МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра РЭС

КУРСОВАЯ РАБОТА

по дисциплине «Информационные технологии»
Тема: «Игра Маджонг (пасьянс)»

Студент гр. 1181
Шишков Д.А.

Преподаватель Ситников И.Ю.

Санкт-Петербург
2022

1.Содержание

2. Спецификация задания....3
Требования к расчету и программе....3
Требования к отчету....3
Требования к пользовательскому интерфейсу программы....3
3. Постановка задачи, описание предметной области....5
4. Формализованное словесное описание алгоритма решения задачи....6
Загрузка карты (XmlLayout::readLayout):....6
Генерация поля (Controller::fillRandom, Controller::fillSolvableTable)....6
Полностью случайный....6
Претендующий на решаемость....6
Алгоритм получения случайной позиции в нижней строке....6
Алгоритм вставки элемента в поле....7
Алгоритм получения следующей после ptr позиции....7
Изменение размеров доски (Drawer::resizeBoard)....7
Перерисовка стола (Drawer::composeBoard)....9
Рисование камня (Drawer::drawTile)....9
Установка минимального размера окна (Drawer::composeMinSize)....10
5. Варианты взаимодействия оператора и программы (Use Case)....11
6. Разработка дружественного пользовательского интерфейса....12
7. Блок-схема движения данных (Data flow diagram)....14
Запуск игры....14
Изменение размера окна....14
Отрисовка окна....15
Нажатие левой кнопки мыши....15
Тик таймера....15
Перемешивание стола....16
Отмена хода....16
8. Выбор и обоснование типов переменных. Разработка структур данных....17
9. Вводимые и выводимые параметры и их типы....22
10. Диаграмма классов....23
11. Структура проекта, перечисление нужных файлов....24
12. Инструкция по использованию....26
13. Текст программы и файлов заголовков с комментариями....27
14. Рисунки с копиями экрана при работе программы....62
15. Контрольный пример....66
16. Ведомость соответствия программы спецификации....69
Требования к расчету и программе....69
Требования к отчету....69
Требования к пользовательскому интерфейсу программы....69
17. Выводы, включающие область применения, ограничения, достоинства и недостатки программы, размер исполняемого файла на диске....71
Достоинства....71
Недостатки....71
Размер исполняемого файла на диске....71
18. Ссылки....73

2. Спецификация задания

Требования к расчету и программе

- Реализовать игру Маджонг:
  - Игровое поле с отображением костей и взаимодействие с ними
  - Генерацию игрового поля по заданной схеме;
  - Таймер и процент решения;
  - Сохранение истории ходов нынешней игры (с возможностью отката);
  - Сохранение в реестре истории игр (время и процент решения);
- Поддерживать загрузку своих схем из файлов формата .smlf;
- Иметь графический пользовательский интерфейс;
- Применять разработанную схему передачи и преобразования данных в проекте Data Flow Diagram с программными интерфейсами передачи данных и диаграммой классов с указанием наследования и программных интерфейсов;
- Использовать определенный индивидуальный тип данных для хранения данных об одиночной кости. Оперировать динамическими массивами (для сохранения истории ходов).

Требования к отчету

- Отчёт должен соответствовать ГОСТу 19.701-90 единой системы программной документации;
- В отчёт необходимо включить описание программного интерфейса, диаграмму классов и диаграмму потоков данных, выбор и обоснование переменных, собственных типов и классов, код с комментариями, пример работы программы и контрольный пример
- Контрольный пример должен быть представлен в виде сгенерированного по заданной схеме поля

Требования к пользовательскому интерфейсу программы

- Содержит сведения о программе;
- Содержит сведения об авторе;
- Содержит сведения об авторских правах;
- Имеет название и иконку;
- Имеет главное окно с игровым полем;
- Имеет окно справки с правилами игры;
- Используется меню для открытия файла со схемой, дополнительных окон;

- Используются кнопки управления приложением;
- Содержатся чекбоксы (флажки);
- Соответствует понятию "дружественный интерфейс";
- Использует русский язык.

Срок сдачи отчета:
10.06.2022

Срок сдачи курсовой работы:
10.06.2022

Преподаватель Ситников И.Ю.

Студент Шишков Д.А.

3. Постановка задачи, описание предметной области

Игра Маджонг – настольная игра для одного человека, напоминающая карточный пасьянс. В ней используется набор из 144 костей таких же как в одноимённой азартной игре. Они раскладываются на поле в случайном порядке, образуя многослойную фигуру.

Картинка 1: Пример разложения в форме черепахи с подсвеченными синим доступными для снятия костями

В игре используются фишки трёх видов: масти, козыри и цветы. Существуют три масти – медяки, бамбуки и тьмы, пронумерованные от одного до девяти, по четыре комплекта каждой. Козыри подразделяются на ветры и драконов. Есть четыре ветра – Восточный, Южный, Западный и Северный по 4 шт. каждого вида. И три дракона – Красный, Зелёный и Белый по 4 шт. каждого вида. Цветы делятся на 4 цветка – слива, орхидея, хризантема и бамбук и 4 времени года – весна, лето, зима и осень, каждых по 1 шт.

Цель игры состоит в том, чтобы, убирая пары одинаковых не заблокированных фишек, очистить поле. Заблокированными считаются фишки, накрытые другими, или не имеющие свободной правой, или левой стороны.

Одинаковыми считаются медяки, козыри и цветы одной масти и достоинства, одинаковые драконы и ветра. Все четыре кости-цветка и времени года считаются одинаковыми.

4. Формализованное словесное описание алгоритма решения задачи

Загрузка карты (XmlLayout::readLayout):

Файл предстовляет собой XML документ, содержащий корневой элемент *layout* с характеристиками карты, внутри которого содержатся элемент с именем *designer*, содержащий информацию об авторе карты, после которого следуют (обычно) 144 элемента с именем *tile*, содержащие x, y и z (layer) координаты карты при том, что ширина и высота одной карты считаются равными 2.

Для его считывания файла, трёхмерный вектор table, в который будет записываться состояние конкретной позиции, инициализируется значением EMPTY. Далее запускается цикл while, пока не просмотрены все дочерние элементы корневого, и, если это tile, тогда из него считываются поля x, y и layout и в массив table в соответствующую позицию записывается значение FREE.

Генерация поля (Controller::fillRandom, Controller::fillSolvableTable)

Программа поддерживает два способа генерации поля – полностью случайный и с заявкой на решаемость.

Полностью случайный

Инициализируется генератор случайных чисел. Потом, в цикле по элементам массива, каждый раз проверяя при помощи переменной-счётчика, заполняем позиции, чьи значения равны FREE случайным id карты и уменьшаем счётчик на 1.

Претендующий на решаемость

Инициализируется генератор случайных чисел. В отдельную переменную сохраняется количество оставшихся камней. Потом, в набор координат positions записывается случайная координата с идентификатором FREE из нижнего уровня. Создаётся итератор next_ptr, указывающий на начало сета (только что помещённый туда элемент). Далее, в цикле, пока набор positions не пуст, получаем случайный id камня. Вставляем его в позицию next_ptr, добавляя сопряжённые позиции в positions. Уменьшаем счётчик оставшихся для вставки камней. Находим случайную новую позицию next_ptr так, чтобы она не накрывала предыдущую. Если это id парной карты (меньше 34), тогда уменьшаем счётчик карт для этого id, иначе, получаем новый id в нижней плоскости. Снова вставляем id в next_ptr. Уменьшаем счётчик оставшихся для вставки камней и находим случайную новую позицию next_ptr.

Алгоритм получения случайной позиции в нижней строке

В цикле do-while, пока элемент по выбранной позиции не является свободным, получаем случайное число в диапазоне от 0 до произведения размера доски по x и по y. Тогда x координата точки получается делением этого числа на количество строк карты, а y – остатком от деления на него же.

Алгоритм вставки элемента в поле

Вставляем в позицию next_ptr выбранный id. Далее добавляем в positions все позиции, смежные с next_ptr и которые не блокируют другие доступные позиции, где ещё не стоят карты.

Алгоритм получения следующей после ptr позиции

Сохраняем значение итератора ptr в prev. Удаляем элемент, обозначаемый итератором из positions. Если ещё есть камни, которые нужно вставить:

> Устнанавливаем ptr равным итератору к первому элементу набора positions.
>
> Смещаем ptr на случайную величину меньшую размера positions и сохраняем этот итератор в rand_ptr.
>
> Если нельзя, чтобы новая позиция перекрывала старую:
>
> > В цикле, пока ptr не указывает на конец positions и перекрывает prev, наращиваем ptr.
> >
> > Если ptr == positions.end(), аналогично итерируемся через ptr от начала positions до rand_ptr.
> >
> > Если ptr == rand_ptr и ptr перекрывает prev:
> >
> > > Если количество камней для вставки равно количеству позиций в positions, просто берём ptr указывающим на первый элемент positions. Иначе, пока не удаётся вставить новый элемент в positions, пытаемся вставить туда позицию из нижней плоскости. Вставленный элемент и будет ptr.

Изменение размеров доски (Drawer::resizeBoard)

Расчёт размеров и перерисовка игрового поля – весьма дорогая операция, поэтому вводятся величина gridPoint, благодаря которой появляются некоторые “брейкпоинты”, в пределах которых размеры будут оставаться неизменными, а значит, и не нужно будет перерисовывать окно.

В случае плоской карты с квадратными камнями она вычисляется при помощи простой формулы:

$$gridPoint = \frac{resolution}{gridSize * scale}$$

При этом половина стороны камня (половина, так как gridSize – размер поля, предполагающий, что размер камня составляет две условные единицы):

$$tilePixelSize = gridSize * scale$$

А размер игрового поля:

$$tablePixelRect = tilePixelSize * gridSize$$

Если же соотношение сторон у камня не 1:1, тогда берётся минимум из приведённого выше выражения для каждой из сторон:

$$gridPoint = min\left(\frac{resoloution.x}{gridSize.x * TILE\_WITH}, \frac{resolution.y}{gridSize.y * TILE\_HEIGHT}\right),$$

где TILE_WIDTH = 6, TILE_HEIGH = 8.

Теперь вернёмся немного назад и добавим к игровому полю отступы для того, чтобы в него помещались второй слой камня, добавленный для придания объёма, а так же, камни, смещённые в сторону для видимости третьего измерения.

Функция преобразования координаты курсора мыши в координаты сетки не учитывает смещение камней, находящихся выше по оси z. Поэтому могут возникать ошибки, когда пользователь, казалось бы, согласно изображению на экране кликает по одному камню, а на самом деле в плоском представлении там находится другой. Исправление этого поведения потребовало бы значительного усложнения кода, поэтому предполагается, что в таком случае пользователь просто кликнет правее и ниже, чтобы точно попасть по требуемому камню.

Однако, для того, чтобы уместить смещённые из-за поднятия камни на битмапе, необходимо вычесть из разрешения, даваемого для сетки игрового поля паддинги. Максимальное смещение вдоль оси для карты будет у самой верхней карты, поэтому, считая, что крайняя карта (с той стороны, в которую идёт смещение) находится в высшей позиции и что одно смещение будет равно gridPoint, получается, что её смещение равно $gridPoint * (gridSize.z - 1)$. С другой стороны из-за того, что подложка смещена в обратную сторону, с обратного края будет выпирать ещё ода единица смещения камня равная gridPoint. Тогда общая вычитаемая из разрешения панели величина будет равна $gridPoint * gridSize.z$

Тогда для x: $gridPoint = \frac{resolution.x - gridPoint * gridSize.z}{gridSize.x * TILE\_WIDTH}$

Для y: $gridPoint = \frac{resolution.y - gridPoint * gridSize.z}{gridSize.y * TILE\_HEIGHT}$

Откуда выражается

$$gridPoint = min\left(\frac{resolution.x}{gridSize.x * TILE\_WIDTH + gridSize.z}, \frac{resolution.y}{gridSize.y * TILE\_HEIGHT + gridSize.z}\right)$$

Ещё немного усложним формулу: позже, в реальных тестах выяснилось, что красивее выглядит, когда отступ по оси y больше, чем по оси x, а так же что оба отступа не помешало бы увеличить. Так, введя TILE_PADDING_SCALE = 1.25 и считая, что отступ по y будет $tilePadding.y = gridPoint * \frac{TILE\_HEIGH}{TILE\_WIDTH} * TILE\_PADDING\_SCALE$,

а по x $tilePadding.x = gridPoint * TILE\_PADDING\_SCALE$, получим новое выражение для gridPoint:

$$gridPoint = \frac{resolution.x - gridPoint * gridSize.z * TILE\_PADDING\_SCALE}{gridSize.x * TILE\_WIDTH}$$

$$gridPoint = \frac{resolution.y - gridPoint * gridSize.z * \frac{TILE\_HEIGH}{TILE\_WIDTH} * TILE\_PADDING\_SCALE}{gridSize.y * TILE\_HEIGHT}$$

$$gridPoint = min\left(\frac{\frac{resolution.x}{gridSize.x * TILE\_WIDTH + gridSize.z * TILE\_PADDING\_SCALE}}{\frac{resolution.y}{gridSize.y * TILE\_HEIGHT * TILE\_WIDTH + gridSize.z * TILE\_PADDING\_SC}}\right.$$

Размеры половин сторон камня:

$$tilePixelSize = \begin{bmatrix} x : gridPoint * TILE\_WIDTH \\ y : gridPoint * TILE\_HEIGHT \end{bmatrix}$$

Смещения от левого верхнего угла для всей сетки будут равны:

$$boardPadding = tilePadding * (gridSize.z - 1)$$

Размеры доски:

$$boardPixelSize = \begin{bmatrix} x : (tilePixelSize.x * gridSize.x) + boardPadding.x + tilePadding.x \\ y : (tilePixelSize.y * gridSize.y) + boardPadding.y + tilePadding.y \end{bmatrix}$$

В конце концов, для центрирования доски вычисляем положение битмапа внутри окна:

$$boardPixelPos = \frac{resolution - boardPixelSize}{2}$$

В самой функции, сначала вычисляется gridPoint. Если он не равен предыдущему, тогда вычисляются все остальные величины, после чего перерисовывается поле, сохраняется в prevGridPoint gridPoint. Иначе, вычисляется только boardPixelPos и битмап с полем не перерисовывается.

Перерисовка стола (Drawer::composeBoard)

Создаётся новый битмап расчитанного в вышеописанной функции размера boardPixelSize. Далее создаётся временный dc, рисующий в нём. В цикле по позициям table проверяется, если в данной точке камень и если есть, рисуется в точке, переведённой из координат сетки, согласно нижеописанной процедуре.

Для нарисованного поля создаётся маска, в которой чёрный цвет, который покрывает весь битмап по умолчанию, считается фоном.

Рисование камня (Drawer::drawTile)

Каждый камень состоит из трёх частей: подложки, фронтальной части и картинки. В таком порядке они и рисуются. Рассмотрим процесс рисования камня в нижней плоскости: сначала рисуется подложка, смещённая на tilePadding вправо вниз; потом в заданной позиции рисуется фронтальная часть; потом, если при загрузке картинки для данного id не возникло никаких ошибок, масштабируем её до размеров камня минус два tilePadding и рисуем, смещённую на один tilePadding вправо вниз. Это делается для того, чтобы на фронтальной части камня были небольшие поля и картинка не выходила за скруглённые углы подложки и фронтальной части, радиус скругления которых равен tilePadding.y. Для камней, находящихся в более высоких плоскостях, координаты каждого из элементов смещены на $tilePadding * (zIndex - 1)$, где zIndex – положение камня по оси z, начиная снизу.

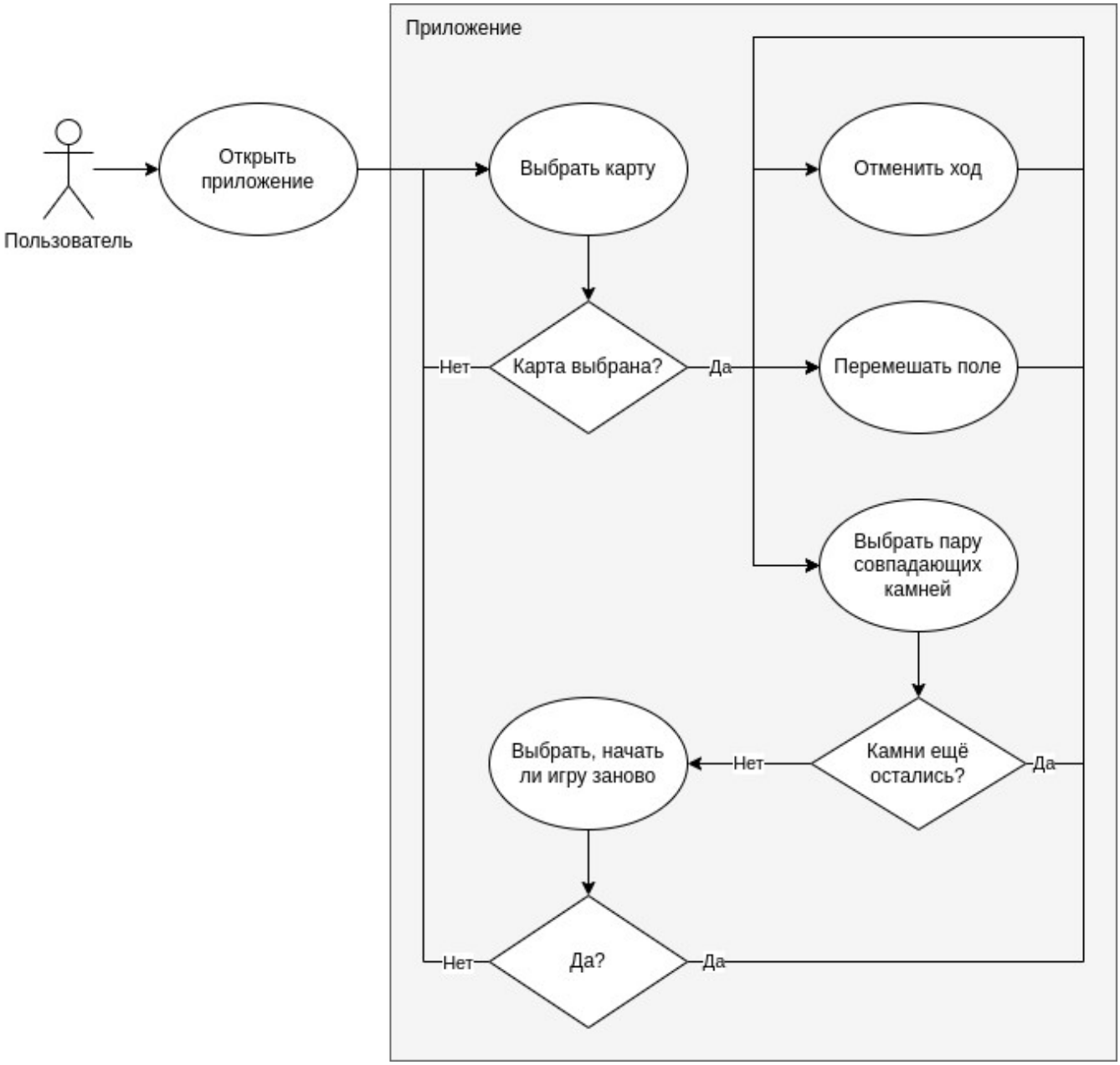
Установка минимального размера окна (Drawer::composeMinSize)

Подставляя в формулы из Drawer:: resizeBoard gridSize равный 1, можно получить выражения для resolution:

$$minSize = \begin{bmatrix} x : TILE\_WIDTH * gridSize.x + gridSize.z * TILE\_PADDING\_SCALE \\ y : TILE\_HEIGHT * gridSize.y + gridSize.z * TILE\_PADDING\_SCALE * \frac{TILE\_HEIG}{TILE\_WIDT} \end{bmatrix}$$

Однако, при округлении до целых (из-за TILE_PADDING_SCALE получаются дробные значения) целая часть полностью отбрасывается, из-за чего может получиться так, что gridSize будет чуть меньше единицы и округлится до 0. Во избежание этого к каждому измерению minSize прибавляется по единице

5. Варианты взаимодействия оператора и программы (Use Case)



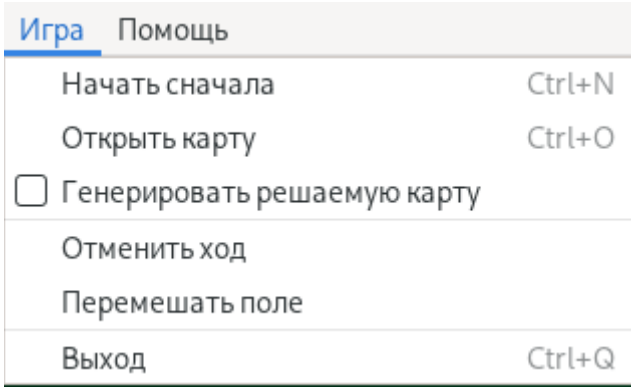
6. Разработка дружественного пользовательского интерфейса

Взаимодействие пользователя с программой происходит через игровое поле, верхнее меню, строку состояния и дополнительные диалоги.

Интерфейс поля предельно прост – на нём рисуются камни и дли выбора пар, необходимо нажимать на них левой кнопкой мыши. Размер камней автоматически адаптируется к размеру окна. При этом, проверяется, чтобы его нельзя было сделать слишком маленьким.

В верхнем меню кнопки распределены по двум пунктам:

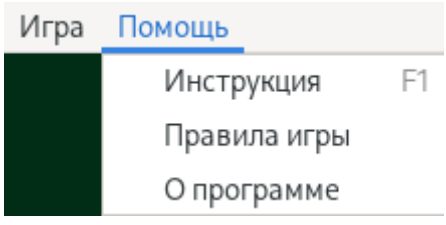
“Игра”



В нём собраны пункты, отвечающие за старт игры, дополнительные дествия, которые можно предпринять в её процессе и кнопка выхода. Каждая из перечисленных категорий отделена горизонтальной чертой.

Выбор способа заполнения карты осуществляется при помощи флажка. По умолчанию генерируется полностью случайная карта, так как этот способ быстрее и это не означает, что карту точно нельзя будет решить. Однако, если пользователь хочет получить карту, которую можно решить, это его выбор. При этом он всё равно может совершить ошибку и не решить карту. Более того, алгоритм, отвечающий за такую генерацию не столь надёжен и в нём могут быть неучтённые ошибки. Поэтому, чтобы игра не выключалась при старте, по умолчанию флажок не установлен.

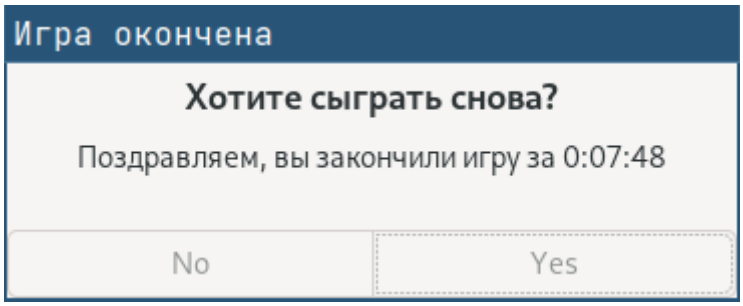
“Помощь”



В нижней части окна расположена строка состояния:

В ней отображаются время, прошедшее с момента начала игры и процент оставшихся камней, которые необходимо убрать.

По окончании игры выводится диалоговое окно с предложением начать игру с такой же картой заново.



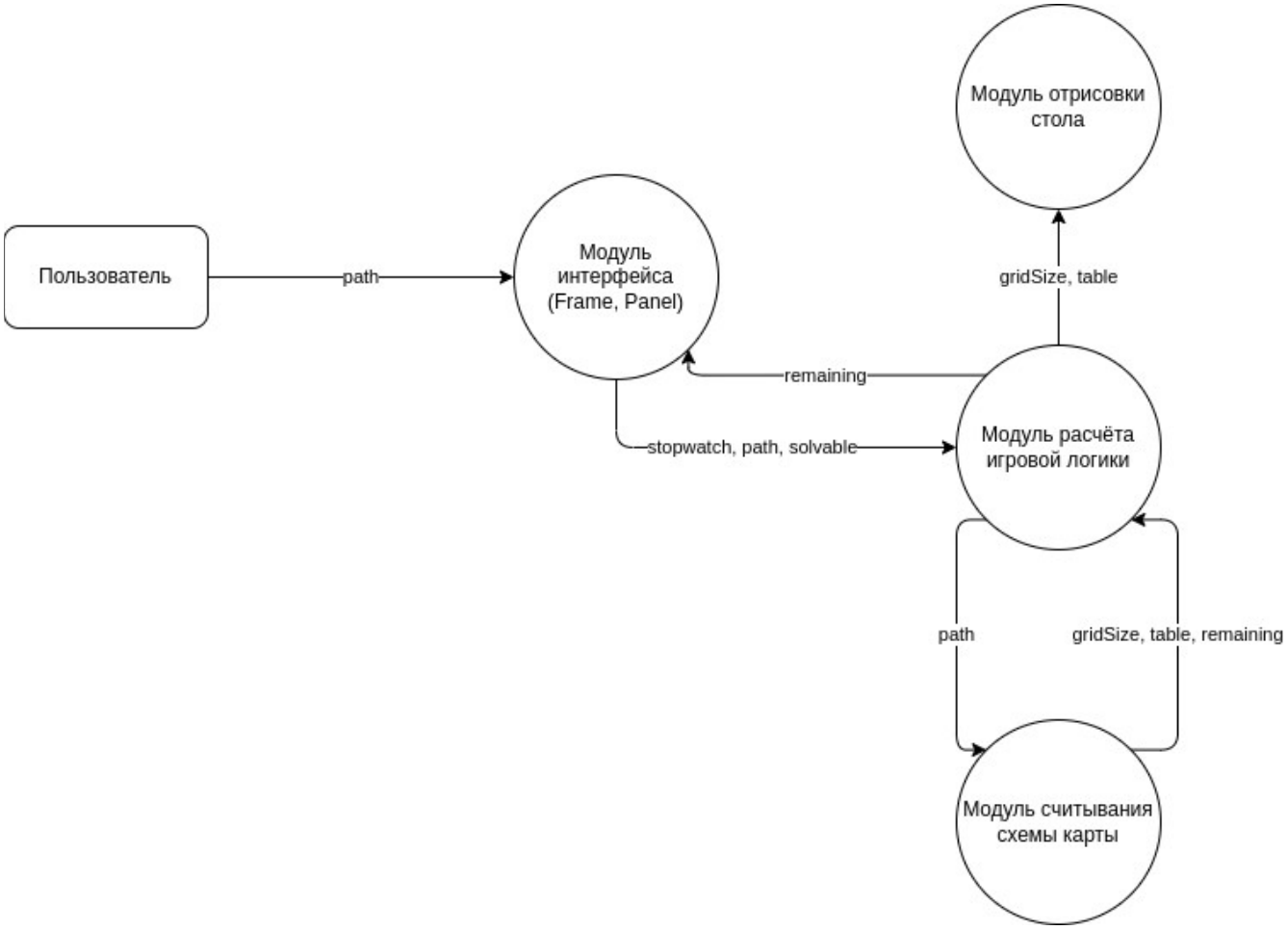
При запуске программы и позже при нажатии “Игра → Открыть карту” (или Ctrl+O) открывается модальный системный диалог выбора файла схемы карты. По умолчанию диалог предлагает выбирать из папки с ресурсами приложения (при корректной установке), однако пользователь может выбрать любой файл формата smlf, хранящийся на диске. Если он выбрал файл и он оказался валидным, начинается игра. Иначе, необходимо снова открывать этот диалог.

Альтернативным способом выбора карты является запуск программы из командной строки, или через ярлык, у которого установлены аргументы. Тогда первый аргумент считается путём до файла карты и при открытии программа сразу же начнёт игру. Однако, если он окажется невалидным, пользователю придётся открывать карту вручную.

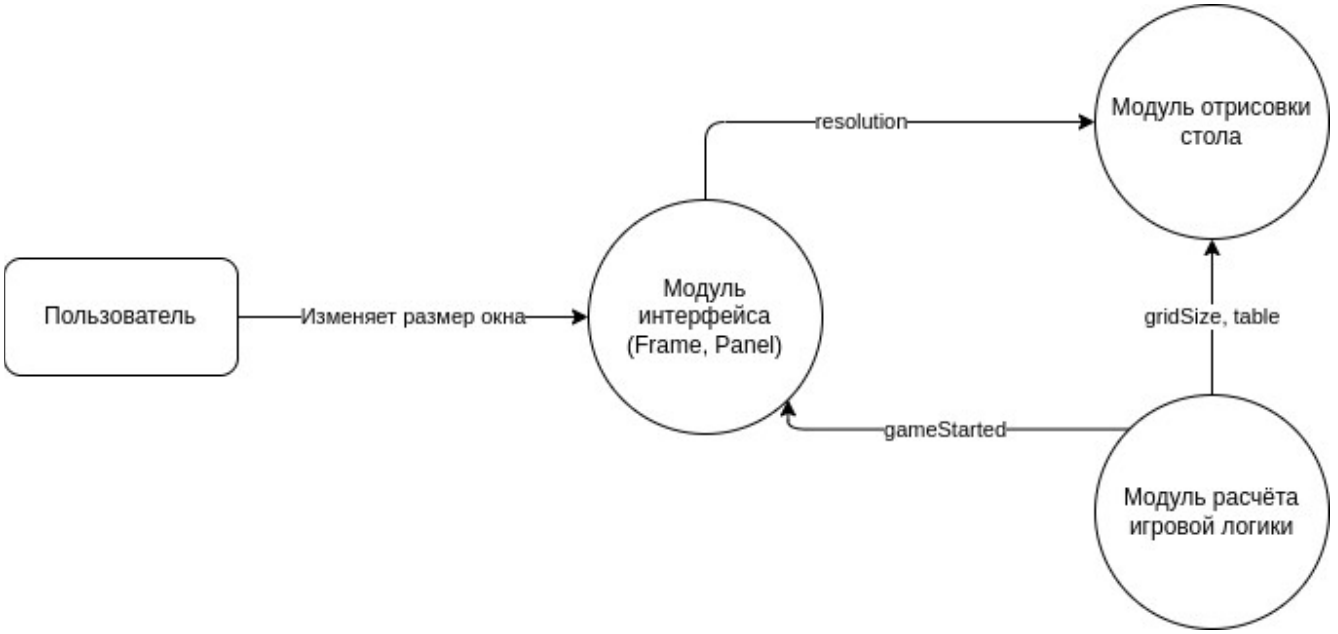
7. Блок-схема движения данных (Data flow diagram)

Поток данных при вызове различных событий:

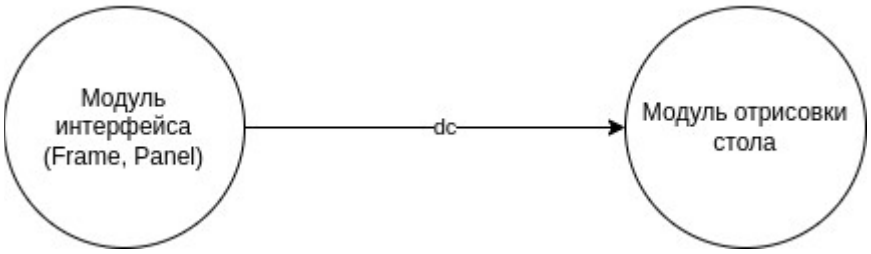
Запуск игры



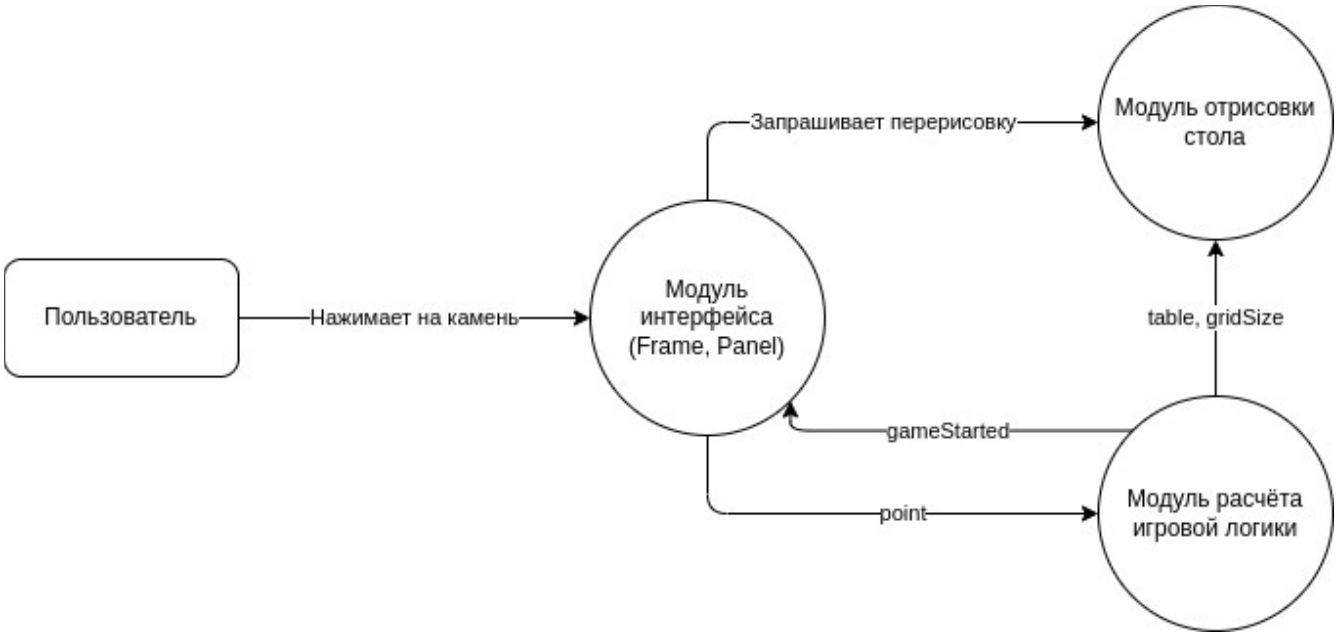
Изменение размера окна



Отрисовка окна

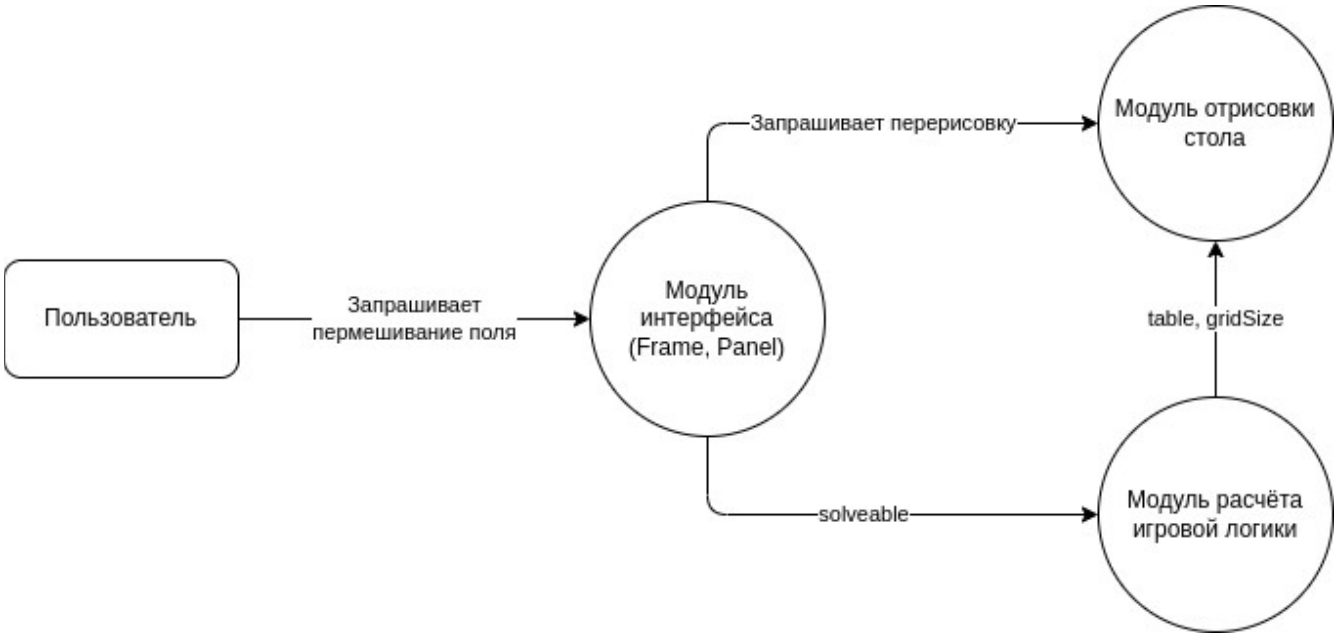


Нажатие левой кнопки мыши

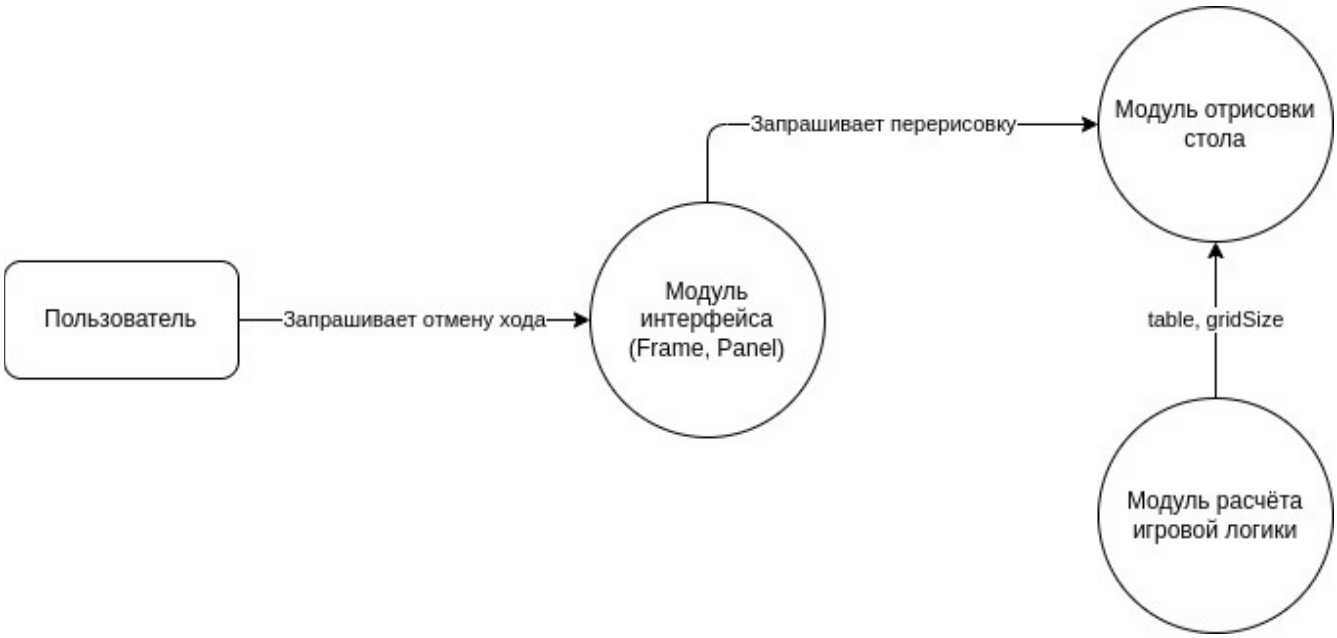


Тик таймера

Перемешивание стола



Отмена хода



8. Выбор и обоснование типов переменных. Разработка структур данных

- class **MyApp –** наследник wxApp, в котором переназначается метод OnInit. Отвечает за запуск приложения, чтение аргументов командной строки и открытие главного окна
- class **Controller** – класс, реализующий логику игры и соединяющий модуль чтения схемы карты из файла с приложением.

  Публичные поля:

  - int **stopwatch** = -1 – счётчик таймера, отсчитывающий то, сколько секунд прошло с момента начала игры. Тип signed, так как принимает отрицательное значение по умолчанию -1 и int для простоты вывода в режиме отладки
  - uint8_t **remaining** – оставшееся количество камней, которые нужно убрать (число от 0 до 144 < 2^8 - 1 = 255, поэтому можно использовать восьмибитный беззнаковый тип)
  - Dimensions **gridSize** – контейнер, хранящий размер сетки в трёх измерениях

  Приватные поля:

  - Drawer& **drawer** – ссылка на drawer, в котором нужно отрисовывать table
  - XmlLayout **layout** – экземпляр класса для считывания схемы поля из файла. Используется в Controller::loadLayout
  - TLVec **table** – трёхмерный вектор, содержащий id камней, которые стоят по данной позиции, или управляющие значения (подробнее в описании TLVec)
  - CardT* **selected** = nullptr – указатель на ячейку table выбранной в данный момент карты
  - std::array<uint8_t, TILE_IMAGES_N> **cardsCounter** – массив, содержащий количество доступных для вставки в table камней с соответствующими его индексам id. Всего имеется TILE_IMAGES_N id, а количество каждой из карт не превышает 4, поэтому восьми бит для него достаточно
  - std::stack<std::array<CardEntry, 2>> **steps** – контейнер для хранения истории ходов. В нём каждый шаг представляет из себя пару позиций и id камней, а так как по std::pair нельзя итерироваться, используется std::array из двух элементов. В качестве контейнера используется стэк, так как для возможности быстрой вставки и удаления последнего элемента подходит этот LIFO-контейнер
- В Controller::getRandLowest:
  - int **overall –** равняется количеству элементов в плоскости x-y table. Так как каждая из сторон массива не может быть больше 287 (144*2 - 1), получается, что произведение не превышает 82369, что помещается в тип int, однако не влезает в uint16_t. Не смотря на то, что на самом деле произведение всегда

будет меньше, чем это число, для того, чтобы не плодить преобразования типов, тут всё-таки используется тип int

- class **Drawer** – класс, отвечающий за рисование интерфейса игры внутри GamePanel при помощи wxDC и операции над размерами в пикселях

  Публичные поля:

  - wxSize **tilePixelSize** – реальные размеры четверти карты (одной клетки в сетке Controller::table)
  - wxSize **resolution** – реальные размеры панели, в которой происходит рисование (GamePanel)
  - wxRect **boardPixelRect** – структура, в которой хранятся одновременно и размеры и положение внутри панели GamePanel битмапа Drawer::boardBitmap
  - ThreePoint **marked** – координаты в сетке Controller::table камня, который необходимо отметить выбранным. Если ничего не выбрано, имеет стандартное значение {-1, -1, -1}

  Приватные поля:

  - wxImage **tileImages**[TILE_IMAGES_N] – массив из изображений, соответствующих индексу id, подгружаемый при создании инстанса Drawer. wxImage поддерживает png формат, благодаря чему картинки можно хранить в более сжатом виде, нежели bmp
  - wxBitmap **bgBitmap –** битмап, в котором рисуется градиентный фон поля. При каждой перерисовке панели копируется в dc
  - wxBitmap **boardBitmap** - битмап, в котором рисуются камни. При каждой перерисовке панели копируется в dc
  - wxPoint **boardPadding** – величина отступов по x и по y внутри Drawer::bgBitmap, нужных для выпирающих частей камней
  - wxPoint **tilePadding –** величина отступов для подложки камня. Так же используется для расчёта поднятия камня по оси z
  - int **prevGridPoint –** предыдущее значение gridPoint. Так как является составляющей размеров, так же имеет тип int.

- const char* **tileImageNames**[TILE_IMAGES_N] – массив из сишных строк, содержащий имена файлов, в которых храняться картинки, соответствующие id камня
- В Drawer::composeBG:
  - const wxColor **lGreen**, **dGreen –** цвета, используемые для рисования градиентного фона. Создаются как глобальные, чтобы не пересоздавать при каждом вызове функции
  - wxMemoryDC **dc** – здесь и далее DC, рисующий не в окне, а в битмапе
- В Drawer::composeBoard:

    - wxMask* **mask –** указатель на маску, выделяющую нарисованные камни в отличие от фона в Drawer::boardBitmap
- В Drawer::drawTile:
    - wxBrush **_bgColor**, **front**, **back –** кисти с устанавливаемым цветом
- В Drawer::composeMinSize:
    - wxSize **ms –** нужен для хранения минимального размера окна, которое можно устанавливать
- const wxEventTypeTag<wxCommandEvent> **START_EVT**, **END_EVT** – кастомные события начала и окончания игры (на данный момент в проекте используется только последний)
- class **GamePanel –** наследник wxPanel. Является поверхностью внутри окна MainFrame, где можно рисовать при помощи wxDC

  Приватные поля:
    - Drawer **drawer –** экземпляр Drawer’a, с помощью которого будет происходить рисование в этой панели
    - Controller **controller** – экземпляр Controller’a, с помощью которого будет расчитываться логика игры в этой панели
    - wxStatusBar* **sb –** указатель на статусбар, принадлежащий родительскому элементу этой панели
    - wxTimer **timer –** таймер, каждую секунду вызывающий GamePanel::OnTimer
- class **MainFrame –** наследник wxFrame, главное окно приложения, внутри которого находится GamePanel, где рисуется игровое поле

  Публичные поля:
    - wxString **layoutPath** – строка, в которой хранится путь до файла схемы карты

  Приватные поля:
    - GamePanel* **panel –** указатель на панель, пренадлежащую этому окну
    - const std::function<void(const wxSize&)> **setMinSize_fn** – класс-функция, инициализируемя лямбдой, устанавливающей минимальный размер окна
    - const wxString dataDirPath – строка, в которой хранится путь до файла схемы карты
    - bool solveable – флаг, устанавливающий, в каком режиме генерировать карту (при старте игры и ресайзе
- В GamePanel::GamePanel:
    - wxMessageDialog **dlg –** диалог для вывода коротких сообщений с выбором
- В GamePanel::openLayout:

    - wxFileDialog **openFileDlg –** системный диалог для выбора файла
- class **TextDlg –** наследник wxDialog, класс для отображения длинного текста в виде диалога с возможностью прокрутки, если текст вышел за пределы окна

  В TextDlg::TextDlg:

    - wxBoxSizer* **sizer** – указатель на сайзер для окна прокрутки
    - wxScrolledWindow* **scrollableWnd** – указатель на окно прокрутки
    - wxStaticText* **text –** указатель на статический текст
    - const wxClientDC **dc –** dc, который может рисовать в реальном окне вне обработчика события wxEVT_PAINT
    - const wxSize& **lineSize –** размеры, которые имела бы строка из 40 символов ‘W’
    - const wxSize& **textSize –** размеры блока с реальным текстом после выравнивания по размеру диалога и с применением переноса
- const wxString TXTContents::**help**, **rules**, **about** – строки, содержащие текст для диалогов
- **CardT** – тип-алиас для int8_t. Используется для обозначения id камней. Принимает значения от –3 до 41 (количество id камней, 42 - 1)
- struct **Dimensions –** наследник wxSize, используется для обозначения размеров объекта в трёх измеоегтях.

  Публичные поля:

    - int **z** – добавленная к wxSize третья координата. Так же как и все остальные размеры и координаты имеет тип int
- struct **ThreePoint** – наследник wxPoint, используется для обозначения координат в трёзмерном пространстве

  Публичные поля:

    - int **z** – добавленная к wxPoint третья координата. Так же как и все остальные размеры и координаты имеет тип int
- struct std::**hash**<ThreePoint> – функтор для хеширования экземпляра класса ThreePoint
- **PosSet** – тип-алиас для std::unordered_set<ThreePoint>, набор ThreePoint, работающий на хэш-таблицах. Имеет константное время вставки и удаления элемента, содержит уникальные значения
- struct **CardEntry –** структура, хранящая характеристики камня для Controller::steps

  Публичные поля:

    - ThreePoint **pos –** позиция камня в Controller::table
    - CardT **id –** id камня

- **TLVec** – тип-алиас для трёхмерного вектора из CardT
- enum **Values** – перечисление, содержащее специальные коды состояния для CardT
  - MATCHED = -3 – камень убран отсюда
  - EMPTY = -2 – пустое место, нельзя вставлять камни
  - FREE = -1 – место, куда можно вставить камень
- class **XmlLayout –** класс для чтения файла схемы карты

  Приватные поля:
  - wxString **path** – строка, содержащая путь до файла
  - int **lx**, **ly** – минимальные координаты в файле
  - wxXmlDocument **layoutDoc** – экземпляр класса для взаимодействия с XML документом

9. Вводимые и выводимые параметры и их типы

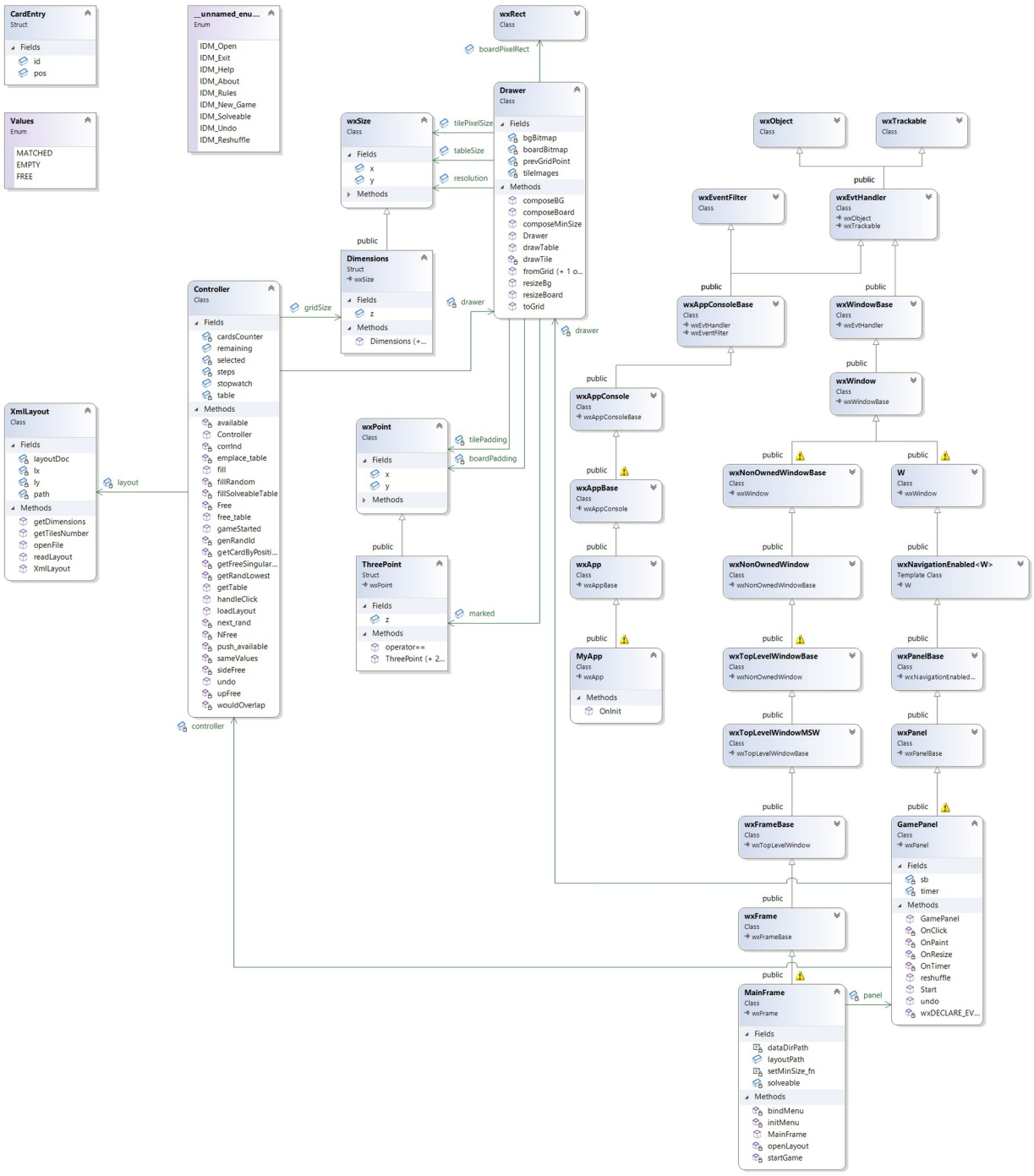
При запуске программы из консоли после имени файла в качестве аргумента можно указать путь до файла карты. Его значение в виде строки Unicode можно получить из проброшенных через кроссплатформенный интерфейс argc, argv.

При помощи стандартного (предоставляемого графической оболочкой ОС) диалога выбора файла пользователь так же передаёт программе путь до файла карты. Система преобразует его в Unicode строку и возвращает в методе диалога GetPath().

Так же в диалоге по окончании игры у пользователя спрашивается, хочет ли он сыграть эту игру снова. Результат его выбора представляется в виде кода, возвращаемого при вызове метода диалога ShowModal() и далее используется как булевое значение сравнения кода с id, который присваивается OK.

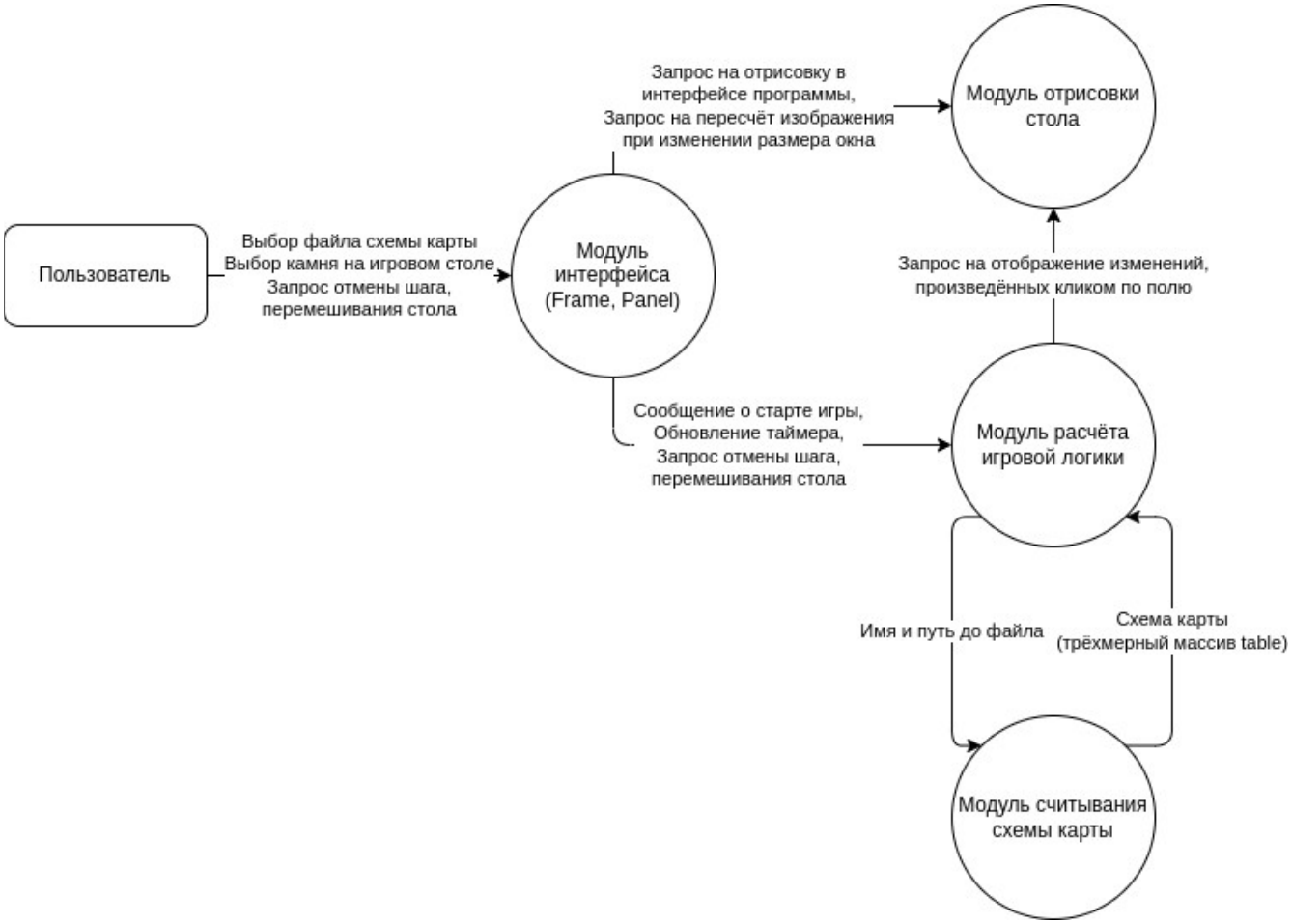
К вводимым данным так же можно отнести положение курсора, получаемое из события, вызываемого при клике мышью по игровому полю. Оно представляется в виде экземпляра предоставляемого библиотекой класса wxPoint, содержащего двумерные координаты (x и y), а так же методы для манипуляций над ним.

10. Диаграмма классов



11. Структура проекта, перечисление нужных файлов

Весь код делится на несколько основных модулей:



Структура файлов:

```
./
├── resources/
│   ├── layouts/
│   ├── tiles/
│   ├── icon.ico
│   └── icon.xpm
├── App.cpp
├── App.h
├── Controller.cpp
├── Controller.h
├── CREDITS
├── Drawer.cpp
├── Drawer.h
├── events.h
├── GamePanel.cpp
├── GamePanel.h
├── gpl.txt
├── LICENSE
├── MainFrame.cpp
├── MainFrame.h
├── Makefile
├── README.md
```

```
├── Resource.rc*
├── TextDlg.cpp
├── TextDlg.h
├── TXTContents.h
├── utils.cpp
├── utils.h
├── wxw.h
├── XmlLayout.cpp
└── XmlLayout.h
```

12. Инструкция по использованию

При запуске программы вы получили запрос на выбор файла с картой. В случае, если вы его выбрали, перед вами должно быть игровое поле и вы можете приступать к игре, при необходимости ознакомившись с правилами (F1, или в меню "Помощь->Правила игры")

Для выбора камня, необходимо нажать на него левой кнопкой мыши. Если он доступен для удаления, он подсветится зелёным цветом. При выборе второго такого же они оба исчезают.

При клике на другой доступный камень (не считающийся с ним одинаковым), выделение с предыдущего снимается и выделяется другой.

При желании вы можете изменить размеры окна. При этом все элементы окна и игровое поле подстроятся под новый.

Для того, чтобы начать игру сначала, можно нажать в меню "Игра->Начать сначала" (или Ctrl+N).

Если вы хотите сменить карту, выберите в меню "Игра->Открыть карту" (или Ctrl+O).

Приложение поддерживает два режима генерации карты: полностью случайный и имеющий хотя бы один способ решения. Для выбора, в каком режиме генерировать, в меню есть флажок "Игра->Генерировать решаемую карту".

Если в процессе игры вам понадобилось отменить ход, или у вас не осталось ходов, вы можете воспользоваться кнопками "Игра->Отменить ход" и "Игра->Перемешать" соответственно.

Для выхода из игры, нажмите "Игра->Выход" (или Ctrl+Q)

Во вкладке "Помощь" так же расположены сведения "О программе", где приведены список использованных ресурсов и лицензии.

В нижней части экрана расположена строка состояния. В ней слева выводится время, прошедшее с начала игры, или подскаски по пунктам меню при наведении на них мыши. Справа выводится процент оставшихся камней, которые необходимо убрать, то есть, в начале игры это 100%, когда останется половина камней, будет 50%, а когда будут убраны все, 0%.

13. Текст программы и файлов заголовков с комментариями

App.h

```
#ifndef APP_H_
#define APP_H_

#include "wxw.h"

class MyApp : public wxApp {
public:
virtual bool OnInit() override;
};

#endif
```

App.cpp

```
#include "App.h"
#include "MainFrame.h"

#include <wx/filefn.h>

wxIMPLEMENT_APP(MyApp);

bool MyApp::OnInit() {
wxImage::AddHandler(new wxPNGHandler());

MainFrame* frame = new MainFrame(); // Создаём окно игры

if (argc ⩾ 2 && wxFileExists(argv[1])) // Если пользователь ввёл какие-то
аргументы
frame→layoutPath = argv[1]; // считаем, что первый аргумент - путь до
файла карты

frame→Show(true); // показываем окно
SetTopWindow(frame); // и устанавливаем главным, а так же выносим вперёд

return true;
}
```

Controller.h

```
#ifndef CONTROLLER_H
#define CONTROLLER_H

#include <array>
#include <stack>
```

```
#include "wxw.h"

#include "Drawer.h"
#include "XmlLayout.h"

class Controller {
public:
Controller(Drawer& drawer) : drawer(drawer){};

int stopwatch = -1;

void loadLayout(const wxString& path);

void handleClick(const wxPoint& point);

TLVec& getTable();

void free_table();

void fill(bool solveable);

uint8_t remaining;

void undo();

bool gameStarted() const;

Dimensions gridSize;

private:
Drawer& drawer;
XmlLayout layout;

TLVec table;

CardT* selected = nullptr;

void fillSolveableTable();

wxPoint getRandLowest() const;

void emplace_table(CardT id, const ThreePoint& pos, PosSet& positions);
void next_rand(PosSet& positions,
PosSet::iterator& ptr, bool canOverlap, uint8_t& not_end);

bool wouldOverlap(const ThreePoint& prev, const ThreePoint& next);

bool corrInd(const ThreePoint& p, const ThreePoint& d) const;
bool Free(const ThreePoint& p, const ThreePoint& d) const;
bool NFree(const ThreePoint& p, const ThreePoint& d) const;
```

```
void push_available(PosSet& positions, const ThreePoint& pos) const;

void fillRandom();

CardT getFreeSingularId(CardT prev);
CardT genRandId();

CardT* getCardByPosition(ThreePoint& point);

bool available(const ThreePoint& point) const;
bool upFree(const ThreePoint& point) const;
bool sideFree(const ThreePoint& point) const;

bool sameValues(CardT a, CardT b) const;

std::array<uint8_t, TILE_IMAGES_N> cardsCounter;

std::stack<std::array<CardEntry, 2>> steps;
};

#endif
```

Controller.cpp

```
#include "Controller.h"

#include <exception>

static const std::array<uint8_t, 42> defaultCardsCounter{
4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 1, 1, 1, 1, 1, 1, 1, 1};

void Controller::loadLayout(const wxString& path) {
layout.openFile(path); // открываем файл карты

gridSize = layout.getDimensions(); // получаем размеры карты

table = TLVec( // создаём трёхмерный вектор из id карт
gridSize.z,
vector<vector<CardT>>(gridSize.x, vector<CardT>(gridSize.y, EMPTY)));

layout.readLayout(table); // считываем формат поля из файла

remaining = layout.getTilesNumber(); // получаем предполагаемое количество
камней

if (remaining == 144) // другие форматы не каноничны, поэтому мы их не
поддерживаем
// Заполняем массив-счётчик карт с различными id
cardsCounter = defaultCardsCounter;
}
```

```
TLVec& Controller::getTable() {
return table;
}

void Controller::fill(bool solveable) {
if (solveable)
fillSolveableTable();
else
fillRandom();
}


void Controller::fillSolveableTable() {
srand(time(NULL)); // инициализируем генератор случайных чисел

auto not_end = remaining; // сохраняем в отдельную переменную количество
оставшихся камней

PosSet positions; // инициализируем сет для хранения доступных для вставки
позиций

positions.insert(getRandLowest()); // вставляем случайную начальную
позицию

auto next_ptr = positions.begin(); // инициализируем указатель на позицию,
куда будет вставляться следующий камень

while (!positions.empty()) {
auto id = genRandId();

emplace_table(id, *next_ptr, positions); // вставляем id в next_ptr
not_end--; // уменьшаем счётчик оставшихся для вставки камней

next_rand(positions, next_ptr, false, not_end); // Находим случайную новую
позицию так, чтобы она не накрывала предыдущую

if (id < 34) // если id парный
cardsCounter[id]--; // уменьшаем счётчик карт этого id ещё на 1, так как
вставим его ещё раз
else
id = getFreeSingularId(id);

emplace_table(id, *next_ptr, positions); // вставляем id в next_ptr
not_end--; // уменьшаем счётчик оставшихся для вставки камней

next_rand(positions, next_ptr, true, not_end); // Находим случайную новую
позицию
}
}

wxPoint Controller::getRandLowest() const {
```

```
int overall = gridSize.x * gridSize.y; // вычисляем количество позиций в
горизонтальном "срезе" массива
int x, y; // объявляем координаты для возвращаемой позиции

do {
int pos = rand() % overall; // получаем случайный номер позиции
x = pos / gridSize.y; // вычисляем x
y = pos % gridSize.y; // и y
} while (table[0][x][y] ≠ FREE); // повторяем цикл, если эта позиция
недоступна для вставки

return wxPoint(x, y); // возвращаем wxPoint
}

void Controller::emplace_table(CardT id, const ThreePoint& pos, PosSet&
positions) {
table[pos.z][pos.x][pos.y] = id;

push_available(positions, pos); // записываем в сет новые позиции
}

#ifdef WXDEBUG // Если компилируем в режиме дебага

#include <wx/file.h>

void print_list(const PosSet& positions) {
wxFile f("tmp.txt", wxFile::write_append); // Открываем файл для записи

for (const auto& el : positions) // Итерируемся по всем позициям
f.Write(itowxS(el.z) + " " + itowxS(el.x) + " " + itowxS(el.y) + "\n"); //
Выводим координаты в файл

f.Write("_ size: " + itowxS(positions.size()) + "\n"); // В конце выводим
количество элементов
}

#endif

void Controller::next_rand(PosSet& positions,
PosSet::iterator& ptr,
bool canOverlap, uint8_t& not_end) {
#ifdef WXDEBUG // Если компилируем в режиме дебага
print_list(positions); // выводим список позиций
#endif

ThreePoint prev = *ptr; // сохраняем предыдущее значение итератора

positions.erase(ptr); // удаляем только что вставленный итератор

if (not_end) { // если ещё есть камни для вставки
if (positions.empty()) // если не осталось позиций,
```

```
ptr = positions.insert(getRandLowest()).first; // вставляем любую позицию
из нижней плоскости и используем её в следующий раз
else { // иначе
ptr = positions.begin(); // устанавливаем итератор в первую позицию

int rand_d = rand() % positions.size(); // получаем случайное смещение
внутри набора возможных позиций

std::advance(ptr, rand_d); // смещаем итератор

const auto rand_ptr = ptr; // сохраняем предыдущее положение итератора

if (!canOverlap) {
while (ptr ≠ positions.end() && wouldOverlap(prev, *ptr)) // Пока не
найдём тот, что не будет закрывать только что вставленную позицию, если не
canBeUp или дошли до конца набора
ptr++; // наращиваем итератор

if (ptr == positions.end()) { // если ни одна из позиций начиная с
rand_ptr не подошла (нельзя выбирать накрывающую предыдущий камень и все
позиции накрывают)
ptr = positions.begin(); // начинаем с начала

while (ptr ≠ rand_ptr && wouldOverlap(prev, *ptr)) // Пока не найдём тот,
что не будет закрывать только что вставленную позицию, если не canBeUp и
не дошли до rand_ptr
ptr++; // наращиваем итератор
}

if (ptr == rand_ptr && wouldOverlap(prev, *ptr)) { // если итератор
совпадает с rand_ptr и при этом ptr перекрывает prev,
if (not_end == positions.size()) // если уже все позиции добавлены в набор
ptr = positions.begin(); // просто выбираем первую из них
else { // иначе
auto res = positions.insert(getRandLowest()); // пытаемся вставить
вставляем случайную позицию в нижней плоскости

while (!res.second) // пока не произошла вставка позиции в набор
res = positions.insert(getRandLowest()); // пытаемся вставить случайную
позицию в нижней плоскости

ptr = res.first; // получаем итератор на только что вставленную позицию
}
}
}
}
}
}

bool Controller::wouldOverlap(const ThreePoint& prev, const ThreePoint&
next) {
```

```
table[next.z][next.x][next.y] = 1; // вставляем в позицию next временный
камень

bool res = !upFree(prev); // проверяем, будет ли свободен сверху камень в
позиции prev

table[next.z][next.x][next.y] = FREE; // удаляем временный камень

return res; // возвращаем результат проверки
}

/**
* Checks if position `p`, shifted by `d`, is not out of bounds of array
`table` with dimensions `gridSize`
*/
bool Controller::corrInd(const ThreePoint& p, const ThreePoint& d) const {
auto& gS = gridSize; // более короткий алиас для переменной

return ((d.z == 0) || (d.z < 0 && p.z >= -d.z) || (d.z > 0 && p.z + d.z <
gS.z)) &&
((d.x == 0) || (d.x < 0 && p.x >= -d.x) || (d.x > 0 && p.x + d.x < gS.x))
&&
((d.y == 0) || (d.y < 0 && p.y >= -d.y) || (d.y > 0 && p.y + d.y < gS.y));
}

/**
* Checks if position `p`, shifted by `d`, is not out of bounds and
available for insert (FREE)
*/
bool Controller::Free(const ThreePoint& p, const ThreePoint& d) const {
return corrInd(p, d) && (table[p.z + d.z][p.x + d.x][p.y + d.y] == FREE);
}

/**
* Checks if position `p`, shifted by `d`, is out of bounds or unavailable
for insert (FREE)
*/
bool Controller::NFree(const ThreePoint& p, const ThreePoint& d) const {
return !corrInd(p, d) || (table[p.z + d.z][p.x + d.x][p.y + d.y] != FREE);
}

/**
* Pushes all positions, that are close to `pos`, available for insert and
don't overlap other available positions
*/
void Controller::push_available(PosSet& positions,
const ThreePoint& pos) const {
auto& p = pos; // короткий алиас для переменной

int z = pos.z, x = pos.x, y = pos.y; // "разбираем" объект pos на
координаты
```

```
// дальше идёт миллион условий, которые проще просто прочитать, нежели
описывать, что они проверяют. В комментариях возле условий указано, в
какую сторону относительно pos смещается вставляемая позиция

if (NFree(p, {-1, -2, 0}) && NFree(p, {-1, -3, 0}) && NFree(p, {-1, -2, -
1}) && NFree(p, {-1, -3, -1}) && NFree(p, {-1, -2, 1}) && NFree(p, {-1, -
3, 1}) && Free(p, {0, -2, 0})) // left
positions.emplace(z, x-2, y);
if (NFree(p, {-1, 2, 0}) && NFree(p, {-1, 3, 0}) && NFree(p, {-1, 2, -1})
&& NFree(p, {-1, 3, -1}) && NFree(p, {-1, 2, 1}) && NFree(p, {-1, 3, 1})
&& Free(p, {0, 2, 0})) // right
positions.emplace(z, x+2, y);

if (NFree(p, {0, 0, -2})) { // half top
if (NFree(p, {-1, -2, -1}) && NFree(p, {-1, -3, -1}) && NFree(p, {-1, -2,
0}) && NFree(p, {-1, -3, 0}) && NFree(p, {-1, -2, -2}) && NFree(p, {-1, -
3, -2}) && NFree(p, {-1, -1, -2}) && Free(p, {0, -2, -1})) // left
positions.emplace(z, x-2, y-1);
if (NFree(p, {-1, 2, -1}) && NFree(p, {-1, 3, -1}) && NFree(p, {-1, 2, 0})
&& NFree(p, {-1, 3, 0}) && NFree(p, {-1, 2, -2}) && NFree(p, {-1, 3, -2})
&& NFree(p, {-1, 1, -2}) && Free(p, {0, 2, -1})) // right
positions.emplace(z, x+2, y-1);
}

if (NFree(p, {0, 0, 2})) { // half bottom
if (NFree(p, {-1, -2, 1}) && NFree(p, {-1, -3, 1}) && NFree(p, {-1, -2,
0}) && NFree(p, {-1, -3, 0}) && NFree(p, {-1, -2, 2}) && NFree(p, {-1, -3,
2}) && NFree(p, {-1, -1, 2}) && Free(p, {0, -2, 1})) // left
positions.emplace(z, x-2, y+1);
if (NFree(p, {-1, 2, 1}) && NFree(p, {-1, 3, 1}) && NFree(p, {-1, 2, 0})
&& NFree(p, {-1, 3, 0}) && NFree(p, {-1, 2, 2}) && NFree(p, {-1, 3, 2}) &&
NFree(p, {-1, 1, 2}) && Free(p, {0, 2, 1})) // right
positions.emplace(z, x+2, y+1);
}

if (NFree(p, {-1, 0, -2}) && NFree(p, {-1, 0, -3}) && NFree(p, {-1, -1, -
2}) && NFree(p, {-1, -1, -3}) && NFree(p, {-1, 1, -2}) && NFree(p, {-1, 1,
-3}) && Free(p, {0, 0, -2})) // top
positions.emplace(z, x, y-2);
if (NFree(p, {-1, 0, 2}) && NFree(p, {-1, 0, 3}) && NFree(p, {-1, -1, 2})
&& NFree(p, {-1, -1, 3}) && NFree(p, {-1, 1, 2}) && NFree(p, {-1, 1, 3})
&& Free(p, {0, 0, 2})) // bottom
positions.emplace(z, x, y+2);

/* Higher */

if (Free(p, {1, 0, 0})) // straight
positions.emplace(z+1, x, y);
```

```
if (NFree(p, {0, -1, -2}) && NFree(p, {0, 0, -2}) && NFree(p, {0, 1, -2})
&& Free(p, {1, 0, -1})) // half top
positions.emplace(z+1, x, y-1);
if (NFree(p, {0, -1, 2}) && NFree(p, {0, 0, 2}) && NFree(p, {0, 1, 2}) &&
Free(p, {1, 0, 1})) // half bottom
positions.emplace(z+1, x, y+1);

if (NFree(p, {0, -2, 0})) {// half left
if (NFree(p, {0, -2, -1}) && NFree(p, {0, -2, 1}) && Free(p, {1, -1,
0})) // straight
positions.emplace(z+1, x-1, y);
if (NFree(p, {0, -2, -1}) && NFree(p, {0, -2, -2}) && NFree(p, {0, 0, -2})
&& Free(p, {1, -1, -1})) // half top
positions.emplace(z+1, x-1, y-1);

if (NFree(p, {0, -2, 1}) && NFree(p, {0, -2, 2}) && NFree(p, {0, 0, 2}) &&
Free(p, {1, -1, 1})) // half bottom
positions.emplace(z+1, x-1, y+1);
}

if (NFree(p, {0, 2, 0})) { // half right
if (NFree(p, {0, 2, -1}) && NFree(p, {0, 2, 1}) && Free(p, {1, 1, 0})) //
straight
positions.emplace(z+1, x+1, y);

if (NFree(p, {0, 2, -1}) && NFree(p, {0, 2, -2}) && NFree(p, {0, 0, -2})
&& Free(p, {1, 1, -1})) // half top
positions.emplace(z+1, x+1, y-1);

if (NFree(p, {0, 2, 1}) && NFree(p, {0, 2, 2}) && NFree(p, {0, 0, 2}) &&
Free(p, {1, 1, 1})) // half bottom
positions.emplace(z+1, x+1, y+1);
}
}

/**
* Removes all set stones and makes their positions free again
*/
void Controller::free_table() {
// Итерируемся по массиву table размерности gridSize
for (int z = 0; z < gridSize.z; z++)
for (int x = 0; x < gridSize.x; x++)
for (int y = 0; y < gridSize.y; y++) {
CardT id = table[z][x][y]; // считываем id данной ячейки

if (id ⩾ 0) { // если это валидный id камня
cardsCounter[id]++; // наращиваем счётчик камней

table[z][x][y] = FREE;
}
}
```

```
steps = decltype(steps)(); // сбрасываем стек steps (decltype удобен, ибо
тогда не зависим от класса, просто вызываем стандартный конструктор)
}

void Controller::fillRandom() {
srand(time(NULL)); // инициализируем генератор случайных чисел

wxLogDebug(itowxS(remaining));

auto not_end = remaining; // сохраняем количество оставшихся для вставки
камней

// итерируемся по всему массиву поля, пока не вставим все камни
for (int z = 0; z < gridSize.z && not_end; z++)
for (int x = 0; x < gridSize.x && not_end; x++)
for (int y = 0; y < gridSize.y && not_end; y++)
if (table[z][x][y] == FREE) { // если в эту позицию можно вставить камень
table[z][x][y] = genRandId(); // получаем случайный id и вставляем его
туда
not_end--; // уменьшаем счётчик оставшихся камней
}
}

CardT Controller::genRandId() {
CardT id;

do {
id = rand() % TILE_IMAGES_N; // получаем случайное число-индекс в массиве
имён камней
} while (cardsCounter[id] == 0); // повторяем тело цикла, если эти id уже
закончились

cardsCounter[id]--; // уменьшаем счётчик оставшихся для вставки камней
этого типа

return id; // возвращаем полученный id
}

CardT Controller::getFreeSingularId(CardT prev) {
CardT id = (prev < 38) ? 34 : 38; // устанавливаем первый из id, которые
считаются одинаковыми с prev

while (id < TILE_IMAGES_N && cardsCounter[id] == 0) // ищем в массиве
оставшихся камней свободный id (если начинаем с 34, так как id выбираются
парами, обязательно останется хотя бы один id, пренадлежащий этой группе
(до 48), поэтому границу можно оставить одинаковой)
id++;

cardsCounter[id]--; // уменьшаем счётчик оставшихся id
```

```
return id; // возвращаем его
}

/**
* Gets pointer to top card by grid position
* It also changes point to top right coordinate of card
*/
CardT* Controller::getCardByPosition(ThreePoint& point) {
int topIndex = -1; // начинаем с -1, чтобы если не нашёлся ни один камень, получить невалидную позицию
CardT* res = nullptr; // указатель на элемент массива

ThreePoint realPos(point); // сохраняем копию позиции, чтобы при смещении не ломать позицию, для которой ищем

// ищем верхнюю карту с верхним левым углом в данной позиции (нажатие в левую верхнюю четверть камня)
for (int z = table.size() - 1; z ⩾ 0; z--)
if (table[z][point.x][point.y] ⩾ 0) {
if (z > topIndex) {
topIndex = z;
res = &table[z][point.x][point.y];
}
break;
}

// ищем верхнюю карту с верхним левым углом в данной позиции, смещённой на единицу влево (нажатие в правую верхнюю четверть камня)
if (point.x > 0)
for (int z = table.size() - 1; z ⩾ 0; z--)
if (table[z][point.x - 1][point.y] ⩾ 0) {
if (z > topIndex) {
topIndex = z;
res = &table[z][point.x - 1][point.y];

realPos.x = point.x - 1;
realPos.y = point.y;
}
break;
}

// ищем верхнюю карту с верхним левым углом в данной позиции, смещённой на единицу вверх (нажатие в левую нижнюю четверть камня)
if (point.y > 0)
for (int z = table.size() - 1; z ⩾ 0; z--)
if (table[z][point.x][point.y - 1] ⩾ 0) {
if (z > topIndex) {
topIndex = z;
res = &table[z][point.x][point.y - 1];

realPos.x = point.x;
```

```
realPos.y = point.y - 1;
}
break;
}

// ищем верхнюю карту с верхним левым углом в данной позиции, одновременно
смещённой вверх и влево (нажатие в правую нижнюю четверть камня)
if (point.x > 0 && point.y > 0)
for (int z = table.size() - 1; z ⩾ 0; z--)
if (table[z][point.x - 1][point.y - 1] ⩾ 0) {
if (z > topIndex) {
topIndex = z;
res = &table[z][point.x - 1][point.y - 1];

realPos.x = point.x - 1;
realPos.y = point.y - 1;
}
break;
}
// обновляем переданную позицию так, чтобы она указывала на левый верхний
угол карты
point.x = realPos.x;
point.y = realPos.y;
point.z = topIndex;

return res;
}

bool Controller::available(const ThreePoint& point) const {
return upFree(point) && sideFree(point);
}

bool Controller::upFree(const ThreePoint& point) const {

if (point.z == table.size() - 1) // если находимся на самом верхнем уровне
по оси z
return true;

return !((table[point.z + 1][point.x][point.y] ⩾ 0) ||
(point.x > 0 && table[point.z + 1][point.x - 1][point.y] ⩾ 0) ||
(point.y > 0 && table[point.z + 1][point.x][point.y - 1] ⩾ 0) ||
(point.x > 0 && point.y > 0 &&
table[point.z + 1][point.x - 1][point.y - 1] ⩾ 0) ||
(point.x < table[point.z].size() - 1 &&
table[point.z + 1][point.x + 1][point.y] ⩾ 0) ||
(point.y < table[point.z][point.x].size() - 1 &&
table[point.z + 1][point.x][point.y + 1] ⩾ 0) ||
(point.x < table[point.z].size() - 1 &&
point.y < table[point.z][point.x].size() - 1 &&
table[point.z + 1][point.x + 1][point.y + 1] ⩾ 0) ||
(point.x > 0 && point.y < table[point.z][point.x].size() - 1 &&
```

```
table[point.z + 1][point.x - 1][point.y + 1] ⩾ 0) ||
(point.x < table[point.z].size() - 1 && point.y > 0 &&
table[point.z + 1][point.x + 1][point.y - 1] ⩾ 0));
}

bool Controller::sideFree(const ThreePoint& point) const {
bool lfree = true;
bool rfree = true;

if (point.x > 1)
lfree =
!((point.y > 0 && table[point.z][point.x - 2][point.y - 1] ⩾ 0) ||
(table[point.z][point.x - 2][point.y] ⩾ 0) ||
(point.y < table[point.z][point.x].size() - 1 &&
table[point.z][point.x - 2][point.y + 1] ⩾ 0));

if (point.x < table[point.z].size() - 2)
rfree =
!((point.y > 0 && table[point.z][point.x + 2][point.y - 1] ⩾ 0) ||
(table[point.z][point.x + 2][point.y] ⩾ 0) ||
(point.y < table[point.z][point.x].size() - 1 &&
table[point.z][point.x + 2][point.y + 1] ⩾ 0));

return lfree || rfree;
}

void Controller::handleClick(const wxPoint& point) {
ThreePoint pos(drawer.toGrid(point)); // переводим позицию в координатах
окна в координаты сетки

if (pos.x > -1) { // если попали по полю
CardT* card = getCardByPosition(pos); // получаем карту, в которую попали
и смещаем позицию в её левый верхний угол

if (pos.z ⩾ 0 && available(pos)) { // если действительно получили карту и
она доступна для убирания
if (selected ≠ nullptr && sameValues(*card, *selected) && // если уже
есть выбранная карта и она такая же, как эта по значению,
selected ≠ card) { // но при этом не является тем же указателем
steps.push({CardEntry{drawer.marked, *selected}, // сохраняем эту пару в
истории
CardEntry{pos, *card}});

*selected = MATCHED; // записываем в доску то, что эти карты убраны
*card = MATCHED;

selected = nullptr; // сбрасываем убранную карту

remaining -= 2; // уменьшаем счётчик оставшихся для убирания карт на 1
```

```
drawer.marked = {-1, -1, -1}; // сбрасываем координаты выбранной карты для
"художника"
} else {
selected = card; // устанавливаем указатель на выбранную сейчас карту
drawer.marked = pos; // устанавливаем координаты выбранной карты для
"художника"
}
}
}
}

bool Controller::sameValues(CardT a, CardT b) const {
if (a == b) // если id карт равны
return true;
else if (a ⩾ 38 && b ⩾ 38) // или они входят в одну
return true;
else if (a ⩾ 34 && a ⩽ 37 && b ⩾ 34 && b ⩽ 37) // из групп, где каждой
карты по одной,но при этом все они считаются одинаковыми
return true;

return false;
}

void Controller::undo() {
if (steps.size()) { // если есть шаги для отмены
for (const CardEntry& entry : steps.top()) // в цикле по каждому из пары
камней
table[entry.pos.z][entry.pos.x][entry.pos.y] = entry.id; // возвращаем его
на доску

remaining += 2; // наращиваем счётчик оставшихся для уборки камней
steps.pop(); // удаляем только что восстановленные камни из истории
}
}

bool Controller::gameStarted() const {
return stopwatch > 0; // если счётчик таймера наращивается, игра началась
}
```

Drawer.h

```
#ifndef DRAWER_H
#define DRAWER_H

#include "wxw.h"

#include "utils.h"

#define TILE_HEIGHT 8
#define TILE_WIDTH 6

#define TILE_IMAGES_N 42
```

```
#define TILE_PADDING_SCALE 1.25

class Drawer {
public:
Drawer();

void drawTable(wxDC& dc) const;

void composeBG();
void composeBoard(const TLVec& table, const Dimensions& gridSize);

void resizeBg(const wxSize& tableSize);
bool resizeBoard(const TLVec& table, const Dimensions& gridSize, bool
force);

wxPoint toGrid(const wxPoint& point) const;
wxPoint fromGrid(int x, int y) const;
wxPoint fromGrid(const wxPoint& point) const;

wxSize composeMinSize(const Dimensions& gridSize) const;

wxSize tilePixelSize; // кратно 3x4, по умолчанию 600x800
wxSize resolution;
wxRect boardPixelRect;

ThreePoint marked;

private:
void drawTile(wxDC& dc, int8_t index, const wxPoint& position,
uint8_t zIndex) const;

wxImage tileImages[TILE_IMAGES_N];

wxBitmap bgBitmap;
wxBitmap boardBitmap;

wxPoint boardPadding;
wxPoint tilePadding;

int prevGridPoint;
};

#endif
```

Drawer.cpp

```
#include "Drawer.h"

#include <wx/filename.h>
#include <wx/stdpaths.h>
```

```
static const char* tileImageNames[TILE_IMAGES_N] = {
// clang-format off
"Pin1", "Pin2", "Pin3", "Pin4", "Pin5", "Pin6", "Pin7", "Pin8", "Pin9",
"Sou1", "Sou2", "Sou3", "Sou4", "Sou5", "Sou6", "Sou7", "Sou8", "Sou9",
"Man1", "Man2", "Man3", "Man4", "Man5", "Man6", "Man7", "Man8", "Man9",
"Chun", "Haku", "Hatsu",
"Nan", "Pei", "Shaa", "Ton",
"Flower1", "Flower2", "Flower3", "Flower4",
"Season1", "Season2", "Season3", "Season4"
// clang-format on
};

Drawer::Drawer() : marked{-1, -1, -1} {
wxString path = wxStandardPaths::Get().GetUserDataDir() +
wxFileName::GetPathSeparator() + _("tiles") +
wxFileName::GetPathSeparator();

for (int i = 0; i < TILE_IMAGES_N; i++) { // В цикле по именам изображений
bool succeed = tileImages[i].LoadFile( // загружаем их
path + _(tileImageNames[i]) + _(".png"), // из стандартной папки для
ресурсов приложения
wxBITMAP_TYPE_PNG); // в формате png
if (!succeed) // В случае ошибки заргрузки выводим сообщение об этом
wxLogDebug(_("failed to load tile ./resources/tiles/") + // с путём,
_(tileImageNames[i]) + _(".png with index") + // именем файла
wxString::Format("%i", i)); // и индексом в массиве
}
}

void Drawer::drawTable(wxDC& dc) const {
dc.DrawBitmap(bgBitmap, 0, 0, false); // отрисовываем в dc битмап с фоном,
начиная из левого верхнего угла без маски
if (boardBitmap.IsOk()) { // Если изображение доски построено
wxLogDebug("Drawing board");
dc.DrawBitmap(boardBitmap, boardPixelRect.GetPosition(), true); //
отрисовываем битмап с ней в установленном при рейсайзе положении,
используя маску
}
}

static const wxColor lGreen{0x07, 0x55, 0x2b};
static const wxColor dGreen{0x01, 0x2d, 0x16};

void Drawer::composeBG() {
bgBitmap = wxBitmap(resolution); // создаём битмап размером со всю панейль

wxLogDebug(
wxString::Format("Rebuild bg %i %i", resolution.x, resolution.y));

wxMemoryDC dc; // создаём dc в памяти
```

```
dc.SelectObject(bgBitmap); // выбираем свежесозданный битмап как холст для
рисования

dc.GradientFillConcentric(wxRect(wxPoint(0, 0), resolution), lGreen, //
рисуем радиальный градиент на весь битмап, переходящий от светлозелёного в
центре
dGreen); // к тёмнозелёному по краям
}

void Drawer::composeBoard(const TLVec& table, const Dimensions& gridSize)
{
boardBitmap = wxBitmap(boardPixelRect.GetSize()); // Создаём битмап
согласно размерам, полученным при последнем ресайзе окна

wxLogDebug(_("Rebuild board"));

wxMemoryDC dc; // создаём dc в памяти
dc.SelectObject(boardBitmap); // выбираем свежесозданный битмап как холст
для рисования

// итерируемся по всем индексам массива table
for (int z = 0; z < gridSize.z; z++)
for (int x = 0; x < gridSize.x; x++)
for (int y = 0; y < gridSize.y; y++) {
CardT c = table[z][x][y]; // получаем id в данной позиции
if (c ⩾ 0) // если тут стоит камень
drawTile(dc, c, fromGrid(x, y), z); // отрисовываем его
}

dc.SelectObject(wxNullBitmap); // отцепляем dc (обычно это происходит при
выходе объекта за поле видимости, но здесь это нужно сделать вручную,
чтобы создать маску)

wxMask* mask = new wxMask(boardBitmap, wxColor(0x00, 0x00, 0x00)); //
создаём маску по полю, чтобы при отрисовке там, где нет камней не было
чёрного фона
boardBitmap.SetMask(mask); // устанавливаем маску для битмапа
}

void Drawer::drawTile(wxDC& dc, int8_t index, const wxPoint& position,
uint8_t zIndex) const {
wxBrush _bgColor = dc.GetBrush(); // сохраняем цвет кисти, которая была в
dc по умолчанию

wxBrush front = wxColor(0xff, 0xff, 0xff); // создаём кисти для лицевой
части камня
wxBrush back = wxColor(0xc8, 0xc8, 0xc8); // и подложки

// wxLogDebug(wxString::Format("%i %i %i - %i %i %i (%i %i %i)", zIndex,
position.x, position.y, marked.z, fromGrid(marked).x, fromGrid(marked).y,
marked.z, marked.x, marked.y));
```

```
if (position == fromGrid(marked) && marked.z == zIndex) { // если данный
камень выбран,
front = wxColor(0xc8, 0xff, 0xc8); // заменяем цвета на аналогичные с
зелёным оттенком
back = wxColor(0xbe, 0xdc, 0xbe);
}

dc.SetBrush(back); // устанавливаем кисть для рисования подложки камня

dc.DrawRoundedRectangle( // рисуем подложку,
position.x - tilePadding.x * (zIndex - 1), // смещённую относительно
плоской позиции на оду единицу смещения вправо и на zIndex единиц влево,
position.y - tilePadding.y * (zIndex - 1), // на единицу смещения вниз и
на zIndex единиц вверх
tilePixelSize.x * 2, tilePixelSize.y * 2, // размер каждой из сторон в два
раза больше, чем размеры на сетке
tilePadding.y // радиус закругления равен одной единице смещения
);

dc.SetBrush(front);

dc.DrawRoundedRectangle( // рисуем лицевую часть,
position.x - tilePadding.x * zIndex, // смещённую относительно плоской
позиции на zIndex единиц смещения влево
position.y - tilePadding.y * zIndex, // и на zIndex единиц вверх
tilePixelSize.x * 2, tilePixelSize.y * 2, // размер каждой из сторон в два
раза больше, чем размеры на сетке
tilePadding.y // радиус закругления равен одной единице смещения
);

dc.SetBrush(_bgColor); // возвращаем изначальный цвет кисти dc

if (tileImages[index].IsOk()) { // если при загрузке картинки не возникло
проблем
wxPoint pos; // верхний левый угол картинки (так же как у подложки) смещён
относительно лицевой части на одну единицу смещения
pos.x = position.x - tilePadding.x * (zIndex - 1);
pos.y = position.y - tilePadding.y * (zIndex - 1);

dc.DrawBitmap( // отрисовываем картинку, масштабируя её под размер камня -
две единицы смещения для отступов
tileImages[index].Scale(tilePixelSize.x * 2 - tilePadding.x * 2,
tilePixelSize.y * 2 - tilePadding.y * 2),
pos);
}
}

void Drawer::resizeBg(const wxSize& resolution) {
if (this->resolution != resolution) { // если размер окна действительно
изменился
```

```
this→resolution = resolution; // устанавливаем новое разрешение и
composeBG(); // перерисовываем фон
}
}

/**
* Resizes tile and whole board bitmap size to the resolution, set in this
* instance
*/
bool Drawer::resizeBoard(const TLVec& table, const Dimensions& gridSize,
bool force) {
bool res = false; // произошла ли полная перерисовка поля, или только был
о рассчитано новое положение битмапа

const int gridPoint = mmin( // минимум из двух осей (по x и по y),
подробнее о формулах в (4)
resolution.x / (gridSize.x * TILE_WIDTH + gridSize.z *
TILE_PADDING_SCALE),
resolution.y * TILE_WIDTH /
(gridSize.y * TILE_HEIGHT * TILE_WIDTH + TILE_HEIGHT * gridSize.z *
TILE_PADDING_SCALE));

wxLogDebug(wxString::Format("Resize board: %i", gridPoint));

if (gridPoint ≠ prevGridPoint || force) { // если gridPoint изменился,
или перерасчёт принудительный
tilePixelSize.Set(gridPoint * TILE_WIDTH, gridPoint * TILE_HEIGHT); //
устанавливаем новый размер половины стороны камня (по сетке)

tilePadding.x = tilePixelSize.x / TILE_WIDTH * TILE_PADDING_SCALE; //
Смещение, даваемое подложками карт вдоль оси x
tilePadding.y = tilePixelSize.y / TILE_WIDTH * TILE_PADDING_SCALE; //
Смещение, даваемое подложками карт вдоль оси y

boardPadding.x = tilePadding.x * (gridSize.z - 1); // Смещение,
создаваемое самыми левыми картами на верхних позициях (их может и не быть,
но проверять это дорого)
boardPadding.y = tilePadding.y * (gridSize.z - 1); // Смещение,
создаваемое самыми верхними (в плоскости xy) картами на верхних позициях
(их может и не быть, но проверять это дорого)

boardPixelRect.SetWidth(
(tilePixelSize.x * gridSize.x) + // Размер только плоских карт
boardPadding.x + // см. выше
tilePadding.x // Смещение, даваемое подложками самых правых
);
boardPixelRect.SetHeight(
(tilePixelSize.y * gridSize.y) + // Размер только плоских карт
boardPadding.y + // см. выше
tilePadding.y // Смещение, даваемое подложками самых нижних (в плоскости
xy)
```

```
);
}

boardPixelRect.SetPosition( // выравниваем по центру окна
wxPoint((resolution.x - boardPixelRect.width) / 2,
(resolution.y - boardPixelRect.height) / 2)
);

if (gridPoint ≠ prevGridPoint || force) { // если gridPoint изменился,
или перерасчёт принудительный
composeBoard(table, gridSize); // перерисовываем стол
res = true; // и сообщаем об этом
}

prevGridPoint = gridPoint; // сохраняем новое значение gridPoint для
"кеширования" стола

return res;
}

wxPoint Drawer::toGrid(const wxPoint& point) const {
wxPoint out(-1, -1); // инициализируем координату не валидными значениями

if (point.x ⩾ boardPixelRect.x + boardPadding.x && // если попали внутрь
стола (за исключением декоративных отступов)
point.x ⩽ boardPixelRect.x + boardPixelRect.width - tilePadding.x &&
point.y ⩾ boardPixelRect.y + boardPadding.y &&
point.y ⩽ boardPixelRect.y + boardPixelRect.height - tilePadding.y) {
out.x = (point.x - boardPixelRect.x - boardPadding.x) /
tilePixelSize.x; // переводим курсор в координаты стола (поэтому вычитаем
положение битмапа и декоративные отступы)
out.y = (point.y - boardPixelRect.y - boardPadding.y) / tilePixelSize.y;
}

return out; // возвращаем либо заглушку, либо, если попали, валидные
координаты
}

/**
* Converts from grid position to board bitmap coordinates
*/
wxPoint Drawer::fromGrid(int x, int y) const {
return {x * tilePixelSize.x + boardPadding.x, y * tilePixelSize.y +
boardPadding.y}; // переводим из координат сетки в позицию внутри битмапа
}

/**
* Converts from grid position to board bitmap coordinates
*/
wxPoint Drawer::fromGrid(const wxPoint& point) const {
```

```
return fromGrid(point.x, point.y); // просто проксируем класс wxPoint в
функцию с двумя аргументами
}

wxSize Drawer::composeMinSize(const Dimensions& gridSize) const {
wxSize ms;

ms.SetWidth(TILE_WIDTH * gridSize.x + gridSize.z * TILE_PADDING_SCALE); //
минимальная ширина экрана при gridPoint = 1
ms.SetHeight(TILE_HEIGHT * gridSize.y + gridSize.z * TILE_PADDING_SCALE *
TILE_HEIGHT / TILE_WIDTH); // минимальная высота экрана при gridPoint = 1

ms += {1, 1}; // добавляем по единице к каждой из сторон, чтобы при
округлении не получить число меньше, чем нужно

wxLogDebug(wxString::Format("MinSize %i %i", ms.x, ms.y));

return ms;
}
```

events.h

```
#ifndef EVENTS_H
#define EVENTS_H

#include "wxw.h"

wxDECLARE_EVENT(START_EVT, wxCommandEvent);
wxDECLARE_EVENT(END_EVT, wxCommandEvent);

#endif
```

GamePanel.h

```
#ifndef GRAPHICS_H
#define GRAPHICS_H

#include "wxw.h"

#include <wx/stopwatch.h>

#include "Controller.h"
#include "Drawer.h"

class GamePanel : public wxPanel {
public:
GamePanel(wxFrame* parent);

void Start(const wxString& path, bool solveable,
std::function<void(const wxSize& size)> setMinSize);

void undo();
void reshuffle(bool solveable);
```

```
private:
Drawer drawer;
Controller controller;

void OnPaint(wxPaintEvent& _);
void OnResize(wxSizeEvent& _);
void OnTimer(wxTimerEvent& _);
void OnClick(wxMouseEvent& _);

wxDECLARE_EVENT_TABLE();

wxStatusBar* sb = nullptr;
wxTimer timer;
};

#define TIMER_ID 1

#endif
```

GamePanel.cpp

```
#include "GamePanel.h"

#include <wx/dcbuffer.h>

#include "events.h"
#include "utils.h"

/**
* Объявляем таблицу обработчиков событий
*/
// clang-format off
wxBEGIN_EVENT_TABLE(GamePanel, wxPanel)
EVT_PAINT(GamePanel::OnPaint)
EVT_SIZE(GamePanel::OnResize)
EVT_TIMER(TIMER_ID, GamePanel::OnTimer)
EVT_LEFT_DOWN(GamePanel::OnClick)
wxEND_EVENT_TABLE();
// clang-format on

GamePanel::GamePanel(wxFrame* parent)
: wxPanel(parent), controller(drawer), // вызываем родительский
конструктор и передаём drawer в конструктор
sb(((wxFrame*)this->GetParent())->GetStatusBar()), timer(this, TIMER_ID) {
// инициализируем указатель на статус бар окна, создаём таймер
SetBackgroundStyle(wxBG_STYLE_PAINT); // Устанавливаем wxBG_STYLE_PAINT
для того, чтобы при вызове OnPaint не мерцало окно
}

void GamePanel::Start(const wxString& path, bool solvable,
std::function<void(const wxSize& size)> setMinSize) {
```

```
wxLogDebug(_("Started game")); // здесь и далее - сообщения, которые
выводятся в консоль в дебаг-версии

controller.stopwatch = 0; // сбрасываем таймер
controller.loadLayout(path); // загружаем в контроллер схему
controller.fill(solvable); // заполняем игровое поле

setMinSize(drawer.composeMinSize(controller.gridSize)); // устанавливаем
минимальный размер окна

timer.Start(1000, wxTIMER_CONTINUOUS); // Запускаем таймер для вызова
события каждую секунду

sb->SetStatusText(LTimeToStr(controller.stopwatch), 0); // В первую
колонку статус бара пишем время игры,
sb->SetStatusText(PRemaining(controller.remaining), 1); // во вторую
процент оставшихся камней

bool redrawn =
drawer.resizeBoard(controller.getTable(), controller.gridSize, true); //
Изменяем размер доски
if (!redrawn) // и если при этом изменился размер камней,
drawer.composeBoard(controller.getTable(), controller.gridSize); //
перерисовываем доску

Refresh(); // вызываем перерисовку окна
}

void GamePanel::undo() {
controller.undo();

drawer.composeBoard(controller.getTable(), controller.gridSize);

Refresh();
}

void GamePanel::reshuffle(bool solvable) {
controller.free_table(); // очищаем стол в контроллере
controller.fill(solvable); // заполняем его заново

drawer.composeBoard(controller.getTable(), controller.gridSize); //
перерисовываем стол

Refresh();
}

void GamePanel::OnPaint(wxPaintEvent& _) {
wxAutoBufferedPaintDC dc(this); // создаём контекст с буфером для
предотвращения мерцания

wxLogDebug(_("OnPaint"));
```

```
drawer.drawTable(dc); // отрисовываем в нём кадр
}

void GamePanel::OnResize(wxSizeEvent& _) {
const wxSize& resolution = GetClientSize(); // получаем размер клиентской
части окна (без рамок, тулбара и статусбара)

wxLogDebug(wxString::Format("OnResize %i %i", resolution.x,
resolution.y));

if (isPositive(resolution)) { // на некоторых платформах первоначальный
размер экрана может установиться в 0, поэтому лучше проверять это
drawer.resizeBg(resolution); // изменяем размер фона

if (controller.gameStarted()) // если уже начали игру
drawer.resizeBoard(controller.getTable(), controller.gridSize, false); //
перерисовываем доску
}

Refresh();
}

wxDEFINE_EVENT(END_EVT, wxCommandEvent); // Определяем событие об
окончании игры

void GamePanel::OnTimer(wxTimerEvent& _) {
controller.stopwatch += 1; // Наращиваем счётчик таймера
sb->SetStatusText(LTimeToStr(controller.stopwatch), 0); // и выводим его
новое значение

if (controller.remaining == 0) { // если убраны все карты,
wxCommandEvent event(END_EVT); // создаём экземпляр события окончания игры
event.SetString(LTimeToStr(controller.stopwatch)); // сохраняем в нём
время игры
wxPostEvent(GetParent(), event); // посылаем событие в родительский класс

timer.Stop(); // останавливаем таймер
controller.stopwatch = 0; // сбрасываем счётчик таймера
}
}

void GamePanel::OnClick(wxMouseEvent& _) {
if (controller.gameStarted()) { // Если игра начата,
controller.handleClick(ScreenToClient(wxGetMousePosition())); // выполняем
обработку клика в контроллере
sb->SetStatusText(PRemaining(controller.remaining), 1); // устанавливаем
процент оставшихся камней в статусбар

drawer.composeBoard(controller.getTable(), controller.gridSize); //
отрисовываем новое поле
```

```
wxLogDebug(wxString::Format(_("Remaining %i"), controller.remaining));

Refresh(); // вызываем перерисовку окна
}
}
```

MainFrame.h

```
#ifndef MainFrame_H_
#define MainFrame_H_

#include "wxw.h"

#include "GamePanel.h"

class MainFrame : public wxFrame {
public:
MainFrame();

wxString layoutPath;

private:
void initMenu();
void bindMenu();

GamePanel* panel;

bool openLayout();
void startGame();

const std::function<void(const wxSize&)> setMinSize_fn;

const wxString dataDirPath;

bool solveable = false; // determites wether to generate solveable or
// completely random map
};

enum {
IDM_Open = wxID_OPEN,
IDM_Exit = wxID_EXIT,
IDM_Help = wxID_HELP,
IDM_About = wxID_ABOUT,
IDM_Rules = wxID_HIGHEST + 1,
IDM_New_Game,
IDM_Solveable,
IDM_Undo,
IDM_Reshuffle
};

#endif
```

MainFrame.cpp

```
#include "MainFrame.h"

#include "TextDlg.h"
#include "TXTContents.h"

#include <wx/filename.h>
#include <wx/stdpaths.h>

#include "events.h"

#include "resources/icon.xpm"

MainFrame::MainFrame()
: wxFrame(nullptr, wxID_ANY, _("Маджонг (пасьянс)"), wxDefaultPosition, // указываем то, что у этого окна нет родителя, оно может иметь любой id, так же устанавливаем заголовок и позицию на усмотрение оконного менеджера
wxSize(800, 600)), // устанавливаем стандартный размер окна
dataDirPath(wxStandardPaths::Get().GetUserDataDir()), // сохраняем стандартный путь до ресурсов программы
setMinSize_fn{[this](const wxSize& size) -> void { // создаём лямбду с замыканием внутри неё методов для установки минимального размера окна и передачи её как аргумент в метод другого класса
this->SetMinClientSize(size); // устанавливаем минимальный размер окна для оконного менеджера
const auto& curr = this->GetClientSize(); // считываем нынешний размер окна
this->SetClientSize({mmax(size.x, curr.x), mmax(size.y, curr.y)}); // если нынешний размер окна меньше, чем требует карта, увеличиваем ту сторону окна до минимальной необходимой
}} {
SetIcon(logo_icon);

initMenu(); // Создание пунктов меню
bindMenu(); // Подключение обработчиков пунктов меню

Bind(wxEVT_SHOW, [this](wxShowEvent& _) -> void { // обработчик события отображения окна
if (!layoutPath.IsEmpty() || openLayout()) // если пользователь выбрал схему
startGame();
});

Bind(END_EVT, [this](wxCommandEvent& evt) -> void { // обработчик кастомного события окончания игры
wxMessageDialog dlg(this, _("Хотите сыграть снова?"),
_("Игра окончена"), wxYES_NO); // Создаём диалог с предложением начать игру заново
dlg.SetExtendedMessage(_("Поздравляем, вы закончили игру за ") +
```

```
evt.GetString()); // Устанавливаем время, затраченное на полное выполнение
карты
if (dlg.ShowModal() == wxID_YES) // Если пользователь хочет,
startGame(); // начинаме игру заново
});

CreateStatusBar(2); // Создаём статусбар с двумя колонками

panel = new GamePanel(this); // Создаём
panel->SetFocus(); // и показываем панель, где отрисовывается игровое поле
}

void MainFrame::initMenu() {
wxMenu* menuGame = new wxMenu; // Создаём подменю
menuGame->Append(IDM_New_Game, _("Начать сначала\tCTRL+N")); // Создаем
пункт меню с id обработчика IDM_New_Game, далее аналогично
menuGame->Append(IDM_Open, _("Открыть карту\tCTRL+O"));
menuGame->AppendCheckItem(IDM_Solveable, _("Генерировать решаемую
карту"));
menuGame->AppendSeparator(); // Добавляем горизонтальный разделитель в
меню
menuGame->Append(IDM_Undo, _("Отменить ход"));
menuGame->Append(IDM_Reshuffle, _("Перемешать поле"));
menuGame->AppendSeparator();
menuGame->Append(IDM_Exit, _("Выход\tCTRL+Q"));

wxMenu* menuHelp = new wxMenu;
menuHelp->Append(IDM_Help, _("Инструкция"));
menuHelp->Append(IDM_Rules, _("Правила игры"));
menuHelp->Append(IDM_About, _("О программе"));

wxMenuBar* menuBar = new wxMenuBar; // Создаём меню бар,
menuBar->Append(menuGame, _("Игра")); // куда подключаем созданные выше
подменю
menuBar->Append(menuHelp, _("Помощь"));

SetMenuBar(menuBar); // И устанавливаем его как основной для этой панели
}

void MainFrame::bindMenu() {
Bind(
wxEVT_MENU, [this](wxCommandEvent& _) -> void { Close(); }, IDM_Exit); //
Вешаем обработчик закртытия игры при выборе соответствующего пункта меню

Bind( // Вешаем обработчик для открытия схемы и запуска соответствующей
игры
wxEVT_MENU,
[this](wxCommandEvent& _) -> void {
if (openLayout()) // запрашиваем в диалоге путь до файла карты и если
получаем,
startGame(); // запускаем игру
```

```
},
IDM_Open);

Bind( // Вешаем обработчик для открытия диалога с "помощью"
wxEVT_MENU,
[this](wxCommandEvent& _) -> void {
(new TextDlg(this, wxID_ANY, _("Помощь"), TXTContents::help))->Show();
},
IDM_Help);

Bind( // Вешаем обработчик для открытия диалога с "правилами"
wxEVT_MENU,
[this](wxCommandEvent& _) -> void {
(new TextDlg(this, wxID_ANY, _("Правила игры"), TXTContents::rules))-
>Show();
},
IDM_Rules);

Bind( // Вешаем обработчик для открытия диалога "о программе"
wxEVT_MENU,
[this](wxCommandEvent& _) -> void {
(new TextDlg(this, wxID_ANY, _("О программе"), TXTContents::about))-
>Show();
},
IDM_About);

Bind( // Вешаем обработчик для запуска новой игры
wxEVT_MENU,
[this](wxCommandEvent& _) -> void {
if (!layoutPath.IsEmpty() || openLayout()) // если ещё не выбран файл с
картой, открываем диалог для его выбора
startGame(); // если путь был, или открыт в диалоге, начинаем игру
},
IDM_New_Game);

Bind( // Вешаем обработчик для установки режима генерации поля
wxEVT_MENU,
[this](wxCommandEvent& evt) -> void { solveable = evt.IsChecked(); }, //
устанавливаем флаг способа генерации поля согласно значению чекбокса в
меню
IDM_Solveable);

Bind( // Вешаем обработчик для отмены хода
wxEVT_MENU, [this](wxCommandEvent& _) -> void { panel->undo(); },
IDM_Undo);

Bind( // Вешаем обработчик для перемешивания поля
wxEVT_MENU,
[this](wxCommandEvent& _) -> void { panel->reshuffle(solveable); },
IDM_Reshuffle);
}
```

```
/**
* Shows a file opening dialog asking for .smlf file if succed, sets its
path to
* layoutPath
* @return true if user have chosen a file, false if cancelled
*/
bool MainFrame::openLayout() {
wxFileDialog openFileDlg( // Создаём диалог для запроса файла схемы
this, _("Открыть карту"),
dataDirPath + wxFileName::GetPathSeparator() + _("layouts"), //
Стандартный путь до файлов приложения
_("Turtle.smlf"), _("Файлы Mahjong карт (*.smlf)|*.smlf"), // Стандартный
файл и поддерживаемые форматы файлов
wxFD_OPEN | wxFD_FILE_MUST_EXIST);

if (openFileDlg.ShowModal() == wxID_CANCEL) // если пользователь не выбрал
файл
return false; // возвращаем false, т.к. открытие карты не успешно

layoutPath = openFileDlg.GetPath(); // сохраняем путь до файла
return true;
}

void MainFrame::startGame() {
panel->Start(layoutPath, solveable, // запускаем игру с путём до схемы,
выбранным режимом заполнения поля
setMinSize_fn // и проброшенной функцией установки минимального размера
окна
);
}
```

TextDlg.h

```
#ifndef TEXTDLG_H
#define TEXTDLG_H

#include "wxw.h"

class TextDlg : public wxDialog {
public:
TextDlg(wxWindow* parent, wxWindowID id, const wxString& title, const
wxString& content);
};

#endif
```

TextDlg.cpp

```
#include "TextDlg.h"

#include <wx/settings.h>
```

```cpp
#include "utils.h"

TextDlg::TextDlg(wxWindow* parent, wxWindowID id, const wxString& title,
const wxString& content)
: wxDialog::wxDialog(parent, id, title) {

wxBoxSizer* sizer = new wxBoxSizer(wxVERTICAL); // создаём сайзер для окна
прокрутки
SetSizer(sizer); // устанавливаем его как главный сайзер окна

wxScrolledWindow* scrollableWnd = new wxScrolledWindow( // создаём окно
прокрутки
this, wxID_ANY, wxDefaultPosition, wxDefaultSize, wxVSCROLL); // указываем
стандартные параметры и говорим, что прокрутка должна быть только
вертикальная

sizer->Add(scrollableWnd, 1, wxGROW | wxALL, 5); // добавляем в сайзер
окно прокрутки с отступами по краям в 5 пикселей

wxStaticText* text = // внутри окна прокрутки создаём статический текст,
содержащий `content`, поддерживающий перенос строки
new wxStaticText(scrollableWnd, wxID_ANY, content, wxDefaultPosition,
wxDefaultSize, wxSP_WRAP);

const wxClientDC dc(text); // создаём dc, используя для него настройки
статического текста
const wxSize& lineSize = dc.GetTextExtent(wxString('W', 40U)); // получаем
из dc размеры строчки из 40 символов 'W' (так как текст удобнее всего
читать, если в нём около 40-60 символов в строке)

scrollableWnd->SetScrollbars(lineSize.x, lineSize.y, 0, 0); //
устанавливаем скорость скролла (количество пикселей, прокручиваемых при
одинарном прокручивании колеса мыши, или нажатие кнопки)

text->Wrap(lineSize.x); // Делаем перенос строк для того, чтобы вместить
текст в ширину строки

const wxSize& textSize = text->GetClientSize(); // получаем размер
статического текста

scrollableWnd->SetVirtualSize(textSize.x, textSize.y); // устанавливаем
виртуальный размер окна прокрутки равным размерам статического текста

SetClientSize( // размер видимого окна устанавливаем равным
textSize.x + 10 + wxSystemSettings::GetMetric(wxSYS_HSCROLL_Y), // по
ширине: ширина текста
mmin(textSize.y, lineSize.y * 30) + 10); // по высоте: минимум из реальной
высоты текста, или 30 строк
// + отступы по 5 пикселей с обеих сторон
}
```

utils.h

```
#ifndef UTILS_H
#define UTILS_H

#include "wxw.h"

#include <unordered_set>
#include <vector>
#include <iterator>

using std::vector;

wxString LTimeToStr(int time);
wxString itowxS(int a);
wxString PRemaining(uint8_t remaining);

#define mmin(a, b) (a + b - abs(a - b)) / 2 // среднее арифметическое
минус половина разницы
#define mmax(a, b) (a + b + abs(a - b)) / 2 // среднее арифметическое плюс
половина разницы

using CardT = int8_t;

struct Dimensions : wxSize { // используется там, где необходимо задать
размеры в трёх координатах
Dimensions(int _z, int _x, int _y) : z(_z), wxSize(_x, _y){};
Dimensions() : wxSize(), z(0){};

int z;
};

struct ThreePoint : wxPoint { // используется там, где необходимо задать
положение в трёх координатах. Так же засчёт наследования от wxPoint, может
быть передан в функцию, принимающую wxPoint
ThreePoint(int _z, int _x, int _y) : z(_z), wxPoint(_x, _y){};
ThreePoint(const wxPoint& a) : z(0), wxPoint(a.x, a.y){};
ThreePoint() : z(0), wxPoint(0, 0){};

bool operator==(const ThreePoint& b) const { // требуется для того, чтобы
использовать в std::unordered_set (так как это множетсво)
return z == b.z && x == b.x && y == b.y;
}

int z;
};

// доопределяем функтор std::hash для параметра шаблона ThreePoint, чтобы
использовать в std::unordered_set
namespace std {
template<> struct hash<ThreePoint> {
size_t operator()(const ThreePoint& p) const {
```

```
return std::hash<uint32_t>()(p.z * 288 * 288 + p.x * 288 + p.y); // координаты точки можно представить как число в 288-ичной системе счисления, так как в крайнем случае (если индексируется прямая линия из всех камней), максимальная координата будет 144*2-1. хэш для этого числа может иметь максимальное значение 287*288*288+287*288+287, что больше, чем 2^16, но меньше, чем 2^32, поэтому используем uint32_t (на самом деле это преобразуется в простое статическое преобразование типа к size_t, но документация совертует делать так)
}
};
}

using PosSet = std::unordered_set<ThreePoint>; // алиас для типа

struct CardEntry { // используется как элемент стека, хранящего историю ходов
ThreePoint pos;
CardT id;
};

using TLVec = vector<vector<vector<CardT>>>;

enum Values { // перечисление псевдо-id в таблице для не занятых реальными id камней позиций
MATCHED = -3, // уже убран
EMPTY, // не должно быть камня
FREE // доступен для вставки камня
};

bool isPositive(const wxSize& size);

#endif
```

utils.cpp

```
#include "utils.h"

wxString LTimeToStr(int time) {
return wxString::Format(_("%d:%02d:%02d"), time / 3600, (time / 60) % 60, time % 60); // переводим количество секунд с начала игры в читабельное время (ч:мм:сс)
}

wxString itowxS(int a) {
return wxString::Format("%i", a);
}

wxString PRemaining(uint8_t remaining) {
return wxString::Format("%i%%", remaining * 100 / 144); // делим количество оставшихся камней на их количество и домножаем на 100, чтобы получить проценты
}
```

```
bool isPositive(const wxSize& size) {
return size.x > 0 && size.y > 0;
}
```

wxw.h

```
#ifndef WXW_H_
#define WXW_H_

#include <wx/wxprec.h>

#ifndef WX_PRECOMP
#include <wx/wx.h>
#endif

// переопределяем макрос для преобразования C-строки в UTF-8 wxString
#ifdef _
#undef _
#endif
#define _(s) wxString::FromUTF8(s)

#endif
```

XMLLayout.h

```
#ifndef XMLLAYOUT_H
#define XMLLAYOUT_H

#include "wxw.h"

#include <wx/xml/xml.h>

#include "utils.h"

class XmlLayout {
public:
XmlLayout();

bool openFile(const wxString& path);
Dimensions getDimensions();
void readLayout(TLVec& table);
uint8_t getTilesNumber();

private:
wxString path;

int lx;
int ly;

wxXmlDocument layoutDoc;
};
```

```
#endif
```

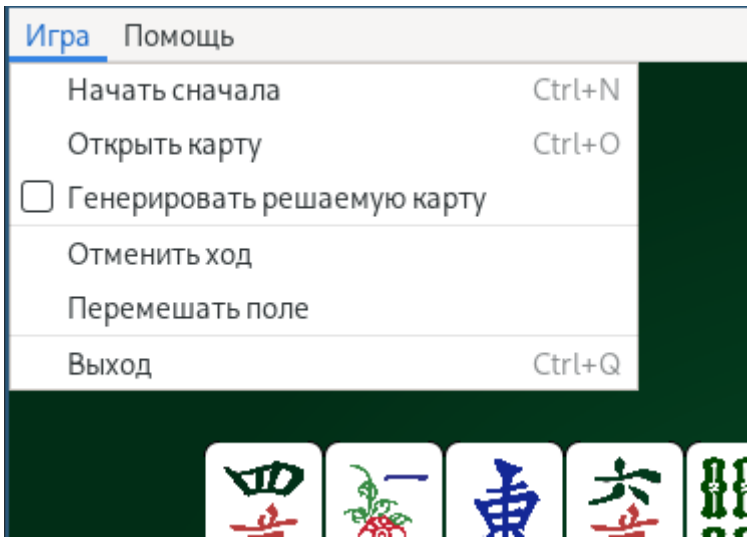
XMLLayout.cpp

```
#include "XmlLayout.h"

XmlLayout::XmlLayout() : path("") {}

bool XmlLayout::openFile(const wxString& openPath) {
if (openPath.IsSameAs(path) && layoutDoc.Load(openPath)) // если открыли
ту же карту, что и в прошлый раз и она открывалась
return true; // можно не читать заново

if (!layoutDoc.Load(openPath)) // если возникла ошибка при открытии файла
return false; // сообщаем об этом

if (layoutDoc.GetRoot() == nullptr) // если не валидный XML
return false; // возвращаем ошибку

return true; // всё хорошо
}

Dimensions XmlLayout::getDimensions() {
auto root = layoutDoc.GetRoot(); // сохраняем указатель на корневой
элемент XML-документа

lx = wxAtoi(root->GetAttribute("lx")); // считываем минимальный x
ly = wxAtoi(root->GetAttribute("ly")); // считываем минимальный y

return {wxAtoi(root->GetAttribute("layers")), // считываем глубину карты
wxAtoi(root->GetAttribute("ux")) + 2 - lx, // считываем ширину карты -
минимальный x
wxAtoi(root->GetAttribute("uy")) + 2 - ly}; // считываем высоту карты -
минимальный y
}

void XmlLayout::readLayout(TLVec& table) {
wxXmlNode* tilePtr = layoutDoc.GetRoot()->GetChildren(); // получаем
первый дочерний элемент корневого XML-элемента

int x, y, l;

while (tilePtr) {
if (tilePtr->GetName().IsSameAs("tile")) { // если это - XML-нода,
описывающая позицию камня
x = wxAtoi(tilePtr->GetAttribute("x")) - lx; // считываем позицию по x и
сдвигаем влево на минимальную позицию (чтобы всегла самый левый камень был
в 0)
y = wxAtoi(tilePtr->GetAttribute("y")) - ly; // считываем позицию по y и
сдвигаем вверх на минимальную позицию (чтобы всегла самый верхний камень
был в 0)
```

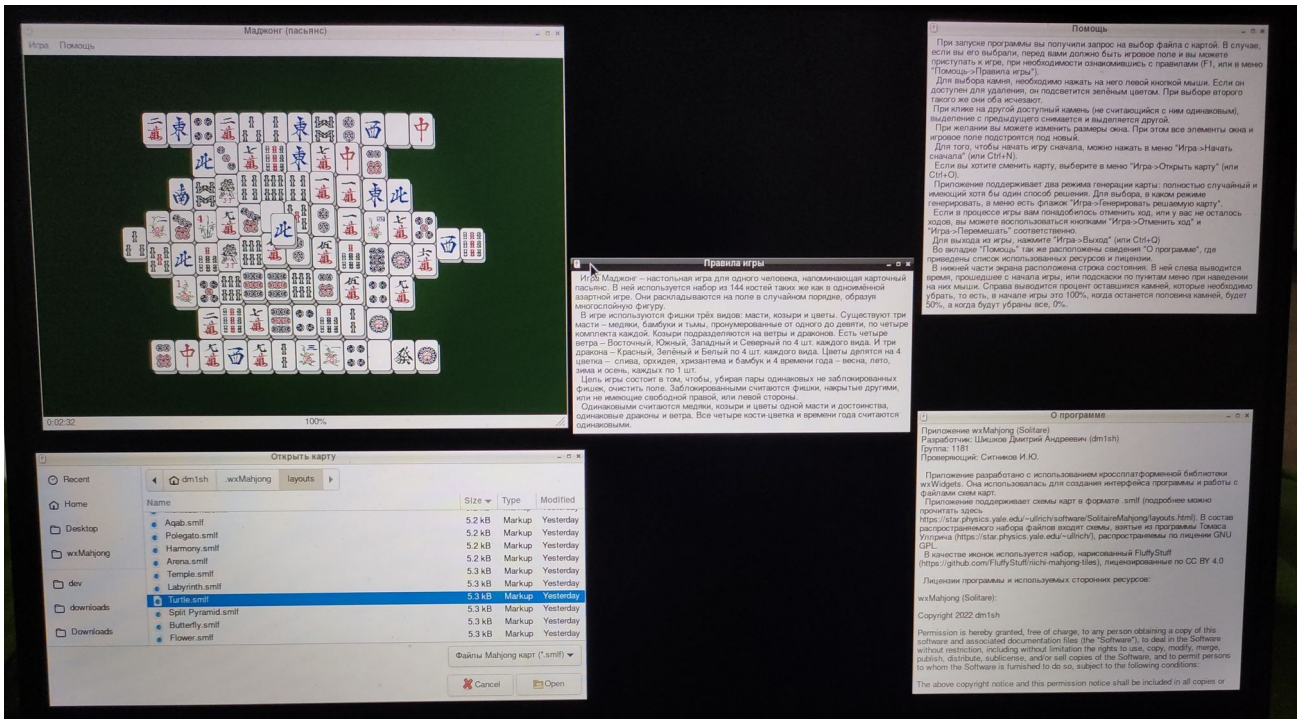
```
l = wxAtoi(tilePtr->GetAttribute("layer")) - 1; // считываем координату по
оси z (начиная с 0)

table[l][x][y] = FREE; // указываем в таблице, что в эту позицию можно
поставить камень
}

tilePtr = tilePtr->GetNext(); // получаем новый элемент из связанного
списка дочерних элементов рут-XML-ноды
}
}

uint8_t XmlLayout::getTilesNumber() {
return wxAtoi(layoutDoc.GetRoot()->GetAttribute("tiles")); // получаем
общее количество используемых id (на случай, если программе подсунули
карту не с 144 камнями)
}
```

14. Рисунки с копиями экрана при работе программы

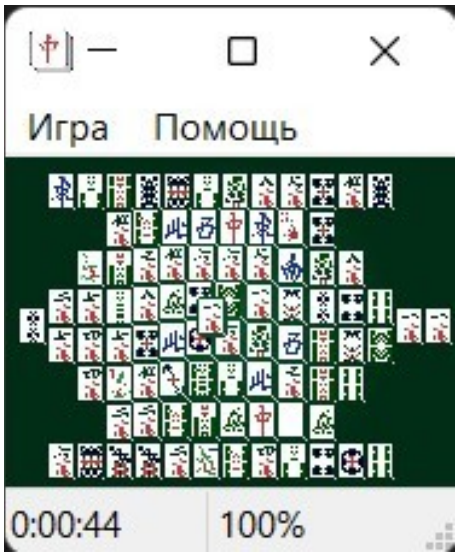
Arch Linux (Wayland, Sway WM)

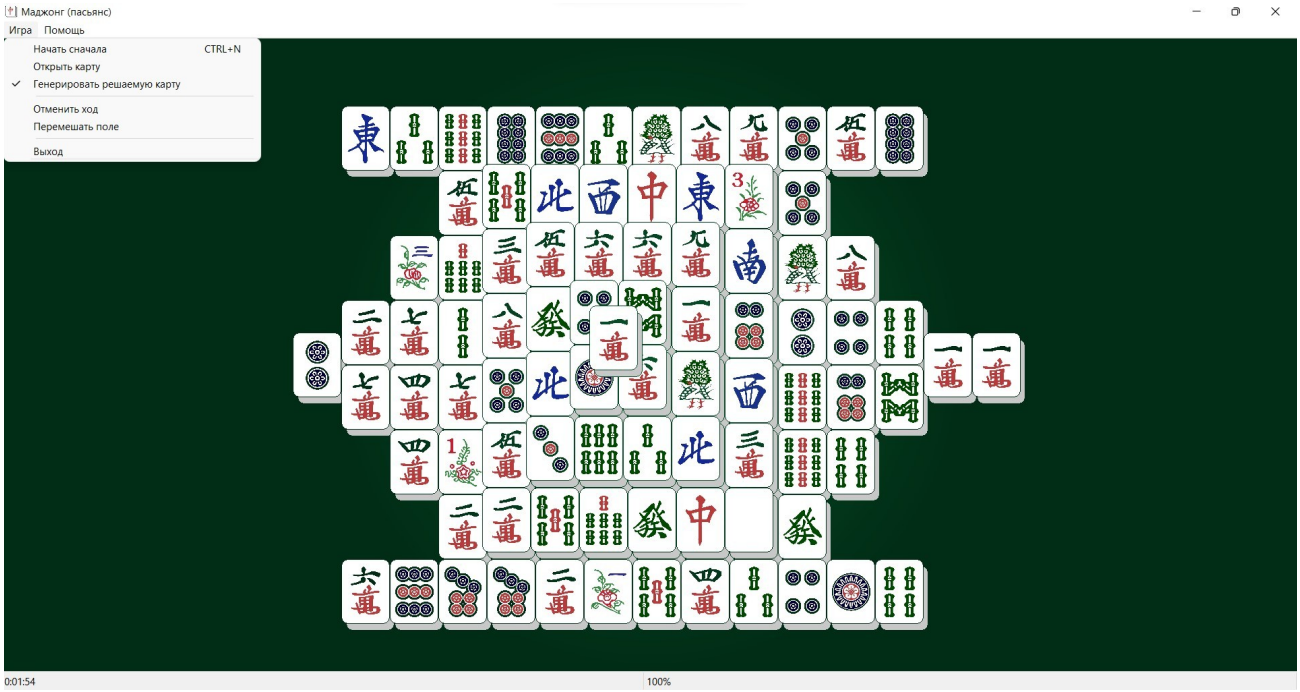


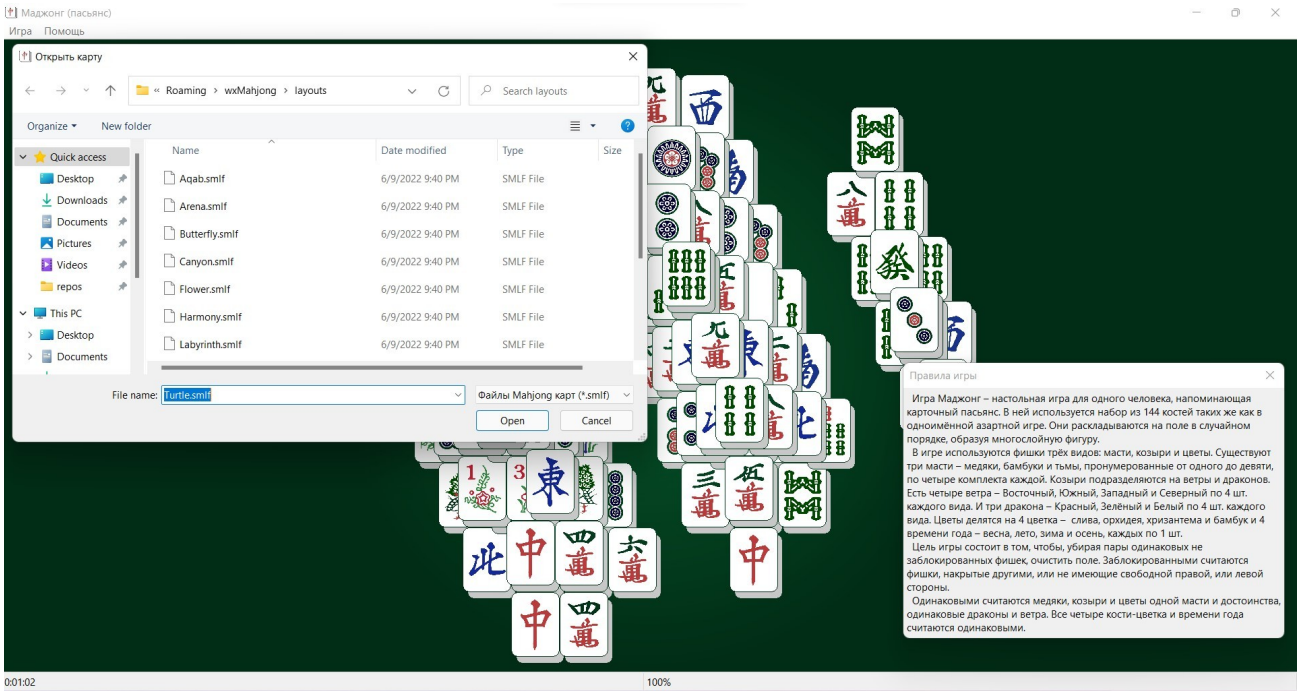
Arch Linux (X11, OpenBox)



Windows 11


Игра
Помощь
0:00:44
100%


Маджонг (пасьянс)
Игра
Помощь
Начать сначала
CTRL+N
Открыть карту
Генерировать решаемую карту
Отменить ход
Перемешать поле
Выход
0:01:54
100%


Маджонг (пасьянс)
Игра Помощь
Открыть карту
Roaming › wxMahjong › layouts
Search layouts
Organize
New folder
Quick access
Desktop
Downloads
Documents
Pictures
Videos
repos
This PC
Desktop
Documents
Name
Date modified
Type
Size
Aqab.smlf 6/9/2022 9:40 PM SMLF File
Arena.smlf 6/9/2022 9:40 PM SMLF File
Butterfly.smlf 6/9/2022 9:40 PM SMLF File
Canyon.smlf 6/9/2022 9:40 PM SMLF File
Flower.smlf 6/9/2022 9:40 PM SMLF File
Harmony.smlf 6/9/2022 9:40 PM SMLF File
Labyrinth.smlf 6/9/2022 9:40 PM SMLF File
File name: Turtle.smlf
Файлы Mahjong карт (*.smlf)
Open
Cancel
Правила игры
Игра Маджонг – настольная игра для одного человека, напоминающая карточный пасьянс. В ней используется набор из 144 костей таких же как в одноимённой азартной игре. Они раскладываются на поле в случайном порядке, образуя многослойную фигуру.
В игре используются фишки трёх видов: масти, козыри и цветы. Существуют три масти – медяки, бамбуки и тьмы, пронумерованные от одного до девяти, по четыре комплекта каждой. Козыри подразделяются на ветры и драконов. Есть четыре ветра – Восточный, Южный, Западный и Северный по 4 шт. каждого вида. И три дракона – Красный, Зелёный и Белый по 4 шт. каждого вида. Цветы делятся на 4 цветка – слива, орхидея, хризантема и бамбук и 4 времени года – весна, лето, зима и осень, каждых по 1 шт.
Цель игры состоит в том, чтобы, убирая пары одинаковых не заблокированных фишек, очистить поле. Заблокированными считаются фишки, накрытые другими, или не имеющие свободной правой, или левой стороны.
Одинаковыми считаются медяки, козыри и цветы одной масти и достоинства, одинаковые драконы и ветра. Все четыре кости-цветка и времени года считаются одинаковыми.
0:01:02
100%

15. Контрольный пример

Файл карты: Turtle.smlf

```
<?xml version="1.0" encoding="UTF-8"?>
<layout name="Turtle" lx="0" ux="28" ly="0" uy="14" layers="5" tiles="144"
verified="1" cs="38881">
<designer name="ullrich" date="Wed Dec 9 07:17:43 2015"/>
<tile layer="1" x="0" y="7"/>
<tile layer="1" x="2" y="0"/>
<tile layer="1" x="2" y="6"/>
<tile layer="1" x="2" y="8"/>
<tile layer="1" x="2" y="14"/>
<tile layer="1" x="4" y="0"/>
<tile layer="1" x="4" y="4"/>
<tile layer="1" x="4" y="6"/>
<tile layer="1" x="4" y="8"/>
<tile layer="1" x="4" y="10"/>
<tile layer="1" x="4" y="14"/>
<tile layer="1" x="6" y="0"/>
<tile layer="1" x="6" y="2"/>
<tile layer="1" x="6" y="4"/>
<tile layer="1" x="6" y="6"/>
<tile layer="1" x="6" y="8"/>
<tile layer="1" x="6" y="10"/>
<tile layer="1" x="6" y="12"/>
<tile layer="1" x="6" y="14"/>
<tile layer="1" x="8" y="0"/>
<tile layer="1" x="8" y="2"/>
<tile layer="1" x="8" y="4"/>
<tile layer="1" x="8" y="6"/>
<tile layer="1" x="8" y="8"/>
<tile layer="1" x="8" y="10"/>
<tile layer="1" x="8" y="12"/>
<tile layer="1" x="8" y="14"/>
<tile layer="1" x="10" y="0"/>
<tile layer="1" x="10" y="2"/>
<tile layer="1" x="10" y="4"/>
<tile layer="1" x="10" y="6"/>
<tile layer="1" x="10" y="8"/>
<tile layer="1" x="10" y="10"/>
<tile layer="1" x="10" y="12"/>
<tile layer="1" x="10" y="14"/>
<tile layer="1" x="12" y="0"/>
<tile layer="1" x="12" y="2"/>
<tile layer="1" x="12" y="4"/>
<tile layer="1" x="12" y="6"/>
<tile layer="1" x="12" y="8"/>
<tile layer="1" x="12" y="10"/>
<tile layer="1" x="12" y="12"/>
<tile layer="1" x="12" y="14"/>
<tile layer="1" x="14" y="0"/>
<tile layer="1" x="14" y="2"/>
<tile layer="1" x="14" y="4"/>
<tile layer="1" x="14" y="6"/>
<tile layer="1" x="14" y="8"/>
<tile layer="1" x="14" y="10"/>
<tile layer="1" x="14" y="12"/>
<tile layer="1" x="14" y="14"/>
<tile layer="1" x="16" y="0"/>
<tile layer="1" x="16" y="2"/>
<tile layer="1" x="16" y="4"/>
```

```
<tile layer="1" x="16" y="6"/>
<tile layer="1" x="16" y="8"/>
<tile layer="1" x="16" y="10"/>
<tile layer="1" x="16" y="12"/>
<tile layer="1" x="16" y="14"/>
<tile layer="1" x="18" y="0"/>
<tile layer="1" x="18" y="2"/>
<tile layer="1" x="18" y="4"/>
<tile layer="1" x="18" y="6"/>
<tile layer="1" x="18" y="8"/>
<tile layer="1" x="18" y="10"/>
<tile layer="1" x="18" y="12"/>
<tile layer="1" x="18" y="14"/>
<tile layer="1" x="20" y="0"/>
<tile layer="1" x="20" y="2"/>
<tile layer="1" x="20" y="4"/>
<tile layer="1" x="20" y="6"/>
<tile layer="1" x="20" y="8"/>
<tile layer="1" x="20" y="10"/>
<tile layer="1" x="20" y="12"/>
<tile layer="1" x="20" y="14"/>
<tile layer="1" x="22" y="0"/>
<tile layer="1" x="22" y="4"/>
<tile layer="1" x="22" y="6"/>
<tile layer="1" x="22" y="8"/>
<tile layer="1" x="22" y="10"/>
<tile layer="1" x="22" y="14"/>
<tile layer="1" x="24" y="0"/>
<tile layer="1" x="24" y="6"/>
<tile layer="1" x="24" y="8"/>
<tile layer="1" x="24" y="14"/>
<tile layer="1" x="26" y="7"/>
<tile layer="1" x="28" y="7"/>
<tile layer="2" x="8" y="2"/>
<tile layer="2" x="8" y="4"/>
<tile layer="2" x="8" y="6"/>
<tile layer="2" x="8" y="8"/>
<tile layer="2" x="8" y="10"/>
<tile layer="2" x="8" y="12"/>
<tile layer="2" x="10" y="2"/>
<tile layer="2" x="10" y="4"/>
<tile layer="2" x="10" y="6"/>
<tile layer="2" x="10" y="8"/>
<tile layer="2" x="10" y="10"/>
<tile layer="2" x="10" y="12"/>
<tile layer="2" x="12" y="2"/>
<tile layer="2" x="12" y="4"/>
<tile layer="2" x="12" y="6"/>
<tile layer="2" x="12" y="8"/>
<tile layer="2" x="12" y="10"/>
<tile layer="2" x="12" y="12"/>
<tile layer="2" x="14" y="2"/>
<tile layer="2" x="14" y="4"/>
<tile layer="2" x="14" y="6"/>
<tile layer="2" x="14" y="8"/>
<tile layer="2" x="14" y="10"/>
<tile layer="2" x="14" y="12"/>
<tile layer="2" x="16" y="2"/>
<tile layer="2" x="16" y="4"/>
<tile layer="2" x="16" y="6"/>
<tile layer="2" x="16" y="8"/>
<tile layer="2" x="16" y="10"/>
<tile layer="2" x="16" y="12"/>
<tile layer="2" x="18" y="2"/>
```

```
<tile layer="2" x="18" y="4"/>
<tile layer="2" x="18" y="6"/>
<tile layer="2" x="18" y="8"/>
<tile layer="2" x="18" y="10"/>
<tile layer="2" x="18" y="12"/>
<tile layer="3" x="10" y="4"/>
<tile layer="3" x="10" y="6"/>
<tile layer="3" x="10" y="8"/>
<tile layer="3" x="10" y="10"/>
<tile layer="3" x="12" y="4"/>
<tile layer="3" x="12" y="6"/>
<tile layer="3" x="12" y="8"/>
<tile layer="3" x="12" y="10"/>
<tile layer="3" x="14" y="4"/>
<tile layer="3" x="14" y="6"/>
<tile layer="3" x="14" y="8"/>
<tile layer="3" x="14" y="10"/>
<tile layer="3" x="16" y="4"/>
<tile layer="3" x="16" y="6"/>
<tile layer="3" x="16" y="8"/>
<tile layer="3" x="16" y="10"/>
<tile layer="4" x="12" y="6"/>
<tile layer="4" x="12" y="8"/>
<tile layer="4" x="14" y="6"/>
<tile layer="4" x="14" y="8"/>
<tile layer="5" x="13" y="7"/>
</layout>
```

Для него генерируется следующая карта:

16. Ведомость соответствия программы спецификации

Требования к расчету и программе	
Реализовать игру Маджонг:	
Игровое поле с отображением костей и взаимодействие с ними	x
Генерацию игрового поля по заданной схеме	x
Таймер и процент решения	x
Сохранение истории ходов нынешней игры (с возможностью отката)	x
Поддерживать загрузку своих схем из файлов формата .smlf	x
Иметь графический пользовательский интерфейс	x
Применять разработанную схему передачи и преобразования данных в проекте Data Flow Diagram с программными интерфейсами передачи данных и диаграммой классов с указанием наследования и программных интерфейсов	x
Использовать определенный индивидуальный тип данных для хранения данных об одиночной кости. Оперировать динамическими массивами (для сохранения истории ходов)	x
Требования к отчету	
Отчёт должен соответствовать ГОСТу 19.701-90 единой системы программной документации	x
В отчёт необходимо включить описание программного интерфейса, диаграмму классов и диаграмму потоков данных, выбор и обоснование переменных, собственных типов и классов, код с комментариями, пример работы программы и контрольный пример	x
Контрольный пример должен быть представлен в виде сгенерированного по заданной схеме поля	x
Требования к пользовательскому интерфейсу программы	
Содержит сведения о программе	x
Содержит сведения об авторе	x

Содержит сведения об авторских правах	x
Имеет название и иконку	x
Имеет главное окно с игровым полем	x
Имеет окно справки с правилами игры	x
Используется меню для открытия файла со схемой, дополнительных окон	x
Используются кнопки управления приложением	x
Содержатся чекбоксы (флажки)	x
Соответствует понятию "дружественный интерфейс"	x
Использует русский язык	x

17. Выводы, включающие область применения, ограничения, достоинства и недостатки программы, размер исполняемого файла на диске

В результате проделанной работы была получена реализация игры маджонг (пасьянс) с использованием библиотеки wxWidgets.

Достоинства

- Приложение кроссплатформенно и может запускаться на любых системах, поддерживающих wxWidgets (протестировано в Linux (X11, Wayland) GTK, Windows 11)
- Поддерживает любые карты формата smlf.
- Может использовать любые наборы картинок, если их правильно назвать и расположить в папке ресурсов
- Поддерживает масштабирование окна, благодаря чему неплохо выглядит как на больших, так и на маленьких мониторах
- Имеет два режима заполнения поля: полностью случайный и решаемый
- Даёт возможность пользователю отменить свой ход, или перемешать поле
- Даёт возможность указать файл схемы карты в качестве аргумента (командной строки, или в ярлыке)
- Засекает время решения карты пользователем

Недостатки

- Медленная отрисовка окна в ОС Windows (невозможно плавное изменение размеров окна)
- Отсутствие автоматического установщика (пользователь должен вручную скопировать файлы)
- Отсутствие сохранения истории игр в файл
- Алгоритм генерации решаемой карты не достаточно протестирован и может генерировать нерешаемую карту или вызывать ошибку

Размер исполняемого файла на диске

В ОС Windows 11 скомпилированное в 32-битном режиме в Visual Studio 2022:

- 8.0K ./resources/layouts
- 1.7M ./resources/tiles
- 1.7M ./resources
- 134 README.txt

- 2.4M wxbase316u_vc14x.dll
- 157K wxbase316u_xml_vc14x.dll
- 325K wxMahjong.exe
- 7.1M wxmsw316u_core_vc14x.dll
- 12M .

B Arch Linux (clang++, wxWidgets GTK3, after strip)

- 383K wxMahjong
- 1.4M /usr/local/lib/libwx_gtk3u_xrc-3.1.so.6.0.0
- 1.1M /usr/local/lib/libwx_gtk3u_html-3.1.so.6.0.0
- 236K /usr/local/lib/libwx_gtk3u_qa-3.1.so.6.0.0
- 11M /usr/local/lib/libwx_gtk3u_core-3.1.so.6.0.0
- 96K /usr/local/lib/libwx_baseu_xml-3.1.so.6.0.0
- 492K /usr/local/lib/libwx_baseu_net-3.1.so.6.0.0
- 3.1M /usr/local/lib/libwx_baseu-3.1.so.6.0.0

18. Ссылки

- https://www.wxwidgets.org - сведения о библиотеке
- https://star.physics.yale.edu/~ullrich/software/SolitaireMahjong/layouts.html – карты в .smlf формате
- https://en.wikipedia.org/wiki/Mahjong_solitaire – Википедия
- https://web.archive.org/web/20120331105950/http://www.personeel.unimaas.nl/uiterwijk/Theses/BSc/Stam_BSc-paper.pdf – статья об различных алгоритмах решения
- http://www.formauri.es/personal/pgimeno/mj/mjsol.html – страница с описанием ИИ для решения
- http://home.halden.net/vkp/vkp/history.html – большой сайт с информацией об игре