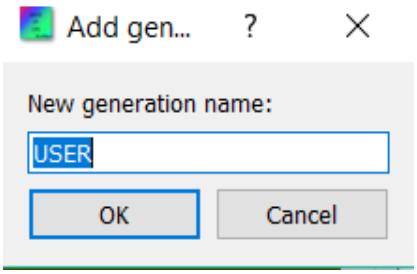

Рисунок 3. Окно добавления поколения.

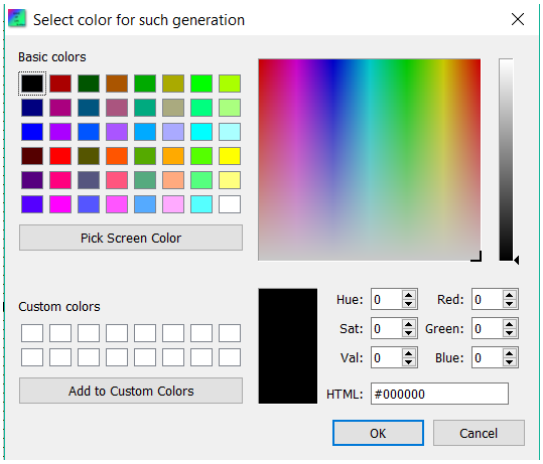

Рисунок 4. Окно выбора цвета для поколения.

5. Чтобы переименовать или поменять цвет поколения, нажмите соответствующие кнопки на экране: «Rename» «Color». И помните, что все последующие поколения должны иметь отличные от имеющихся цвета и имена, иначе вы получите вот такое сообщение как на рис. 5.

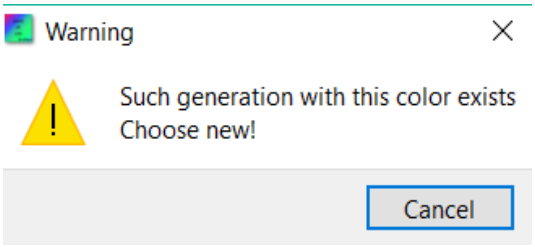

Рисунок 5. Окно предупреждения о неправильном выборе цвета.

6. Чтобы удалить текущее поколение – нажмите кнопку «Delete». Знайте: последнее поколение должно присутствовать, иначе вы получите окно как на рис. 6.

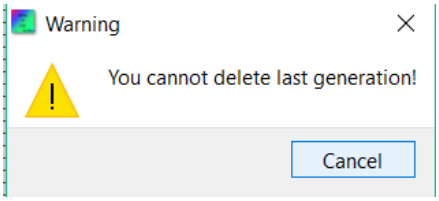

Рисунок 6. Окно предупреждения о том, что последнее поколение должно жить.

7. Чтобы изменить цвет заднего фона – нажмите кнопку «Select background color». Программа предложит вам выбрать любой цвет(рис. 7), также постарается подобрать оптимальный реверс-цвет для того, чтобы компоненты графического интерфейса программы были хорошо видны на фоне.

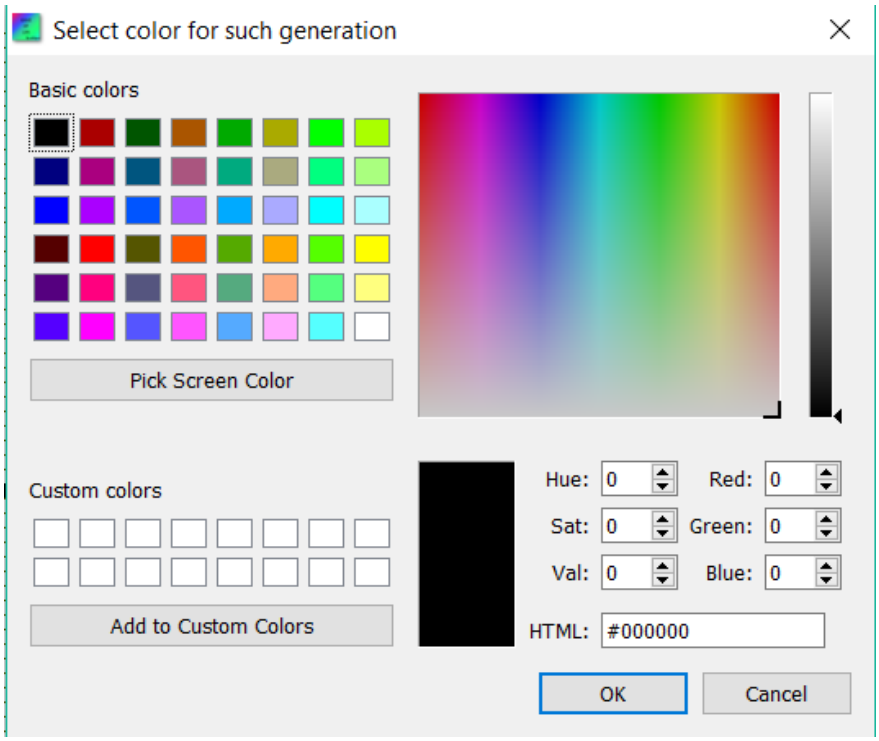

Рисунок 7. Окно выбора цвета заднего фона.

8. Чтобы изменить размер Вселенной, введите число от 10 до 400 в поле «Universe size» или воспользуйтесь соответствующими стрелками вверх/вниз. После изменений во Вселенной – поле становится недоступным.

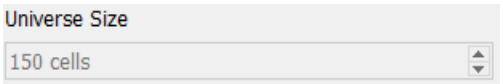

Рисунок 8. Элемент выбоа размера Вселенной.

9. Чтобы изменить интервал между генерациями поколений – введите число от 50 до 10000 в поле «Generation Interval».

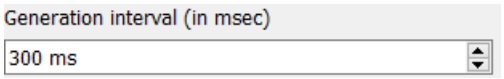

Рисунок 9. Элемент выбора интервала времени между поколениями.

10. Чтобы начать игру - расставьте начальное состояние и нажмите кнопку «Start»(рис. 10).

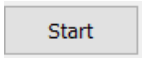

Рисунок 10. Кнопка «Start» на графическом интерфейсе программы.

11. Чтобы создать следующее поколение нажмите кнопку.(рис. 11)

Рисунок 11. Кнопка «Next Generation» на графическом интерфейсе программы.

12. Чтобы остановить игру – нажмите кнопку «Stop»(рис. 12).

Рисунок 12. Кнопка «Stop» на графическом интерфейсе программы.

13. Чтобы очистить поле игры – нажмите кнопку «Clear»(рис. 13).

Рисунок 13. Кнопка «Clear» на графическом интерфейсе программы.

14. Чтобы сохранить игру – нажмите кнопку «Save»(рис. 14) и пропишите путь и имя куда вам нужно сохранить игру. Игра будет сохранена под введенным именем с расширением (.life).

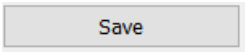

Рисунок 14. Кнопка «Save» на графическом интерфейсе программы.

15. Чтобы загрузить сохраненную игру – нажмите кнопку «Load»(рис. 15), выберите путь, где находится ваше сохранение и нажмите «Открыть».

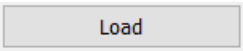

Рисунок 15. Кнопка «Load» на графическом интерфейсе программы.

6. ВЫВОД.

Итак, курсовой проект по объектно-ориентированному программированию считаю выполненным. Основная цель – написание программы–игры «Жизнь» на основе идеи Джона Конвея была достигнута. Написаны множественные доработки и модификации игры. Например, опция смешивания цветов клеток, система поколений, сохранения игры и т.д. Проект является уникальной разработкой с множеством идей и фишек. Считаю, что все поставленные задачи были в полной мере были выполнены. Однако проект всё ещё можно дорабатывать: сделать матрицу Вселенной не двухмерной, а трехмерной; модифицировать правила игры, сделать несколько режимов работы просчета поколений, доработать систему поколений, сделать её наиболее приближенной к какой-либо сфере деятельности, например, к генетике и т.д.

В ходе работы над курсовым проектом было получено и закреплено некоторое количество специальных знаний, проект был полезен и интересен.

7. СПИСОК ЛИТЕРАТУРЫ

[1] Andrew Adamatzky. Game of Life Cellular Automata. — Springer-Verlag London, 2010.
[2] Paul Rendell. Turing Machine Universality of the Game of Life. — Springer International Publishing, 2016. — (Emergence, Complexity and Computation ; vol. 18).
[3] Уэзерелл Ч. Этюды для программистов. — М.: Мир, 1982. — С. 19-22.
[4] Гарднер М. Крестики-нолики. — М.: Мир, 1988. — С. 287—343. —
[5] Щеглов Г. Шахматная Эволюция. — Lambert Academic Publishing, 2012. — 88 с.
[6] Steve McConnell Coding Complete. — Microsoft Press, 2nd edition, 2004. — С. 906.
[7] Steve McConnell Proffesional Software Development. — Addison-Wesley Professional , 1st edition, 2005. — С. 272.
[8] Chad Fowler Clean Code. — Prentice Hall , 1st edition, 2008. — С. 464.
[9] Donald Knuth Computer Science. — Microsoft Press, 1st edition, 2002. — С. 708.
[10] Donald Knuth The Art of Programming. — Addison-Wesley Professional, 1st edition, 2011. — С. 3168.
[11] Bjarne Stroustrup Principle and Practice Using C++. — Addison-Wesley Professional, 1st edition, 2008. — С. 1272.
[12] Bjarne Stroustrup Tour to C++. — Addison-Wesley Professional, 4th edition, 2012. — С. 805.
[13] Max Shclee Qt4 Proffesional Programming with C++. [Электронный ресурс] — 2017. — Режим доступа: https://qt-book.com/ru/homeru/
[14] Qt Manuals and Tutorials [Электронный ресурс] — 2018. — Режим доступа: http://doc.qt.io/qtcreator/

ПРИЛОЖЕНИЕ А
(обязательное)
Листинг программы с комментариями

```
#include <QApplication>
#include "mainwindow.h"

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    return a.exec();
}
```

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QColor>
#include <QListWidgetItem>
#include "gamewidget.h"

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

public slots:
    void selectMasterColor();
    void saveGame();
    void loadGame();

private slots:
    void setBackgroundColor();
    void addNewGeneration();
    void renameGeneration();
    void deleteGeneration();
    void setGenerationHints();

private:
    void messageBoxWarningExec(const QString&, const QString&);
    Ui::MainWindow *ui;
    GameWidget* game;
};

#endif // MAINWINDOW_H
```

```
#include <QTextStream>
#include <QFileDialog>
#include <QDebug>
#include <QColor>
#include <QColorDialog>
#include <QInputDialog>
#include <QListWidget>
#include <QMessageBox>
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow),
    game(new GameWidget(this))
{
    ui->setupUi(this);

    QPixmap pixmap(16, 16);                         //create pixmap
    QIcon icon;                                     //create icon object

    pixmap.load(":/images/next-button", ".jpg");    // load it and use for
next button
    icon.addPixmap(pixmap);
    ui->nextButton->setIcon(icon);                  // and set

    pixmap.fill(Qt::black);                         // reuse pixmap for color
button icon
    icon.addPixmap(pixmap);
    ui->colorButton->setIcon(icon);

    ui->listWidget->addItem("default generation");              // add default
generation
    ui->listWidget->setCurrentItem(ui->listWidget->item(0));    // selection
flags at designer
    ui->listWidget->currentItem()->setIcon(icon);

    pixmap.fill(Qt::gray);                          // reuse for background
color
    icon.addPixmap(pixmap);
    ui->backgroundColor->setIcon(icon);

    // connect signals
    connect(ui->startButton, SIGNAL(clicked()), game, SLOT(startGame()));
    connect(ui->startButton, SIGNAL(clicked(bool)), ui->deleteItemButton,
SLOT(setDisabled(bool)));
    connect(ui->nextButton, SIGNAL(clicked(bool)), ui->deleteItemButton,
SLOT(setDisabled(bool)));
    connect(ui->nextButton, SIGNAL(clicked()), game, SLOT(nextGeneration()));
    connect(ui->stopButton, SIGNAL(clicked()), game, SLOT(stopGame()));
    connect(ui->clearButton, SIGNAL(clicked()), game, SLOT(clear()));
    connect(ui->iterInterval, SIGNAL(valueChanged(int)), game,
SLOT(setInterval(int)));
    connect(ui->cellsControl, SIGNAL(valueChanged(int)), game,
SLOT(setCellNumber(int)));
```

```
    connect(game, SIGNAL(environmentChanged(bool)),ui->cellsControl,
SLOT(setDisabled(bool)));
    connect(game, SIGNAL(gameEnds(bool)),ui->cellsControl,
SLOT(setEnabled(bool)));
    connect(game, SIGNAL(gameEnds(bool)),ui->deleteItemButton,
SLOT(setEnabled(bool)));
    connect(ui->colorButton, SIGNAL(clicked()), this,
SLOT(selectMasterColor()));
    connect(ui->backgroundColor, SIGNAL(clicked()), this,
SLOT(setBackgroundColor()));
    connect(ui->saveButton, SIGNAL(clicked()), this, SLOT(saveGame()));
    connect(ui->loadButton, SIGNAL(clicked()), this, SLOT(loadGame()));
    connect(ui->addItemButton, SIGNAL(clicked()), this,
SLOT(addNewGeneration()));
    connect(ui->renameItemButton, SIGNAL(clicked()), this,
SLOT(renameGeneration()));
    connect(ui->deleteItemButton, SIGNAL(clicked()), this,
SLOT(deleteGeneration()));
    connect(ui->listWidget, SIGNAL(itemSelectionChanged()), this,
SLOT(setGenerationHints()));

    ui->gameLayout->addWidget(game);                            // and add new
widget
}

MainWindow::~MainWindow()
{
    delete ui;
    delete game;
}

void MainWindow::saveGame()
{
    QString filename = QFileDialog::getSaveFileName(this,
                                                    tr("Save current game"),
                                                    QDir::homePath(),
                                                    tr("Conway's Game *.life
Files (*.life)"));
    if(filename.length() < 1)
        return;
    QFile file(filename);
    if(!file.open(QIODevice::WriteOnly | QIODevice::Truncate))
        return;

    file.write((QString::number(game->cellNumber())+"\n").toUtf8());
//write num cells
    file.write(game->dump().toUtf8());
//write dump of map
    file.write((QString::number(ui->listWidget->count())+"\n").toUtf8());
//write number of generations
    for(int i=0; i<ui->listWidget->count(); i++)
//for each gen
    {
        file.write((QString::number(game->generationColor[i].red())+" "+
//write colors of generations
                    QString::number(game->generationColor[i].green())+" "+
```

```
                    QString::number(game->generationColor[i].blue())+"
").toUtf8());
        file.write((ui->listWidget->item(i)->text()+"\n").toUtf8());
//write names of generations
    }

    QString buf ="";
    QColor color = game->getBackgroundColor();
//write background color
    buf = QString::number(color.red())+" "+
          QString::number(color.green())+" "+
          QString::number(color.blue())+"\n";
    file.write(buf.toUtf8());
    buf.clear();

    buf = QString::number(ui->iterInterval->value())+"\n";
//write interval
    file.write(buf.toUtf8());
    file.close();
}

void MainWindow::loadGame()
{
    QString filename = QFileDialog::getOpenFileName(this,
                                                     tr("Open saved game"),
                                                     QDir::homePath(),
                                                     tr("Conway's Game Of Life
File (*.life)"));
    if(filename.length() < 1)
        return;
    QFile file(filename);
    if(!file.open(QIODevice::ReadOnly))
        return;
    QTextStream in(&file);

    //read and set number of cells
    int sv;
    in >> sv;
    ui->cellsControl->setValue(sv);

    //read and set map this size
    const int constSize = sv*sv;
    int *dump = new int[constSize];
    for(int k=0; k != constSize; k++) {
        in >> dump[k];
    }
    game->setDump(dump);
    delete[] dump;

    int numOfGenerations;                           //read and set number of
colors
    in >> numOfGenerations;

    ui->listWidget->clear();                        //clear all previous data
from listWidget
    game->generationColor.clear();                  //clear vector of colors
```

```
    //read and set names of generations and rgb colors in loop
    int r,g,b;                                      // RGB color
    for(int i=0; i<numOfGenerations; i++)
    {
        in >> r >> g >> b;                          // read rgb of generation
        QColor color(r,g,b);                        // create such color
        game->generationColor.push_back(color);     // push color into vector
of colors

        QString genName = in.readLine();            // read name of generation
        genName.remove(0, 1);                       // remove first ' '
        ui->listWidget->addItem(genName);           // add new generation of
readed name

        QPixmap icon(16,16);                        // create pixmap obj
        icon.fill(color);                           // fill it with such color
        ui->listWidget->item(i)->setIcon(QIcon(icon));// set it's color icon
    }

    game->setMasterColor(QColor(r,g,b));            // set last color as
master
    ui->listWidget->setCurrentItem(ui->listWidget->item(ui->listWidget-
>count()-1));  //set last readed item as current
    QPixmap icon(16, 16);                           // icon on the color
button
    icon.fill(game->getMasterColor());              // fill with new color
    ui->colorButton->setIcon( QIcon(icon) );        // set icon for button

    //read and set rgb background
    in >> r >> g >> b;                              // read background color
    //set backgroung color with hints
    QColor color(r,g,b);
    game->setBackgroundColor(color);                //set new background color
at game class
    QColor invertedBackgroundColor = game->invertBackgroundColor();
    ui->universeSize->setStyleSheet(QString("color:
%1").arg(invertedBackgroundColor.name()));      //set stylesheets for labels
and groupboxes
    ui->interval->setStyleSheet(QString("color:
%1").arg(invertedBackgroundColor.name()));
    ui->generationsGroupBox->setStyleSheet(QString("color:
%1").arg(invertedBackgroundColor.name()));

    QPalette p(ui->centralWidget->palette());       //create object of palette
class
    p.setColor(QPalette::Background, color);        //set new color on palette
    ui->centralWidget->setAutoFillBackground(true); //set autofill background
    ui->centralWidget->setPalette(p);               //set new palette on
widget

    icon.fill(color);
    ui->backgroundColor->setIcon(QIcon(icon));      //set new icon for
backgroundColor button

    //read and set interval
    in >> r;                                    // r will be interval number
    ui->iterInterval->setValue(r);
```

```
    game->setInterval(r);
    file.close();
    emit game->environmentChanged(true);     //lock universe size
}

void MainWindow::selectMasterColor()
{
    QColor color = QColorDialog::getColor(game->getMasterColor(), this,
tr("Select color of cells generations"));
    if(!color.isValid())
        return;

    for(int i = 0; i < game->generationColor.size(); i++)
    {
        if(color == game->generationColor[i])
        {
            messageBoxWarningExec("Warning", "Such generation with this color
exists\nChoose new!");
            return;
        }
    }

    game->changeCellsOfGenColor(game->convertQColorToInt(game-
>generationColor.at(ui->listWidget->currentRow())), game-
>convertQColorToInt(color));   //replace prev color with new choosen color
    game->generationColor.remove(ui->listWidget->currentRow());         //
remove prev color
    game->generationColor.insert(ui->listWidget->currentRow(), color);  // set
new color to vector
    game->setMasterColor(color);                                        // set
new master
    QPixmap pixmap(16, 16);                                    // change icon
    pixmap.fill(color);
    QIcon icon(pixmap);                                // create icon
    ui->colorButton->setIcon(icon);                    // set icon on color
button
    ui->listWidget->currentItem()->setIcon(icon);    // and also set icon in
listWidget
}

void MainWindow::setBackgroundColor()
{
    QColor color = QColorDialog::getColor(game->getMasterColor(), this,
tr("Select Background Color"));
    if(!color.isValid())
       return;

    game->setBackgroundColor(color);                //set new background color
at game class
    QColor invertedBackgroundColor = game->invertBackgroundColor();
    ui->universeSize->setStyleSheet(QString("color:
%1").arg(invertedBackgroundColor.name()));
    ui->interval->setStyleSheet(QString("color:
%1").arg(invertedBackgroundColor.name()));
    ui->generationsGroupBox->setStyleSheet(QString("color:
%1").arg(invertedBackgroundColor.name()));
```

```
    QPalette p(ui->centralWidget->palette());       //create object of palette
class
    p.setColor(QPalette::Background, color);        //set new color on palette
    ui->centralWidget->setAutoFillBackground(true); //set autofill background
    ui->centralWidget->setPalette(p);               //set new palette on
widget

    QPixmap icon(16, 16);                           //change icon
    icon.fill(color);
    ui->backgroundColor->setIcon(QIcon(icon));
}

void MainWindow::addNewGeneration()
{
    bool ok;
    QString text = QInputDialog::getText(this, tr("Add generation!\n"),
                                         tr("New generation name:"),
QLineEdit::Normal,
                                         QDir::home().dirName(), &ok);
    if (!ok) return;
//if it's not ok
    if(text.isEmpty())
//it cannot be empty string
    {
        messageBoxWarningExec("Warning", "Please, name generation!");
        return;
    }

    for(int i = 0; i < ui->listWidget->count();i++)
//search in listview for such generations
    {
        if(ui->listWidget->item(i)->text() == text)
        {
            messageBoxWarningExec("Warning", "Such generation exists!");
            return;
        }
    }

    QColor color;
//now choose color for generation
    while(1)
    {
        bool isColorValid = true;
        color = QColorDialog::getColor(game->getMasterColor(), this, "Select
color for such generation");
        if(!color.isValid())
        {
            messageBoxWarningExec("Warning", "Color is not valid! Try
again!");
            continue;
        }

        for(int i = 0; i < game->generationColor.size(); i++)
//search in listview for such generations
        {
            if(color == game->generationColor.at(i))
            {
```

```
                messageBoxWarningExec("Warning", "Such color exists!\nYou
can't have TWO same colors at one time!\nChoose new color!");
                isColorValid = false;
                break;
            }
        }

        if(isColorValid) break;
    }

    ui->listWidget->addItem(text);
//add new item
    ui->listWidget->setCurrentItem(ui->listWidget->item(ui->listWidget-
>count()-1));  //set new current

    QPixmap pixmap(16, 16);
//create pixmap for item icon
    pixmap.fill(color);
    QIcon icon(pixmap);
//fill pixmap with color
    ui->listWidget->currentItem()->setIcon(icon);
//set new icon of black color
    ui->colorButton->setIcon(icon);
    game->setMasterColor(color);
//set new master color
    game->generationColor.push_back(color);
//color by default push into vector
}

void MainWindow::renameGeneration()
{
    bool ok;
    QString text = QInputDialog::getText(this, tr("Rename generation!\n"),
                                         tr("New generation name:"),
QLineEdit::Normal,
                                         QDir::home().dirName(), &ok);
    if (!ok) return;
    if(text.isEmpty())
    {
        messageBoxWarningExec("Warning", "Name connot be empty string!");
        return;
    }

    for(int i = 0; i<ui->listWidget->count();i++)
    {
        if(ui->listWidget->currentItem()->text() == text)
        {
            messageBoxWarningExec("Warning", "Such generation exists!");
            return;
        }
    }

    ui->listWidget->currentItem()->setText(text);     //change generation name
}

void MainWindow::deleteGeneration()
{
```

```
    if(ui->listWidget->count() == 1)
    {
        messageBoxWarningExec("Warning", "You cannot delete last
generation!");
        return;
    } else
    {
        game->deleteMembersOfGeneration(game->convertQColorToInt(game-
>generationColor[ui->listWidget->currentRow()])); //remove all members of
generation
        game->generationColor.remove(ui->listWidget->currentRow());
//remove color from vector
        if(ui->listWidget->currentRow() == 0)
        {
            ui->listWidget->setCurrentRow(ui->listWidget->count()-1);
            delete ui->listWidget->takeItem(0);
            return;
        }
        delete ui->listWidget->takeItem(ui->listWidget->currentRow());
//delete from list of gui
    }
}

void MainWindow::setGenerationHints()
{
    game->setMasterColor(game->generationColor[ui->listWidget-
>currentRow()]) ; //set new current color
    QIcon icon(ui->listWidget->currentItem()->icon());
//get icon from listWidget item
    ui->colorButton->setIcon(icon);
//set new cell button color
}

void MainWindow::messageBoxWarningExec(const QString& title, const QString&
message)
{
    QMessageBox info(QMessageBox::Icon::Warning,
                     title,
                     message,
                     QMessageBox::StandardButton::Cancel,
                     this);
    info.exec();
}

#ifndef GAMEWIDGET_H
#define GAMEWIDGET_H

#include <QColor>
#include <QWidget>

class GameWidget : public QWidget
{
    Q_OBJECT
public:
    explicit GameWidget(QWidget *parent = nullptr);
    void deleteMembersOfGeneration(int);
    int convertQColorToInt(const QColor&) const;  //convertation functions
```

```
    QColor convertIntToQColor(const int&) const;
    QVector<QColor> generationColor;        //all colors
    ~GameWidget();

protected:
    void paintEvent(QPaintEvent *);
    void resizeEvent(QResizeEvent *);
    void mousePressEvent(QMouseEvent *);
    void mouseMoveEvent(QMouseEvent *);
    void countSellAndUpdateIt(const int&, const int&, const bool);

signals:
    //when one of the cell has been changed,emit this signal to lock the
universeSize
    void environmentChanged(bool ok);
    //when game is over or clear is called,emit it to unlock the universeSize
    void gameEnds(bool ok);

public slots:
    void startGame();  // start
    void nextGeneration();
    void stopGame(); // finish
    void clear();   // clear

    void setCellNumber(const int &s); // set number of the cells in one row
    void setInterval(int msec); // set interval between generations
    void setBackgroundColor(const QColor&);
public:
    void changeCellsOfGenColor(const int, const int);

    QColor invertBackgroundColor() const;

    int cellNumber() const; // number of the cells in one row

    int interval() const; // interval between generations

    QColor getMasterColor() const; // color of the cells
    void setMasterColor(const QColor&); // set color of the cells

    QColor getBackgroundColor() const;

    QString dump() const; // dump of current universe
    void setDump(int *&data); // set current universe from it's dump

private slots:
    void paintGrid(QPainter &p);
    void paintUniverse(QPainter &p);
    void newGeneration();

private:
    int widthMargins, heightMargins;                          //margins of
gamewidget
    int gridSize;
    int prevSelectedCellX, prevSelectedCellY;
    QColor m_masterColor;
    QColor m_backgroundColor;
    QTimer* timer;
```

```
    int* universe; // current map(contains color or -1)
    int* next;     // next map
    int universeSize;
    void givePowerAndExtractColors(int&, int&, int&, int&, const int&, const
int&); //gives power to cell and count its color
    int isAlive(int k, int j); // return true if universe[k][j] accept rules
    void resetUniverse();      // reset the size of universe
    int extractRed(const int&);// extract functions
    int extractGreen(const int&);
    int extractBlue(const int&);
    void countMargins();
};

#endif // GAMEWIDGET_H

#include <QMessageBox>
#include <QTimer>
#include <QMouseEvent>
#include <QDebug>
#include <QRectF>
#include <QPainter>
#include <qmath.h>
#include "gamewidget.h"

GameWidget::GameWidget(QWidget *parent) :
    QWidget(parent),
    timer(new QTimer(this)),
    universe(nullptr),
    next(nullptr),
    universeSize(50)
{
    m_masterColor = Qt::black;                          //set default color
    m_backgroundColor = Qt::gray;                       //set default
background
    timer->setInterval(300);                            //set default timer
interval
    generationColor.push_back(m_masterColor);           //set first color
black by default
    prevSelectedCellX = -1;
    prevSelectedCellY = -1;
    connect(timer, SIGNAL(timeout()), this, SLOT(newGeneration()));
    resetUniverse();                                    //set current and
next universes and fill them
}

GameWidget::~GameWidget()
{
    delete [] universe;
    delete [] next;
}

void GameWidget::startGame()
{
    timer->start();
}

void GameWidget::nextGeneration()
```

```
{
    newGeneration();
}

void GameWidget::stopGame()
{
    timer->stop();
}

void GameWidget::clear()
{
    for(int k = 1; k <= universeSize; k++) {
        for(int j = 1; j <= universeSize; j++) {
            universe[k*universeSize + j] = -1;
        }
    }
    gameEnds(true);
    update();
}

QColor GameWidget::invertBackgroundColor() const
{
    QColor color;
    int h = (m_backgroundColor.hue() + 180) % 360;
    int s = 255 - m_backgroundColor.saturation();
    int v = 255 - m_backgroundColor.value();

    color.setHsv(h, s, v);
    return color;
}

int GameWidget::cellNumber() const
{
    return universeSize;
}

void GameWidget::setCellNumber(const int &s)
{
    universeSize = s;
    resetUniverse();
    update();
}

void GameWidget::resetUniverse()
{
    delete [] universe;
    delete [] next;
    universe = new int[(universeSize + 2) * (universeSize + 2)];
    next = new int[(universeSize + 2) * (universeSize + 2)];
    memset(universe, -1, sizeof(int) * static_cast<unsigned int>(universeSize
+ 2) * static_cast<unsigned int>(universeSize + 2));
    memset(next, -1, sizeof(int) * static_cast<unsigned int>(universeSize + 2)
* static_cast<unsigned int>(universeSize + 2));
    countMargins();
//always after resizing change margins
}
```

```
void GameWidget::countMargins()
{
    int w = width();
    int h = height();

    if(w < h)
    {
        gridSize = w / universeSize * universeSize;
    } else
    {
        gridSize = h / universeSize * universeSize;
    }
    widthMargins = (w - gridSize) / 2;
    heightMargins = (h - gridSize) / 2;
}

QString GameWidget::dump() const
{
    QString master = "";
    for(int k = 1; k <= universeSize; k++) {
        for(int j = 1; j <= universeSize; j++) {
            master += QString::number(universe[k*universeSize+j], 10) + ' ';
        }
        master.append("\n");
    }
    return master;
}

void GameWidget::setDump(int *&data)
{
    for(int k = 1; k <= universeSize; k++)
        for(int j = 1; j <= universeSize; j++)
            universe[k*universeSize + j] = data[(k-1)*universeSize + j-1];
    update();
}

int GameWidget::interval() const
{
    return timer->interval();
}

void GameWidget::setInterval(int msec)
{
    timer->setInterval(msec);
}

void GameWidget::givePowerAndExtractColors(int& power, int& red, int& green,
int& blue, const int& k, const int& j)
{
    power++;
    red += extractRed(universe[k*universeSize + j]);
    green += extractGreen(universe[k*universeSize + j]);
    blue += extractBlue(universe[k*universeSize + j]);
}

int GameWidget::isAlive(int k, int j)
{
```

```
    int power = 0, r = 0, g = 0, b = 0;             //power red green blue
                                                    //give some power to cell
and extract colors
    if(universe[(k+1)*universeSize + j] != -1)
givePowerAndExtractColors(power, r, g, b, k + 1, j);
    if(universe[(k-1)*universeSize + j] != -1)
givePowerAndExtractColors(power, r, g, b, k - 1, j);
    if(universe[k*universeSize + (j+1)] != -1)
givePowerAndExtractColors(power, r, g, b, k, j + 1);
    if(universe[k*universeSize + (j-1)] != -1)
givePowerAndExtractColors(power, r, g, b, k, j - 1);
    if(universe[(k+1)*universeSize + (j-1)] != -1)
givePowerAndExtractColors(power, r, g, b, k + 1, j - 1);
    if(universe[(k-1)*universeSize + (j+1)] != -1)
givePowerAndExtractColors(power, r, g, b, k - 1, j + 1);
    if(universe[(k-1)*universeSize + (j-1)] != -1)
givePowerAndExtractColors(power, r, g, b, k - 1, j - 1);
    if(universe[(k+1)*universeSize +  (j+1)] != -1)
givePowerAndExtractColors(power, r, g, b, k + 1, j + 1);
    if (((universe[k*universeSize + j] != -1) && (power == 2)) || (power ==
3))
    {
        r = r / power;          //count average color from relatives
        g = g / power;
        b = b / power;
        QColor color(r, g, b);
        return convertQColorToInt(color);
    }
    return -1;
}

void GameWidget::newGeneration()
{
    int notChanged=0;
    for(int k=1; k <= universeSize; k++) {
        for(int j=1; j <= universeSize; j++) {
            next[k*universeSize + j] = isAlive(k, j);
            if(next[k*universeSize + j] == universe[k*universeSize + j])
                notChanged++;
        }
    }
    if(notChanged == universeSize*universeSize) {
        QMessageBox::information(this,
                                 tr("Game lost sense"),
                                 tr("The End. Now game finished because all
the next generations will be the same."),
                                 QMessageBox::Ok);
        stopGame();
        gameEnds(true);
        return;
    }
    for(int k=1; k <= universeSize; k++) {
        for(int j=1; j <= universeSize; j++) {
            universe[k*universeSize + j] = next[k*universeSize + j];
        }
    }
    update();
```

```
}

void GameWidget::paintEvent(QPaintEvent *)
{
    QPainter p(this);
    paintGrid(p);
    paintUniverse(p);
}

void GameWidget::resizeEvent(QResizeEvent *)
{
    countMargins();
}

void GameWidget::mousePressEvent(QMouseEvent *e)
{
    int x = e->x();
    int y = e->y();
    int w = width();
    int h = height();
    if (x < w - gridSize - widthMargins ||
            y < h - gridSize - heightMargins ||
            x >= w - widthMargins - 1 ||
            y >= h - heightMargins - 1) return;           //to avoid the free
clickeng processing
    emit environmentChanged(true);                                          //to
block universe size
    countSellAndUpdateIt(x, y, false);
}

void GameWidget::mouseMoveEvent(QMouseEvent *e)
{
    int x = e->x();
    int y = e->y();
    int w = width();
    int h = height();

    if (x < w - gridSize - widthMargins ||
            y < h - gridSize - heightMargins ||
            x >= w - widthMargins - 1 ||
            y >= h - heightMargins - 1) return;           //to avoid the free
clickeng processing
    countSellAndUpdateIt(x, y, true);
}

void GameWidget::countSellAndUpdateIt(const int& x,const int& y, const bool
isMove)
{
    int cellWidthAndHeight = gridSize / universeSize;
    int k = (y - heightMargins) / cellWidthAndHeight + 1;
    int j = (x - widthMargins) / cellWidthAndHeight + 1;

    int currentColorInt = convertQColorToInt(m_masterColor);
    if(!isMove)                     //if button was pressed
    {
        if(universe[k*universeSize + j] != currentColorInt)
            universe [k*universeSize + j] = currentColorInt;
```

```
        else
            universe [k*universeSize + j] = -1;
    }
    else                            //if button was pressed and moved
    {
        if(prevSelectedCellX != j || prevSelectedCellY != k)
        {
            if(universe[k*universeSize + j] != currentColorInt)
                universe [k*universeSize + j] = currentColorInt;
            else
                universe [k*universeSize + j] = -1;
            prevSelectedCellX = j;
            prevSelectedCellY = k;
        }
    }
    update();
}

void GameWidget::paintGrid(QPainter &p)
{
    QRect borders(widthMargins, heightMargins , gridSize, gridSize);  // set
borders of the universe
    QColor gridColor = invertBackgroundColor();
// color of the grid
    p.setPen(gridColor);

    int cellWidthAndHeight = gridSize / universeSize;
    int x1 = width() - widthMargins - gridSize;
    int y1 = height() - heightMargins - gridSize;
    int x2 = x1 + gridSize;
    int y2 = y1 + gridSize;

    for(int i = 1; i < universeSize; i++)
    {
        p.drawLine(x1 + i * cellWidthAndHeight, y1, x1 + i *
cellWidthAndHeight, y2);       //draw grid lines
        p.drawLine(x1, y1 + i * cellWidthAndHeight, x2, y1 + i *
cellWidthAndHeight);
    }

    p.drawRect(borders);                                                    //draw
borders of universe
}

void GameWidget::paintUniverse(QPainter &p)
{
    int cellWidthAndHeight = gridSize / universeSize;
    int x1 = width() - widthMargins - gridSize;
    int y1 = height() - heightMargins - gridSize;

    for(int k=1; k <= universeSize; k++) {
        for(int j=1; j <= universeSize; j++) {
            if(universe[k*universeSize + j] != -1) {                    // if
there is any sense to paint it
                int left = x1 + cellWidthAndHeight*(j-
1);                    // margin from left
```

```
                int top  = y1 + cellWidthAndHeight*(k-
1);                     // margin from top
                QRect r(left, top, cellWidthAndHeight, cellWidthAndHeight);
                p.fillRect(r,
QBrush(convertIntToQColor(universe[k*universeSize + j]))); // fill cell with
brush of main color
            }
        }
    }
}

QColor GameWidget::getMasterColor() const
{
    return m_masterColor;
}

void GameWidget::setMasterColor(const QColor &color)
{
    m_masterColor = color;
    update();
}

QColor GameWidget::getBackgroundColor() const
{
    return m_backgroundColor;
}

void GameWidget::setBackgroundColor(const QColor& backgroundColor)
{
    m_backgroundColor = backgroundColor;
}

void GameWidget::changeCellsOfGenColor(const int prevColor, const int
newColor)
{
    for(int k = 1; k <= universeSize; k++)
        for(int j = 1; j <= universeSize; j++)
            if(universe[k*universeSize + j] == prevColor)
                universe[k*universeSize + j] = newColor;
}

void GameWidget::deleteMembersOfGeneration(int color)
{
    for(int k=1; k <= universeSize; k++)
        for(int j=1; j <= universeSize; j++)
            if(universe[k*universeSize + j] == color)
                universe[k*universeSize + j] = -1;
}

int GameWidget::convertQColorToInt(const QColor &color) const
{
    return ((color.red() & 0xff) << 16) + ((color.green() & 0xff) << 8) +
(color.blue() & 0xff);
}

QColor GameWidget::convertIntToQColor(const int &rgb) const
{
```

```
    int red = 0, green = 0, blue = 0;
    red = (rgb >> 16) & 0xff;
    green = (rgb >> 8) & 0xff;
    blue = rgb & 0xff;
    QColor color(red, green, blue);
    return color;
}

int GameWidget::extractRed(const int& color)
{
   return (color >> 16) & 0xff;
}

int GameWidget::extractGreen(const int& color)
{
   return (color >> 8) & 0xff;
}

int GameWidget::extractBlue(const int& color)
{
   return color & 0xff;
}
```

ПРИЛОЖЕНИЕ Б
(обязательное)
Диаграмма классов

ПРИЛОЖЕНИЕ В

(обязательное)

Диаграмма последовательности

ПРИЛОЖЕНИЕ Г
(обязательное)
Структурная схема