Практические задания

Выпуск

Nickolay Kudasov

02 March 2016

Глава 1

Методические указания

Цель практического задания — разработка законченного приложения или программной библиотеки на языке программирования Haskell.

1.1 Групповая работа

Каждое задание может быть выполнено как в одиночку, так и группой в несколько человек (максимальное количество человек в группе зависит от задания).

В каждом задании выделена базовая часть. Все участники должны разбираться в реализации базовой части и быть в состоянии воспроизвести её.

Помимо базовой части в задании присутствует несколько дополнительных частей, по одной на каждого участника. Таким образом, если задание выполняется в одиночку, студент должен реализовать базовую и хотя бы одну дополнительную часть.

Для эффективной работы (в том числе в одиночку) предлагается:

- обсуждать структуру проекта и каждой части до реализации (работать вместе проще, когда вы работаете в одном направлении);
- разбивать реализацию на логические модули (например, дополнительные части заданий должны находиться в разных модулях и минимально зависеть друг от друга);
- пользоваться системой контроля версий (например, Git);
- придерживаться правил оформления исходного кода (хорошо написанный код легко читать, проверять и модифицировать);
- проводить просмотр кода (code review) других участников проекта (это позволяет улучшить общее представление о проекте, уменьшает шанс логических и других ошибок в общем проекте, позволяет поддерживать исходный код оформленным).

1.2 Правила оформления исходного кода

Программирование — это в равной мере искусство и ремесло. И, как известно любому художнику, ограничения расширяют, а не подавляют творческие способности.

Правила оформления исходного кода:

- используйте camelCase для именования функций и переменных;

- используйте описательные названия функций, пусть они будут настолько длинные, насколько необходимо, но не длинее этого. Хорошо: `solveRemaining`. Плохо: `slv`. Ужасно: `solveAllTheCasesWhichWeHaven'tYetProcessed`.
- **не** используйте символы табуляции: Haskell чувствителен к отступам и табуляция может сильно испортить вам жизнь. Заметьте, что это не значит, что вам придётся нажимать пробел миллион раз: ваш любимый текстовый редактор умеет заменять табуляцию на пробелы автоматически.
- старайтесь не писать строчки длинее 80 символов. Код, который не приходится пролистывать по горизонтали читать обычно удобнее.
- описывайте тип для каждой функции на верхнем уровне. Сигнатура типа значительно улучшает документацию и способствует правильному мышлению. Также явное указание типов способствует лучшим сообщениям об ошибках при компиляции. Локально определенные функции и константы (определенные при помощи `let` и `where`) не нуждаются в сигнатурах, но их выписывание не навредит (это также улучшает сообщения об ошибках).
- формулируйте развёрнутый комментарий для каждой функции на верхнем уровне.
- используйте флаг `-Wall` при компиляции или напишите `{-# OPTIONS_GHC -Wall #-}` на самом начале файла с кодом. Этот флаг включает все предупредительные сообщения компилятора.
- по возможности разбивайте ваш код на простые функции (каждая с одним ясным предназначением) и составляйте из них программу;
- пишите всюду определённые функции: они не должны завершаться аварийно ни на одном возможном входе.

1.3 Средства разработки

1.3.1 Stack

Для разработки проекта рекомендуется использовать Stack.

Stack может установить нужную версию компилятора и необходимых библиотек. Для установки компилятора необходимо запустить команду `stack setup`.

Для установки зависимостей Stack использует Stackage — стабильный репозиторий пакетов. Для выполнения практического задания рекомендуется использовать последний доступный LTS Haskell.

Для сборки проекта используйте команду `stack build` или `stack test`.

Для запуска интерпретатора GHCi с автоматической загрузкой модулей проекта используйте команду `stack ghci`.

Для запуска исполняемой программы (например, графического интерфейса) используйте команду `stack exec <имя программы>`.

Для автоматической сборки документации проекта используйте команду `stack haddock`.

1.3.2 Haddock

По возможности, используйте используйте разметку Haddock при написании комментариев в коде. Таким образом вы можете легко получить полноценную документацию кода вашего проекта. Вы можете локально собрать Haddock документацию, используя команду `stack haddock`.

Вам и вашей команде будет проще разобраться в проекте, если он будет хорошо документирован. Кроме того, навык написания хорошей документации не раз может пригодиться вам за пределами данного курса.

1.4 Рекомендации по оформлению задания

Каждое практическое задание можно реализовать множеством способов с различными наборами возможностей. Для того, чтобы другим людям (в том числе соучастникам и преподавателям) было проще разобраться в законченном проекте, рекомендуется:

- добавить файл `README`:
  - с описанием проекта;
  - инструкциями по установке и запуску;
  - описанием реализованных возможностей;
- оформить код в виде проекта Stack:
  - для сборки проекта должно быть достаточно выполнить команду `stack build`;
  - запуск интерпретатора GHCi с автоматической загрузкой модулей проекта осуществляется командой `stack ghci`;
  - запуск исполняего файла проекта — командой `stack exec <имя программы>`;
  - тестирование проекта осуществляется командой `stack test`;
- если вы храните основную версию репозитория на GitHub, вы можете использовать Travis CI для автоматической сборки и тестирования вашего проекта.

1.5 Рекомендации по выбору библиотек

При реализации некоторых частей практических заданий может потребоваться использование сторонних библиотек — например, для графических интерфейсов, клиент-серверной архитектуры, работы с базой данных, генерации кода и пр.

1.5.1 Синтаксический разбор

Для синтаксического разбора рекомендуется использовать комбинаторные библиотеки — например, Parsec или attoparsec. Parsec предоставляет более выразительные средства и лучше подходит для разбора исходного кода и конфигурационных файлов. `attoparsec` предлагает более простой интерфейс и меньше возможностей, но на несколько порядков лучше по производительности и подходит для разбора сетевых протоколов, логов, бинарных данных.

1.5.2 Генерация кода

Для генерации объектного кода проще всего использовать существующий низкоуровневый язык программирования, из которого уже можно легко получить объектный код. К таким языкам относятся C, C– и язык LLVM. Последний часто используется в компиляторах, поскольку специально создан для этой цели.

Генерация кода для LLVM на Haskell реализуется при помощи библиотеки llvm-general.

1.5.3 Графический интерфейс

Библиотека gloss предоставляет простой и удобный интерфейс для работы с векторной 2D графикой. Для игр рекомендуется использование модулей `Graphics.Gloss.Interface.Pure.Game` или `Graphics.Gloss.Interface.IO.Game`. Для моделирования можно использовать модули `Graphics.Gloss.Interface.Pure.Simulate` или `Graphics.Gloss.Interface.IO.Simulate`.

Для игр также стоит использовать библиотеку gloss-game, которая предоставляет несколько удобных функций для работы со сценами.

1.5.4 Клиент-серверная архитектура

Для большинства практических заданий в качестве протокола общения между клиентом и сервером можно использовать HTTP. При реализации HTTP сервера рекомендуется использовать архитектуру REST.

Существует множество web-фреймворков для реализации серверной части. Для выполнения практических заданий рекомендуется использовать использовать один из следующих: - servant — относительно простой в использовании и в то же время

> мощный фреймворк для работы с REST API; в отличие от большинства других фреймворков покрывает не только серверную, но и клиентскую части, а так же автоматическую документацию, инструменты для тестирования, генерация клиентского кода для других языков программирования;

- spock — неплохой фреймворк с неплохой документацией; некоторые возможности требуют хорошенько разобраться, но для выполнения практического задания они необязательны; использовать в паре с wreq для клиентской части;
- scotty — наверное, самый простой фреймворк; использовать в паре с wreq для клиентской части;

Для более тесной связи клиента и сервера можно использовать протокол TCP. Соответствующая библиотека — network-simple.

Для передачи данных по сети рекомендуется использовать сериализацию/десериализацию данных. В случае HTTP предлагается использовать формат JSON (используя библиотеку aeson). В случае TCP — бинарное представление (используя библиотеку binary).

1.5.5 Многопоточность

Серверные приложения используют многопоточность, чтобы взаимодействовать одновременно с множеством клиентов. Приложения с графическим интерфейсом используют многопоточность, чтобы избежать эффекта замирания во время потенциально длительных расчётов (например, в реализации ИИ) или сетевого взаимодействия.

Для использования общей памяти между потоками одного приложения в Haskell используется программная транзакционная память. Соответствующая библиотека stm входит в список стандартных пакетов. В практических заданиях достаточно использования TVar и, возможно, TChan.

Для ознакомления с программной транзакционной памятью, рекомендуется прочтение статьи Software Transactional Memory.

1.5.6 База данных

Для работы с базой данных рекомендуется использовать библиотеку persistent. Эта библиотека предоставляет интерфейс, не зависящий от конкретной используемой СУБД и поддерживает как минимум

PostgreSQL, SQLite, MySQL and MongoDB. Для сложных запросов (например, по нескольким таблицам) предлагается использовать библиотеку esqueleto, которая работает поверх библиотеки persistent.

Глава 2

Интерпретаторы

2.1 Лямбда-исчисление

2.1.1 Описание

λ-исчисление (лямбда-исчисление) лежит в основе большинства функциональных языков программирования: семейства Лиспа (Common Lisp, Scheme, Clojure и др.) и семейства ML (Standard ML, Haskell, Agda и пр.).

λ-исчисление состоит из языка λ-выражений и набора правил преобразования. Базовые правила построения λ-выражений:

- переменная x является λ-выражением;
- если e — λ-выражение, а x — переменная, то $(\lambda x.\ e)$ также λ-выражение (**лямбда-абстракция**);
- если e_1 и e_2 — λ-выражения, то $(e_1\ e_2)$ также λ-выражение (**аппликация**).

Для удобства работы с λ-выражениями, при записи могут использоваться следующие упрощения:

внешние скобки могут быть опущены $(\lambda x.\ e_1)\ e_2$;

аппликация считается лево-ассоциативной $e_1\ e_2\ e_3 \equiv ((e_1\ e_2)\ e_3)$;

тело λ-выражения распространяется вправо насколько возможно $\lambda x.\ e_1\ e_2 \equiv \lambda x.\ (e_1\ e_2)$;

последовательные λ-абстракции схлопываются в одну $\lambda x.\ \lambda y.\ \lambda z.\ e \equiv \lambda x\ y\ z.\ e$.

Оператор λ *связывает* переменную в выражении $\lambda x.\ e$. Переменные, подпадающие под какой-либо оператор λ-абстракции, называются *связанными*. Все прочие переменные называются *свободными*. Например, переменная y свободна в выражении $\lambda x.\ y\ x$. Переменная связывается *ближайшим* оператором λ. Например, единственное вхождение переменной x в выражении $\lambda x.\ y\ (\lambda x.\ x)$ связано со вторым оператором λ.

Значение λ-выражения определяется правилами редукции:

α-конверсия переименовывание связанных переменных

β-редукция применение функции к аргументам

η-конверсия выражает принцип *две функции идентичны, если имеют одинаковый результат на всех входах*

Применение функции к аргументам осуществляется засчёт подстановки выражения-аргумента вместо связанной переменной в теле λ-выражения:

$$x[x \mapsto e] \equiv e$$
$$y[x \mapsto e] \equiv y, \text{если } y \neq x$$
$$(e_1\ e_2)[x \mapsto e] \equiv (e_1[x \mapsto e])\ (e_2[x \mapsto e])$$
$$(\lambda x.\ e_1)[x \mapsto e] \equiv \lambda x.\ e_1$$
$$(\lambda y.\ e_1)[x \mapsto e] \equiv \lambda y.\ (e_1[x \mapsto e]), \text{если } y \neq x \text{ и } y \text{ не входит свободно в } e$$

Язык λ-выражений может быть расширен:

- базовыми типами данных (например, числа, булевы значения, строки);
- базовыми контейнерными типами (списки и кортежи);
- встроенными операциями (например, $+$, $-$, sin, cos, and, or, $++$)
- специальными конструкциями (например, $if \ldots then \ldots else \ldots$ или $let \ldots = \ldots in \ldots$);
- системой типов;
- пользовательские структуры данных;
- и т.д.

2.1.2 Минимальные требования (базовая часть)

Базовая реализация проекта, в которой должны разбираться все участники, должна:

- определять структуру данных для синтаксического дерева;
- содержать раздельно парсер и интерпретатор (с любой стратегией редукций);
- предоставлять простую интерактивную среду программирования (REPL).

2.1.3 Расширенный парсер (дополнительная часть)

Расширенный парсер должен добавлять хотя бы 2 различные возможности к базовому варианту. Ниже перечислены возможные варианты расширения парсера, однако этим списком они не ограничены:

- разбор расширенного λ-исчисления;
- восстановление после ошибок (например, если пользователь написал запятую (`,`) вместо точки (`.`), парсер может запомнить эту ошибку и продолжить разбор программы);
- поддержка пользовательских инфиксных операций с возможностью задать приоритет и ассоциативность;
- и т.д.

2.1.4 Система типов (дополнительная часть)

Система типов помогает определить семантику λ-выражений, но также может быть использована для оптимизаций при интерпретации и генерации кода.

Система типов должна поддерживать хотя бы 2 различные возможности:

- базовые типы (числа, булев тип, строки);
- контейнерные типы (списки, кортежи, суммы);
- параметрический полиморфизм;
- пользовательские типы данных;

- механизм автоматического вывода типа для терма;
- и т.д.

2.1.5 Расширенная интерактивная среда (дополнительная часть)

Расширенная интерактивная среда должна добавлять хотя бы 2 различные возможности к базовому интерфейсу. Ниже перечислены возможные варианты расширения интерактивной среды, однако этим списком они не ограничены:

- команды интерпретатора:
  - показать все возможные варианты редукции терма (одного шага) и выбрать один;
  - показать тип выражения;
  - поменять порядок редукции;
  - перевести терм из/в кодировку Чёрча;
  - загрузить программу из файла;
- интерпретация расширенного λ-исчисления;
- дружелюбные сообщения об ошибках (например, для замкнутых термов при опечатке в имени переменной можно предложить имена переменных, отличающихся одной буквой, которые находятся в области видимости);
- и т.д.

2.1.6 Генерация кода (дополнительная часть)

Модуль генерации кода — предпоследний этап компиляции. Генерация кода может быть реализована многими способами, но чтобы простым образом получить портируемый компилятор, можно генерировать промежуточный код на низкоуровневом языке программирования, таком как C или еще ниже, например, LLVM.

Генерация кода должна переводить именованные λ-термы в соответствующие функции (для этого язык должен быть расширен возможностью именования λ-термов).

Демонстрация генерации кода должна включать в себя программу на любом языке, использующую сгененированный объектный код при сборке.

2.2 Лисп

2.2.1 Описание

В данном задании предлагается реализовать интерпретатор диалекта языка программирования Лисп.

Базовым объектом Лисп является S-выражение и в предлагаемом диалекте оно может быть представлено

- атомом (`IAMATOM`, `numberp`, `setf`),
- числовым литералом (`10`, `34.2`),
- строковым литералом (`"hello, world!"`),
- пустым списком (`nil`, `()`),

- точечной парой (`(1 . 2)`).

Как обычно, непустой список представляется точечной парой, вторым элементом которой является другой список: `(1 "asd"atom)` эквивалентно `(1 . ("asd". (atom . nil)))`.

Каждое S-выражение может быть вычислено по следующим правилам:

- числовые, строковые литералы, пустой список и атом вычисляются в себя;
- список `(f ...)` вычисляется применением функции (или раскрытием макроса) `f` к остальным элементам списка;
- функция перед применением вычисляет все свои аргументы;
- макросы и специальные формы не вычисляют аргументы перед применением.

Программа на этом диалекте состоит из последовательности S-выражений.

Диалект языка выражений может быть расширен:

- встроенными специальными формами:
  - `quote` (`'`);
  - `if`;
  - `let`;
  - `macro`;
  - `setf`;
- пользовательскими функциями: `lambda`, `defun`;
- лексическим/динамическим связыванием;
- операциями ввода-вывода;
- функциями с переменным числом аргументов;
- и т.д.

2.2.2 Минимальные требования (базовая часть)

Базовая реализация проекта, в которой должны разбираться все участники, должна:

- определять структуру данных для синтаксического дерева;
- содержать раздельно парсер и интерпретатор;
- предоставлять простую интерактивную среду программирования (REPL);
- предоставлять встроенные предикаты и функции:
  - `atomp`, `numberp`, `listp`, `=`;
  - `car`, `cdr`, `cons`.

2.2.3 Расширенный парсер (дополнительная часть)

Расширенный парсер должен добавлять хотя бы 2 различные возможности к базовому варианту. Ниже перечислены возможные варианты расширения парсера, однако этим списком они не ограничены:

- разбор расширенного диалекта;

- восстановление после ошибок (например, если пользователь написал лишнюю закрывающую скобку, парсер может запомнить эту ошибку и продолжить разбор программы);
- и т.д.

2.2.4 Расширенная интерактивная среда (дополнительная часть)

Расширенная интерактивная среда должна добавлять хотя бы 2 различные возможности к базовому интерфейсу. Ниже перечислены возможные варианты расширения интерактивной среды, однако этим списком они не ограничены:

- команды интерпретатора:
  - загрузить программу из файла;
  - провести один шаг вычисления;
  - раскрыть вызовы макросов;
- интерпретация расширенного диалекта;
- дружелюбные сообщения об ошибках (например, при опечатке в имени переменной можно предложить имена переменных, отличающихся одной буквой, которые находятся в области видимости);
- и т.д.

2.2.5 Генерация кода (дополнительная часть)

Модуль генерации кода — предпоследний этап компиляции. Генерация кода может быть реализована многими способами, но чтобы простым образом получить портируемый компилятор, можно генерировать промежуточный код на низкоуровневом языке программирования, таком как C или еще ниже, например, LLVM.

Генерация кода должна переводить пользовательские функции в соответствующие функции целевого языка (для этого диалект должен быть расширен возможностью определения пользовательских функций).

Демонстрация генерации кода должна включать в себя программу на любом языке, использующую сгенерированный объектный код при сборке.

Встроенные языки программирования

3.1 Генетика Менделя

3.1.1 Описание

В данном задании предлагается реализовать программную библиотеку, представляющую встроенный язык программирования для моделирования наследования признаков по Менделю.

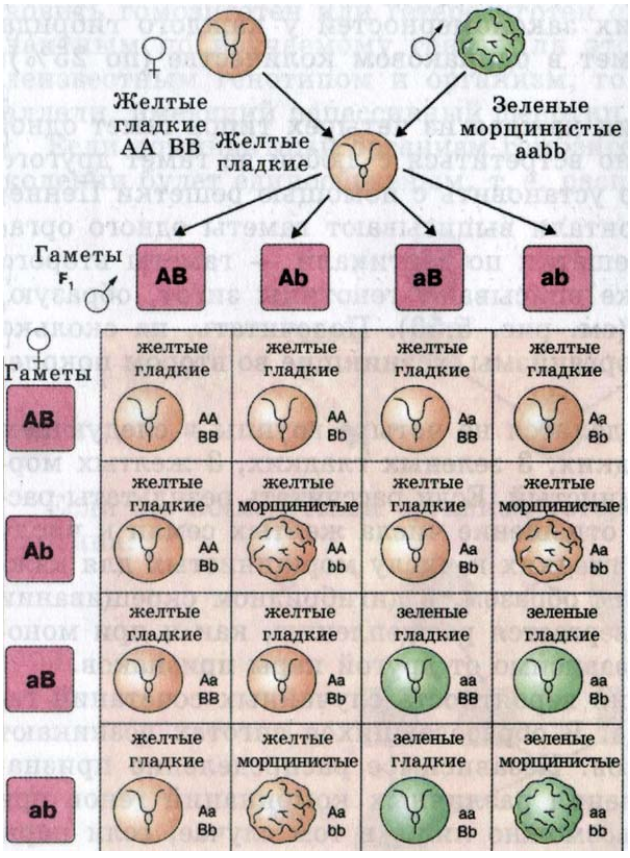

Fig. 3.1: Независимое наследование признаков.

Предсказывание наличия признаков у поколений основывается на нескольких законах Менделя:

Первый закон Менделя Каждый организм обладает парой *аллелей* (мы рассматриваем диплоидные организмы) для каждого отдельного свойства. При размножении для передачи потомку выбирается одна аллель случайным образом.

Второй закон Менделя. Различные признаки организма наследуются независимо друг от друга.

Третий закон Менделя. Аллели бывают доминантные и рецессивные. При наличии обеих аллелей организм проявляет признак, закодированный в доминантной аллели. Таким образом, если один родитель имеет две доминантные аллели, то первое потомство всегда будет проявлять доминантный признак.

Генотип — это набор аллелей для каждого признака (например, $AAbbCc$).

Фенотип — это набор проявляющихся свойств (например, AbC).

Основные задачи, связанные с генетикой Менделя:

- определение признаков (n-ого) потомства (процентные соотношения) при известном генотипе родителей;
- определение генотипа родителей при известном фенотипе родителей и потомства.

Созданная библиотека должна предоставлять удобные стредства для решения подобных задач.

3.1.2 Минимальные требования (базовая часть)

Базовая реализация проекта, в которой должны разбираться все участники, должна:

- описывать структуры, представляющие генотип, фенотип и распределение вероятностей в поколении;
- предоставлять функции для задания генотипа организма;
- предоставлять функцию для вычисления первого поколения потомства.

3.1.3 Расширенная библиотека (дополнительная часть)

Библиотека для работы с генетикой Менделя должна быть расширена как минимум двумя различными возможностями:

- отдельный модуль для работы с распределениями вероятностей;
- пользовательские признаки/наборы признаков (библиотека не должна зависеть от какого-то заданного набора признаков);
- не менделевская генетика:
  - множественные аллели (каждый признак может диктоваться не двумя, а произвольным количеством аллелей);
  - ко-доминантные аллели (аллели, которые не доминируют друг над другом; например, из трёх аллелей две могут доминантны над третей, но ко-доминантны между собой);
  - многогенное наследование;
  - и т.п.;
- функции определения потомства с и без вовлечения родителей в процессы скрещивания потомства;
- функции определения возможного генотипа родителей по распределению вероятностей потомства (любого поколения);

- и т.д.

3.1.4 Графический интерфейс (дополнительная часть)

Графический интерфейс должен предоставлять визуальное представление фенотипа, буквенное представление генотипа, и визуальное разделение поколений.

Графический интерфейс должен реализовывать хотя бы 2 различные возможности:

- визуализация наследования признаков при помощи решётки Пеннета;
- расчёт следующего поколения (с выбором организмов предыдущих поколений, которые будут вовлечены в процесс скрещивания);
- интерфейс для задания распределения генотипа/фенотипа популяции организмов;
- определение родителей по популяции организмов;
- меню выбора организмов из базы;
- меню создания организмов (заведение наборов признаков);
- и т.д.

3.1.5 Работа с базой данных (дополнительная часть)

Модуль для работы с базой данных должен предоставлять хотя бы 2 различные возможности:

- база признаков;
- база организмов;
- база популяций;
- и т.д.

3.2 Композиция музыкальных партий

3.2.1 Описание

В данном задании предлагается реализовать программную библиотеку, представляющую встроенный язык программирования для композиции музыкальных партий.

Основными сущностями этого языка являются:

- нота (C, D, E, F, G, A, B и альтерации: ♯, ♭ и ♮);
- пауза;
- интервал (диатонический и хроматический);
- длительность (целая, половинная, четверная и т.д);
- лад (последовательность нот в рамках одной октавы);
- гармония (аккорды, сопровождающие мелодию).

Библиотека должна предоставлять возможности для создания мелодий, заготовок партий и средства объединения этих базовых кусочков в более сложные партии, партитуры и композиции.

Fig. 3.2: Английская народная песня «Зелёные рукава».

3.2.2 Минимальные требования (базовая часть)

Базовая реализация проекта, в которой должны разбираться все участники, должна:

- предоставлять возможность задания одноголосых мелодий (с паузами);
- транспонирование (перемещение всех нот на заданный интервал);
- последовательное склеивание нескольких мелодий;
- наложение мелодий (создание многоголосной партии).

3.2.3 Расширенная библиотека (дополнительная часть)

Библиотека для композиции музыкальных партий должна быть расширена как минимум двумя различными возможностями:

- поддержка триолей, квартолей и пр.
- поддержка различных размеров;
- сжатие/растяжение мелодии;
- гармония, генерация нот по аккордам, угадывание гармонии;
- поддержка текста песен;
- и т.д.

3.2.4 Генерация композиции (дополнительная часть)

Возможна генерации композиции в любом из следующих форматов:

- LilyPond;
- MusicXML;
- MIDI;
- и др.

3.2.5 Графический интерфейс (дополнительная часть)

Графический интерфейс должен предоставлять визуальное представление партии, а также возможности редактирования (которые могут быть реализованы исключительно «горячими клавишами»).

Графический интерфейс должен реализовывать как минимум ввод отдельных нот/пауз с разными длительностями, а также выделение и удаление нот/пауз. Дополнительно должна быть реализована как минимум одна возможность:

- выбор/смена размера композиции;
- выделение/копирование/вставка/удаление отрезка партии;
- транспонирование выделенного участка на заданный интервал (хроматический/диатонический);
- поддержка триолей, квартолей и пр.

- визуализация и редактирование гармонии;
- визуализация и редактирование знаков артикуляции и динамики;
- визуализация и редактирование текста;
- и т.д.

Головоломки

4.1 Чайнворд

4.1.1 Описание

Чайнворд — это разновидность кроссворда, где слова составляют единую цепочку (без пересечений).

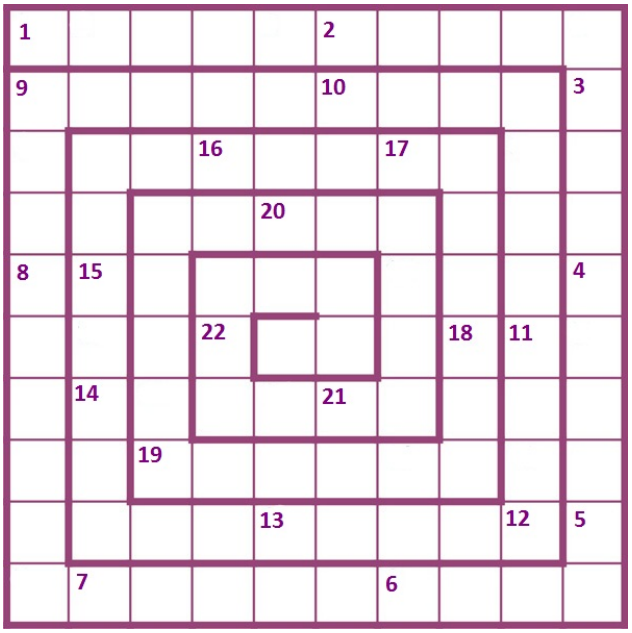

Fig. 4.1: Спиральное расположение.

4.1.2 Минимальные требования (базовая часть)

Базовая реализация проекта, в которой должны разбираться все участники, должна включать:

- интерфейс для разгадывания чайнворда (чайнворд + описания слов);
- загрузку головоломки из файла.

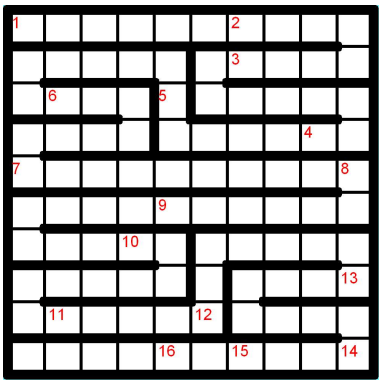

Fig. 4.2: Расположение лабиринтом.

4.1.3 Расширенный интерфейс (дополнительная часть)

Расширенный интерфейс должен добавлять хотя бы 2 различные возможности к базовому интерфейсу. Ниже перечислены возможные варианты расширения интерфейса, однако этим списком они не ограничены:

- показ верно угаданных слов;
- меню генерации чайнворда:
  - словарь;
  - сложность (например, количество слов);
  - вариант расположения (например, спираль, змейкой, лабиринтом, и т.д.)
- подсказки (показать одну букву);
- таймер решения головоломки;
- просмотр таблицы рекордов;
- интерфейс сохранения/загрузки игр;
- альтернативные способы отображения чайнворда (многоугольники, круги, спирали и пр.);
- редактор головоломок;
- и т.д.

4.1.4 Генерация головоломок (дополнительная часть)

Генератор головоломок должен:

- использовать для генерации как минимум два параметра из трёх:
  - словари;
  - схемы расположения цепочки слов;
  - уровни сложности.
- по возможности, предоставлять равномерное распределение вероятности попадания слова из словаря в головоломку.

Выбор из нескольких заранее подготовленных головоломок не считается генерацией.

4.1.5 Работа с базой данных (дополнительная часть)

Модуль для работы с базой данных должен предоставлять хотя бы 2 различных возможности:

- таблица рекордов;
- сохранения игровых сессий;
- база головоломок;
- и т.д.

4.2 Японский кроссворд

4.2.1 Описание

Изображения зашифрованы числами, расположенными слева от строк, а также сверху над столбцами. Числа показывают, сколько групп чёрных (либо своего цвета, для цветных кроссвордов) клеток находятся в соответствующих строке или столбце и сколько слитных клеток содержит каждая из этих групп (например, набор чисел 4, 1, и 3 означает, что в этом ряду есть три группы: первая — из четырёх, вторая — из одной, третья — из трёх чёрных клеток). В чёрно-белом кроссворде группы должны быть разделены, как минимум, одной пустой клеткой, в цветном это правило касается только одноцветных групп, а разноцветные группы могут быть расположены вплотную (пустые клетки могут быть и по краям рядов). Необходимо определить размещение групп клеток.

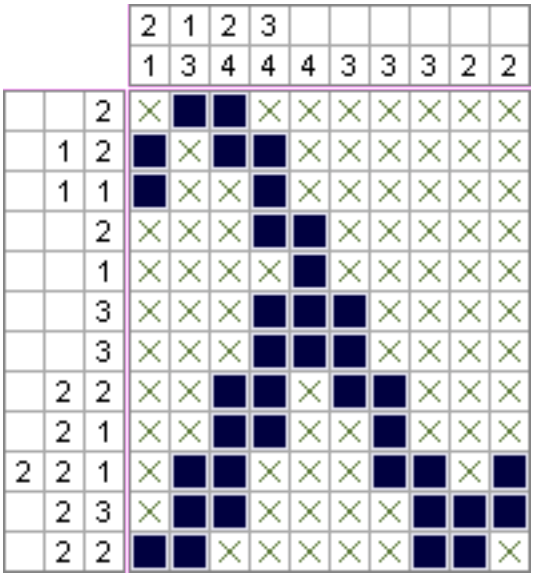

Fig. 4.3: Пример решённой головоломки.

4.2.2 Минимальные требования (базовая часть)

Базовая реализация проекта, в которой должны разбираться все участники, должна включать:

- интерфейс для разгадывания японского кроссворда;
- загрузку головоломки из файла.

4.2.3 Расширенный интерфейс (дополнительная часть)

Расширенный интерфейс должен добавлять хотя бы 2 различные возможности к базовому интерфейсу. Ниже перечислены возможные варианты расширения интерфейса, однако этим списком они не ограничены:

- меню выбора головоломок;
- запуск автоматического решателя;
- таймер решения головоломки;
- просмотр таблицы рекордов и архива решенных головоломок;
- интерфейс сохранения/загрузки частичного решения;
- редактор головоломок;
- и т.д.

4.2.4 Генерация и автоматическое решение головоломок (дополнительная часть)

В этой дополнительной части требуется реализовать генерацию головоломок и автоматический решатель.

Генерация головоломок должна происходить по картинке, заданной в файле (например, псевдографикой в текстовом файле или в любом медиа-формате). После генерации головоломки, её сложность должна быть оценена при помощи автоматического решателя.

Автоматическое решение головоломки может быть реализовано несколькими способами. Чтобы использовать тот же алгоритм для оценки сложности головоломок, необходимо использовать стратегии, моделирующие решение человеком. При решении на каждом шаге пробуются стратегии, начиная с самой простой, и первая подходящая используется. Средняя сложность использованных стратегий может расцениваться как общая сложность головоломки.

Выбор из нескольких заранее подготовленных головоломок не считается генерацией.

4.2.5 Работа с базой данных (дополнительная часть)

Модуль для работы с базой данных должен предоставлять хотя бы 2 различных возможности:

- таблица рекордов;
- сохранения игровых сессий;
- база головоломок;
- и т.д.

4.3 Сокобан

4.3.1 Описание

Сокобан — логическая игра-головоломка, в которой игрок передвигает ящики по лабиринту, показанному в виде плана, с целью поставить все ящики на заданные конечные позиции. Только один ящик

может быть передвинут за раз, причём герой игры — «кладовщик» — может только толкать ящики, но не тянуть их.

Fig. 4.4: Головоломка «Сокобан».

В данном задании допускается реализация любой разновидности головоломки, включая следующие возможные модификации:

- использование нестандартной сетки (например, треугольной или гексагональной);
- использование более одного «кладовщика»;
- добавление игровых элементов (например, телепортов, односторонних проходов, лишних предметов и пр.)
- и т.д.

4.3.2 Минимальные требования (базовая часть)

Базовая реализация проекта, в которой должны разбираться все участники, должна включать:

- интерфейс решения головоломки;
- загрузка головоломки из файла.

4.3.3 Расширенный интерфейс (дополнительная часть)

Расширенный интерфейс должен добавлять хотя бы 2 различные возможности к базовому интерфейсу. Ниже перечислены возможные варианты расширения интерфейса, однако этим списком они не ограничены:

- меню выбора головоломок;
- запуск и визуализация процесса автоматического решателя;
- таймер решения головоломки;
- просмотр таблицы рекордов и архива решенных головоломок;
- интерфейс сохранения/загрузки частичного решения;
- редактор головоломок;
- и т.д.

4.3.4 Генерация и автоматическое решение головоломок (дополнительная часть)

В этой дополнительной части требуется реализовать генерацию головоломок и автоматический решатель.

Генерация головоломки может быть реализована различными способами. Однако, наиболее стабильный способ генерации, пожалуй, использование небольшого числа шаблонов, совмещаемых некоторое количество раз. При хорошем выборе начальных шаблонов вероятность генерации решаемой головоломки может быть достаточно высока.

Идеи о генерации головоломок можно черпать из статьи Automatic Making of Sokoban Problems [PDF].

После генерации головоломки, её сложность должна быть оценена при помощи автоматического решателя.

Автоматическое решение головоломки может также быть реализовано несколькими способами. Для реалистичной оценки сложности головоломок, рекомендуется использовать стратегии, моделирующие решение человеком. При решении на каждом шаге пробуются стратегии, начиная с самой простой, и первая подходящая используется. Средняя сложность использованных стратегий может расцениваться как общая сложность головоломки.

Выбор из нескольких заранее подготовленных головоломок не считается генерацией.

4.3.5 Работа с базой данных (дополнительная часть)

Модуль для работы с базой данных должен предоставлять хотя бы 2 различных возможности:

- таблица рекордов;
- сохранения игровых сессий;
- база головоломок;
- и т.д.

4.4 Судоку

4.4.1 Описание

Игровое поле представляет собой квадрат размером 9×9, разделённый на меньшие квадраты со стороной в 3 клетки. Таким образом, всё игровое поле состоит из 81 клетки. В них уже в начале игры стоят некоторые числа (от 1 до 9), называемые подсказками. От игрока требуется заполнить свободные клетки цифрами от 1 до 9 так, чтобы в каждой строке, в каждом столбце и в каждом малом квадрате 3×3 каждая цифра встречалась бы только один раз.

Сложность судоку зависит не от количества изначально заполненных клеток, а от методов, которые нужно применять для её решения. Самые простые решаются дедуктивно: всегда есть хотя бы одна клетка, куда подходит только одно число. Некоторые головоломки можно решить за несколько минут, на другие можно потратить часы.

Правильно составленная головоломка имеет только одно решение.

4.4.2 Минимальные требования (базовая часть)

Базовая реализация проекта, в которой должны разбираться все участники, должна включать:

- интерфейс для разгадывания судоку;
- загрузку головоломки из файла.

4.4.3 Расширенный интерфейс (дополнительная часть)

Расширенный интерфейс должен добавлять хотя бы 2 различные возможности к базовому интерфейсу. Ниже перечислены возможные варианты расширения интерфейса, однако этим списком они не ограничены:

- настройки сложности судоку;

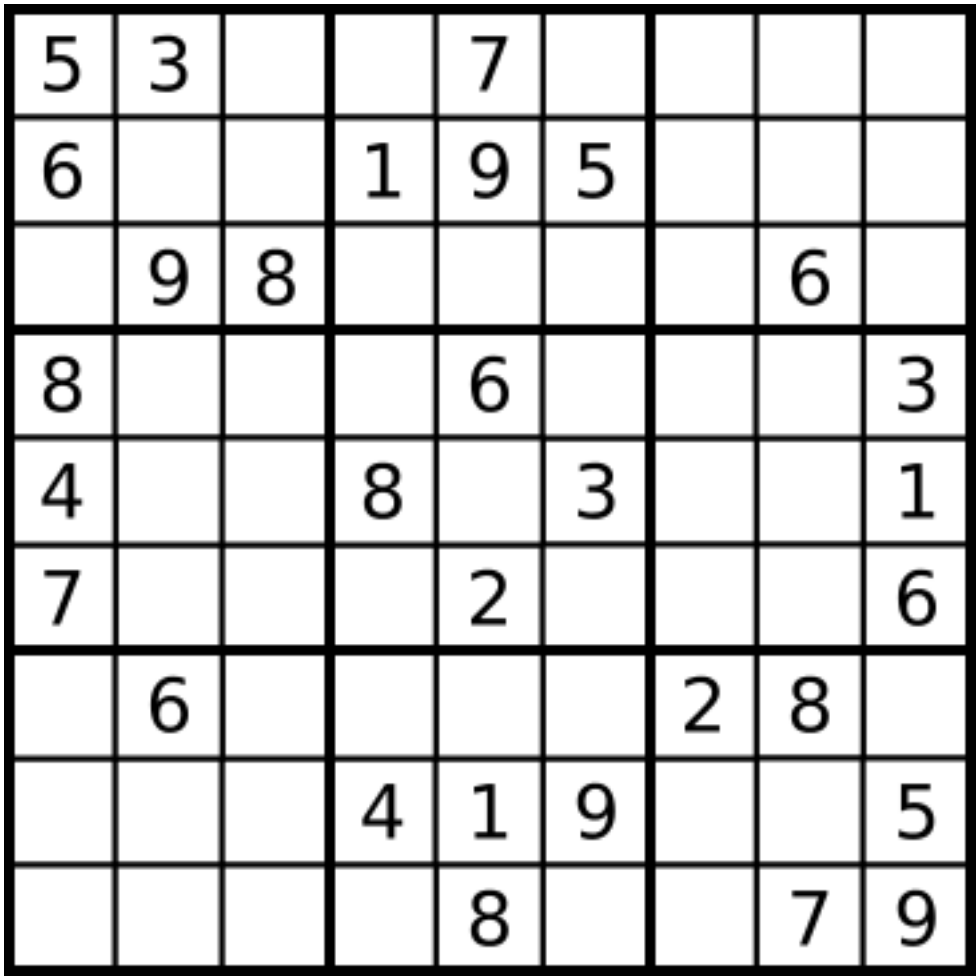

Fig. 4.5: Оригинальная головоломка.

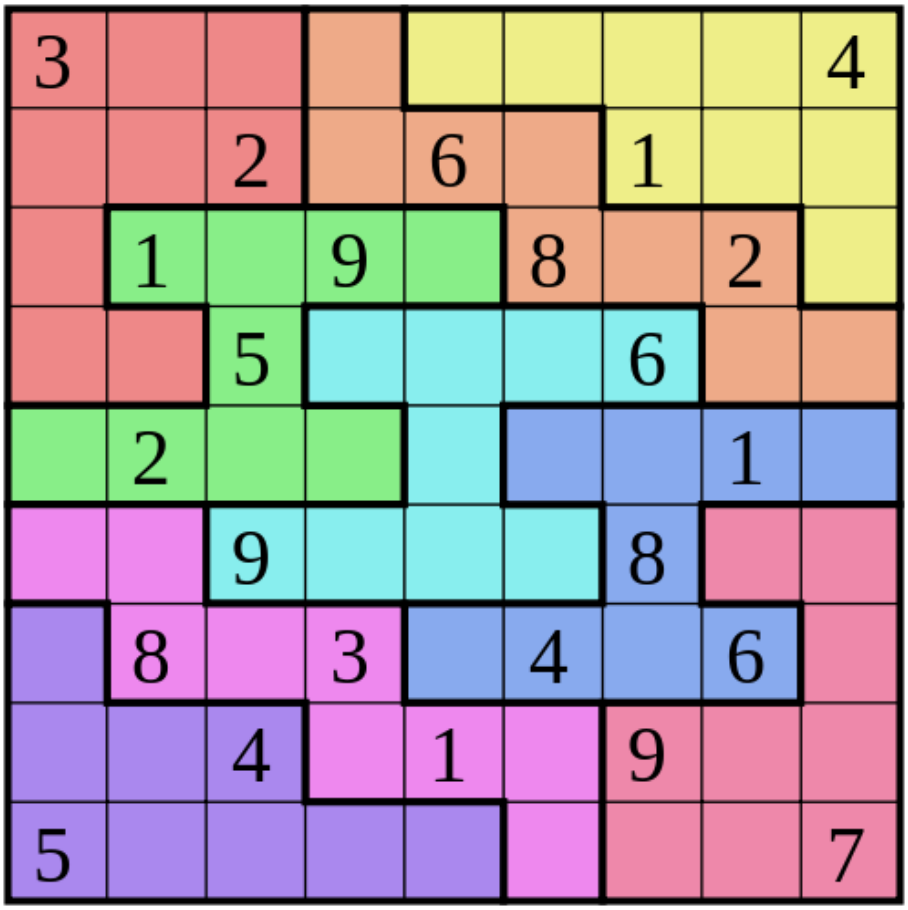

Fig. 4.6: Разновидность Jigsaw Sudoku.

- подсказки (например, показать, какие цифры можно поставить в данную клетку);
- запуск автоматического решателя;
- таймер решения головоломки;
- просмотр таблицы рекордов;
- интерфейс сохранения/загрузки игр;
- выбор разновидности головоломки;
- редактор головоломок;
- и т.д.

4.4.4 Генерация и автоматическое решение головоломок (дополнительная часть)

В этой дополнительной части требуется реализовать генерацию головоломок с заданной сложностью и автоматический решатель.

Простым способом генерации головоломки «Судоку» является случайная перестановка строк и стобцов и вычёркивание случайных клеток. Для генерации головоломок заданной сложности можно воспользоваться одним из двух вариантов:

- вычёркивать клетки по одной, каждый раз оценивая насколько игра усложняется (насколько трудно человеку будет проставить вычеркнутую цифру);
- генерировать головоломку, используя автоматический решатель.

Автоматическое решение головоломки может быть реализовано несколькими способами. Чтобы использовать тот же алгоритм для генерации головоломок заданной сложности, необходимо использовать стратегии, моделирующие решение человеком:

- при решении на каждом шаге пробуются стратегии, начиная с самой простой, и первая подходящая используется.
- при генерации на каждом шаге добавляется очередная цифра в поле, на основе одной из стратегий заданной сложности.

Выбор из нескольких заранее подготовленных головоломок не считается генерацией.

4.4.5 Работа с базой данных (дополнительная часть)

Модуль для работы с базой данных должен предоставлять хотя бы 2 различных возможности:

- таблица рекордов;
- сохранения игровых сессий;
- база головоломок;
- и т.д.

Глава 5

Игры

5.1 Астероиды

5.1.1 Описание

В данном задании предлагается реализовать клон популярной игры «Asteroids».

Цель оригинальной игры состоит в том, чтобы получить как можно больше очков, расстреливая астероиды и летающие тарелки, и избегая при этом столкновения с обломками. Игрок управляет космическим кораблём в форме стрелки, которая может крутиться влево и вправо, а также двигаться и стрелять, но только вперёд. При движении импульс не сохраняется: если не включать двигатель, то корабль постепенно остановится. Игрок также может использовать гиперпространственный двигатель — это приводит к тому, что корабль исчезает и затем появляется в случайном месте экрана, с риском уничтожения из-за появления на месте астероида.

5.1.2 Минимальные требования (базовая часть)

Базовая реализация проекта, в которой должны разбираться все участники, должна:

- предоставлять возможность играть игру «Астероиды» (т.е. управлять кораблём, стрелять и уничтожать астероиды);
- определять момент поражения.

5.1.3 Расширенный интерфейс (дополнительная часть)

Расширенный интерфейс должен добавлять хотя бы 2 различные возможности к базовому интерфейсу. Ниже перечислены возможные варианты расширения интерфейса, однако этим списком они не ограничены:

- меню:
  - просмотр таблицы рекордов;
  - интерфейс сохранения/загрузки игр;
  - выбор режимов игры:
    - однопользовательская/многопользовательская;
    - уровни сложности;

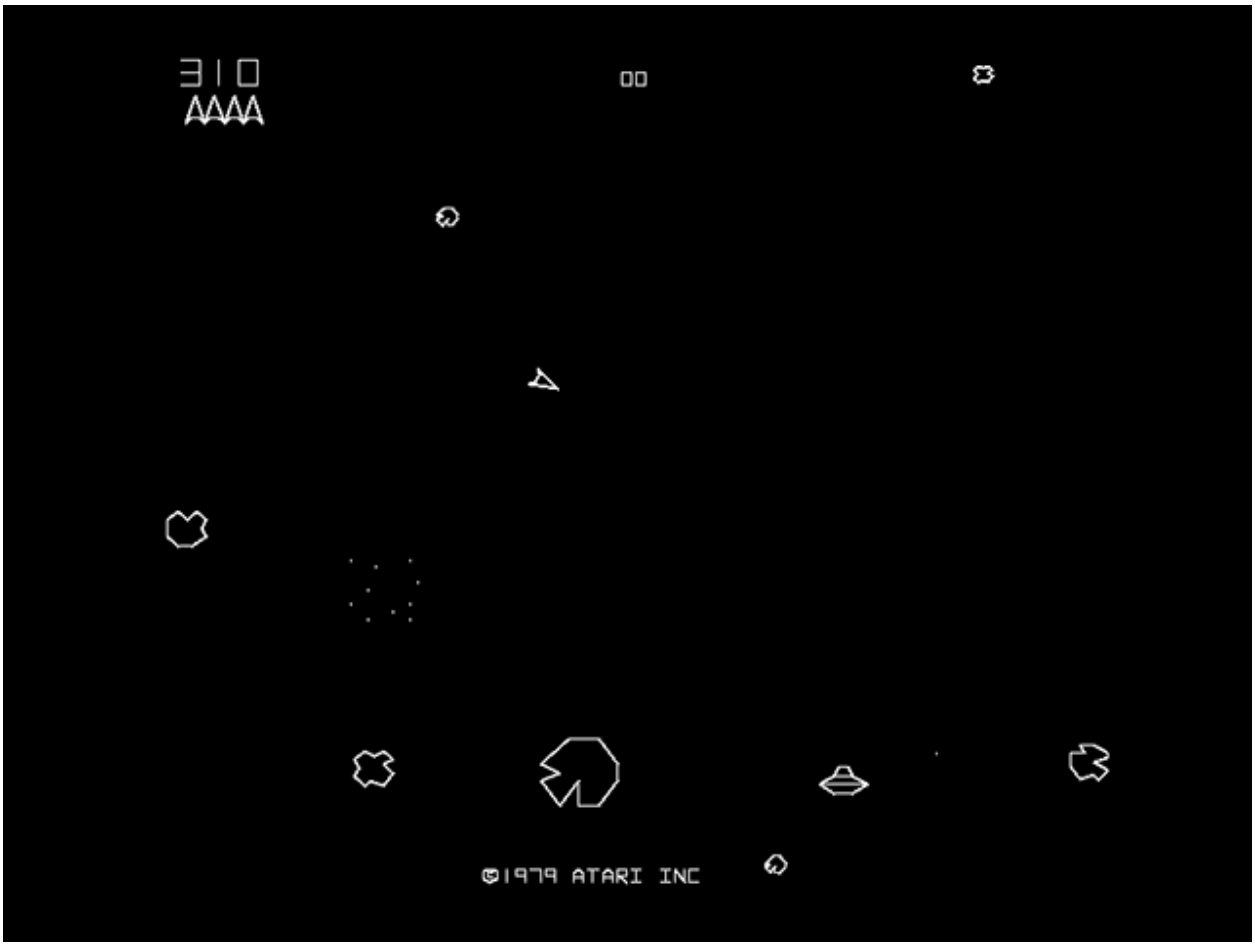

Fig. 5.1: Оригинальная игра фирмы Atari.

Fig. 5.2: Один из клонов.

- расширения игры:
  - бонусы и специальные объекты, которые может подбирать корабль;
  - другие космические объекты (звёзды, чёрные дыры, НЛО и пр.);
  - щиты и маскировка;
  - другие типы орудий;
- трехмерная реализация игры;
- и т.д.

5.1.4 Искусственный интеллект (дополнительная часть)

Алгоритм поведения, никак не оценивающий ситуацию (например, случайное движение), не считается за реализацию искусственного интеллекта.

Искусственный интеллект в игре «Астероиды» управляет полётом НЛО или противника-компьютера (при многопользовательской игре). Реализация искусственного интеллекта должна предоставлять настройки сложности, чтобы сложность поведения НЛО увеличивалась с очередным появлением или повышением уровня.

5.1.5 Клиент-серверная архитектура (дополнительная часть)

Помимо возможности просто играть в игру «Астероиды» по сети (см. Минимальные требования), клиент-серверная архитектура должна предоставлять хотя бы 2 дополнительные возможности:

- поддержка нескольких игровых сессий одновременно;

- запуск ИИ на серверной стороне;
- регистрация, аутентификация и авторизация (вход в систему и права на доступ);
- доступ к таблице рекордов;
- сохранение/загрузка игр;
- и т.д.

5.1.6 Работа с базой данных (дополнительная часть)

Модуль для работы с базой данных должен предоставлять хотя бы 2 различных возможности:

- таблица рекордов;
- сохранения игровых сессий;
- база пользователей;
- и т.д.

5.2 Настольные игры

5.2.1 Описание

В данном задании предлагается реализовать любую из настольных игр (например, реверси, шашки, шахматы, нарды, го и пр.).

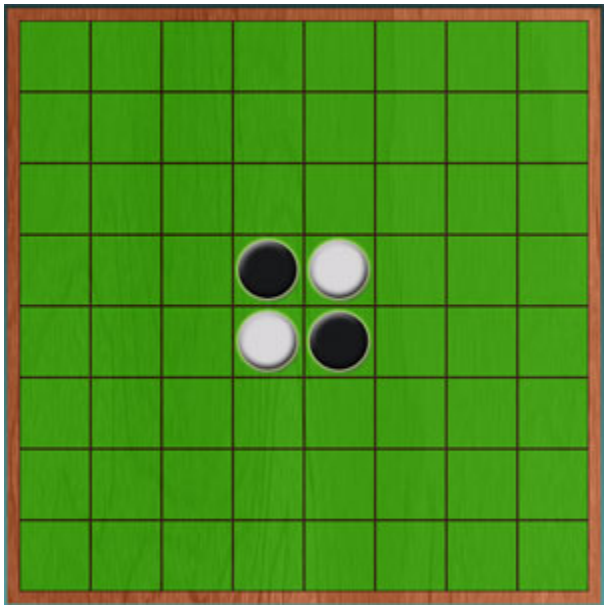

Fig. 5.3: Реверси. Начальное расположение шашек на поле 8×8.

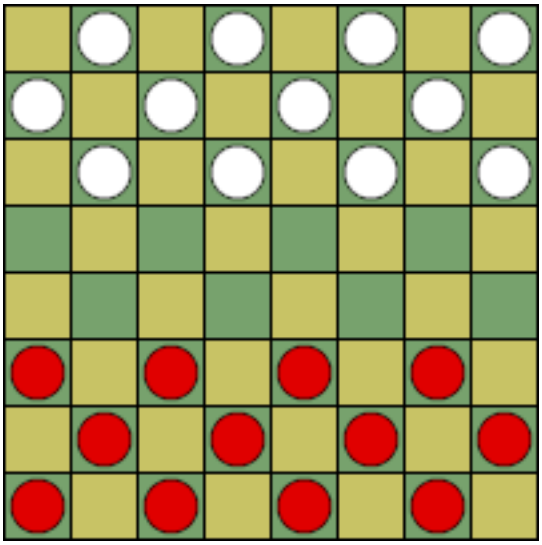

Fig. 5.4: Шашки. Начальное расположение.

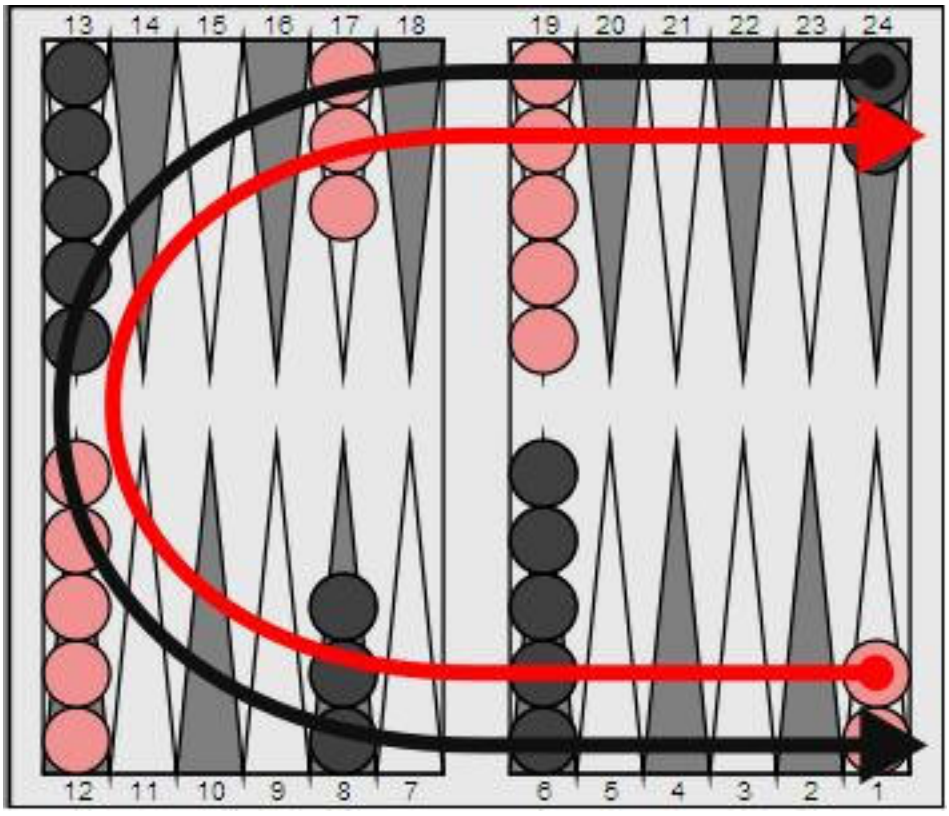

Fig. 5.5: Нарды. Начальное расположение и направление движения шашек.

5.2.2 Минимальные требования (базовая часть)

Базовая реализация проекта, в которой должны разбираться все участники, должна:

- предоставлять возможность играть в реверси в режиме «человек против человека», используя графический интерфейс;
- не допускать невозможных по правилам игры ходов игроков;
- определять момент победы или ничьей и демонстрировать пользователям результат.

5.2.3 Расширенный интерфейс (дополнительная часть)

Расширенный интерфейс должен добавлять хотя бы 2 различные возможности к базовому интерфейсу. Ниже перечислены возможные варианты расширения интерфейса, однако этим списком они не ограничены:

- показ возможных ходов для выбранной фигуры/шашки;
- отмена хода/ходов;
- меню выбора режима игры:
  - человек против человека;
  - человек против компьютера;
  - компьютер против компьютера;
  - сетевая игра;
- меню настроек ИИ;
- меню выбора разновидности настольной игры (например, шашки с дамками или без);
- просмотр таблицы рекордов;
- проигрывание записанных игр;
- интерфейс сохранения/загрузки игр;
- и т.д.

5.2.4 Искусственный интеллект (дополнительная часть)

Алгоритм, делающий ход, никак не оценивая ситуацию (например, случайный или первый доступный ход), не считается за реализацию искусственного интеллекта.

Реализация искусственного интеллекта должна предоставлять:

- настройки сложности;
- как минимум 2 различных стратегии (это может быть один алгоритм с разными эвристиками).

5.2.5 Клиент-серверная архитектура (дополнительная часть)

Помимо возможности просто играть в настольную игру по сети (см. Минимальные требования), клиент-серверная архитектура должна предоставлять хотя бы 2 дополнительные возможности:

- поддержка нескольких игровых сессий одновременно;
- запуск ИИ на серверной стороне в качестве противника;

- регистрация, аутентификация и авторизация (вход в систему и права на доступ);
- доступ к таблице рекордов;
- сохранение/загрузка игр;
- и т.д.

5.2.6 Работа с базой данных (дополнительная часть)

Модуль для работы с базой данных должен предоставлять хотя бы 2 различных возможности:

- таблица рекордов;
- сохранения игровых сессий;
- база пользователей;
- и т.д.

5.3 Бесконечный платформер

5.3.1 Описание

Платформер — это жанр компьютерных игр, в которых основной чертой игрового процесса является прыгание по платформам, лазанье по лестницам, собирание предметов, обычно необходимых для завершения уровня.

Бесконечны платформер — это платформер, в котором персонаж бежит в одном направлении по миру, который генерируется по ходу движения. Таким образом, игровой мир потенциально бесконечен.

Основная цель игры — добежать как можно дальше.

В данном задании предлагается любую разновидность бесконечного платформера.

5.3.2 Минимальные требования (базовая часть)

Базовая реализация проекта, в которой должны разбираться все участники, должна:

- предоставлять простейшую генерацию игрового мира — повторение заданного шаблона;
- создавать игровой мир с возможность проиграть или двигаться дальше;
- предоставлять минимальное управление персонажем;
- определять момент поражения и подсчитывать пройденную дистанцию.

5.3.3 Расширенный интерфейс (дополнительная часть)

Расширенный интерфейс должен добавлять хотя бы 2 различные возможности к базовому интерфейсу. Ниже перечислены возможные варианты расширения интерфейса, однако этим списком они не ограничены:

- меню:
  - просмотр таблицы рекордов;
  - интерфейс сохранения/загрузки игр;

Fig. 5.6: Worm Run 2.

Fig. 5.7: Tiny Wings.

  - выбор режимов игры:
    * однопользовательская/многопользовательская;
    * уровни сложности;
- расширения игры:
  - многопользовательский режим;
  - бонусы и специальные объекты, которые может подбирать персонаж;
  - различные препятствия;
  - отмотка времени;
- и т.д.

5.3.4 Генератор игрового мира (дополнительная часть)

Генератор игрового мира должен постепенно повышать сложность трассы.

Для реализации различных режимов игры, генератор должен быть конфигурируемым. В частности, пользователь генератора должен иметь возможность

- ограничить объекты игрового мира, используемые для генерации;
- определить относительную сложность трассы, на которой может появляться заданный объект;
- определить частоту встречаемости объекта на трассе (в зависимости от сложности);
- и т.д.

5.3.5 Искусственный интеллект (дополнительная часть)

Алгоритм поведения, никак не оценивающий ситуацию (например, случайное движение), не считается за реализацию искусственного интеллекта.

Искусственный интеллект в бесконечном платформере может управлять - активными противниками персонажа, встречающиеся на пути; - соперничающим персонажем (в многопользовательском режиме вместо второго игрока).

Реализация искусственного интеллекта должна предоставлять настройки сложности.

5.3.6 Клиент-серверная архитектура (дополнительная часть)

Помимо возможности просто играть в игру по сети (см. Минимальные требования), клиент-серверная архитектура должна предоставлять хотя бы 2 дополнительные возможности:

- поддержка нескольких игровых сессий одновременно;
- запуск ИИ на серверной стороне;
- регистрация, аутентификация и авторизация (вход в систему и права на доступ);
- доступ к таблице рекордов;
- сохранение/загрузка игр;
- и т.д.

5.3.7 Работа с базой данных (дополнительная часть)

Модуль для работы с базой данных должен предоставлять хотя бы 2 различных возможности:

- таблица рекордов;
- сохранения игровых сессий;
- база пользователей;
- и т.д.

Глава 6

Моделирование

6.1 Жидкость

6.1.1 Описание

Гидродинамика сглаженных частиц (ГСЧ) — это один из распространенных методов моделирования жидкостей и газов. Он отличается простотой программирования и понимания процесса в отличие от моделирования на сетке. Этот метод был впервые предложен для моделирования астрофизических явлений, но сейчас используется также для моделирования жидкостей, газов и процессов деформации в твердых телах.

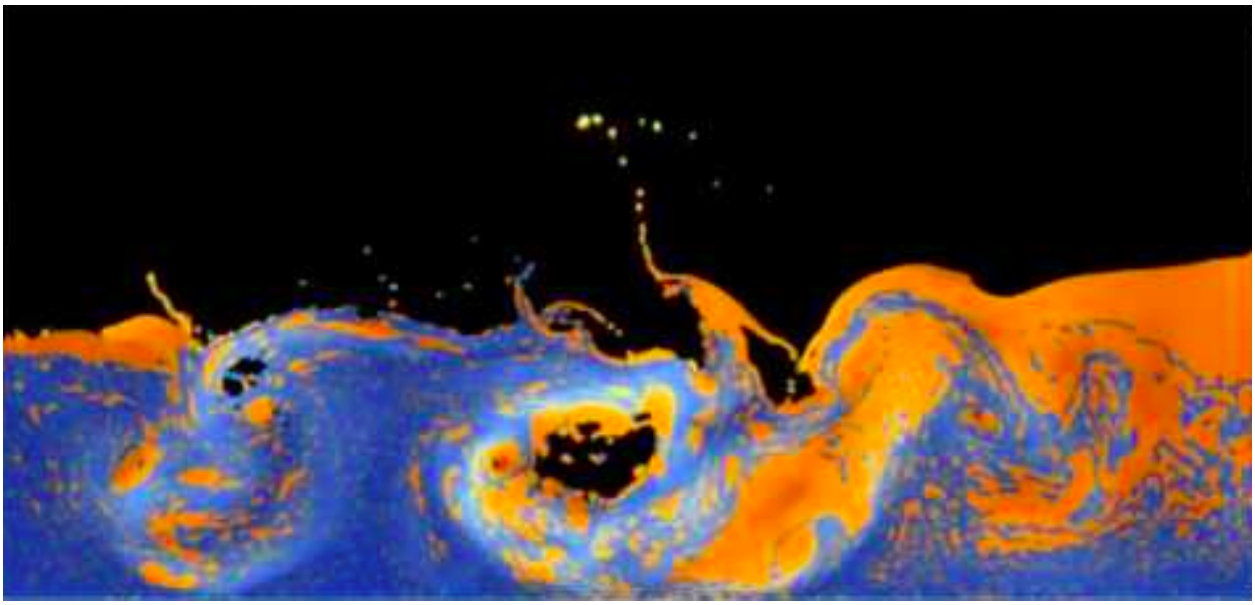

Fig. 6.1: Двухфазное моделирование жидкости.

При моделировании ГСЧ жидкость представляется набором частиц, каждая из которых обладает набором физических характеристик, как-то: положение $\mathbf{r}_i$, скорость $\mathbf{v}_i$, масса m_i. По этим значениям для частиц восстанавливается значение физических величин во всех точках простанства. Для частиц определяется длина сглаживания h, на расстоянии которой свойства частиц «сглаживаются».

Вклад каждой частицы в значение физической величины в точке $\mathbf{r}$ определяется так называемой функцей ядра $W(\mathbf{r} - \mathbf{r}_i, h)$. Чаще всего в качестве функции ядра используются Гауссова функция и

полиномиальные сплайны. Значения последних равны нулю для частиц, расположенных дальше, чем на h от точки $\mathbf{r}$. Это позволяет эффективнее расчитывать модель, игнорируя ничтожно малый вклад далёких частиц.

Любая физическая величина в точке $\mathbf{r}$ может быть рассчитана по следующей формуле: $\mathrm{A(r)} = \sum_j m_j \frac{A_j}{\rho_j} W(\mathbf{r}-\mathbf{r}_j, h)$, A_j — значение величины в точке $\mathbf{r}_j$, ρ_j — плотность частиц в точке $\mathbf{r}_j$. Например, плотность может быть вычислена следующим образом: $\rho_{\mathbf{r}} = \rho(\mathbf{r}) = \sum_j m_j W(\mathbf{r}-\mathbf{r}_j, h).,.$

Давление для частицы расчитывается исходя из её плотности: $\mathrm{p}_i = k(\rho_i - \rho_0), \rho_0$ — это плотность окружающей среды, а k — коэффициент жесткости (сжимаемости). Этот параметр необязателен, но позволяет точнее настроить систему. Сила давления расчитывается по следующей формуле: $\mathrm{f}_{pressure}(\mathbf{r}) = \sum_j m_j \frac{p_i+p_j}{2\rho_j} \nabla W(\mathbf{r}-\mathbf{r}_j, h)^-,. : f_{viscosity}(\mathbf{r}) = \mu \sum_j m_j \frac{\mathbf{v_i}-\mathbf{v_j}}{\rho_j} \nabla^2 W(\mathbf{r}-\mathbf{r}_j, h), \mu$ — коэффициент вязкости.

Сила поверхностного натяжения расчитывается следующим образом: $\mathrm{f}_{tension}(\mathbf{r}) = \sigma \sum_j m_j \frac{1}{\rho_j} \nabla^2 W(\mathbf{r}-\mathbf{r}_j, h), \sigma$ — коэффициент поверхностного натяжения.

Для различных величин могут использоваться различные функции ядра, наиболее точно описывающие вклад соседних частиц. Для большинства величин достаточно хорошо подходит функция W_{poly6}:

$$W_{poly6}(\mathbf{r}, h) = \frac{315}{64\pi h^9} \begin{cases} (h^2 - r^2)^3, & 0 \le r \le h \\ 0, & r > h \end{cases}$$

$$\nabla W_{poly6}(\mathbf{r}, h) = -\frac{315}{64\pi h^9} \begin{cases} 6r(h^2 - r^2)^2, & 0 \le r \le h \\ 0, & r > h \end{cases}$$

$$\nabla^2 W_{poly6}(\mathbf{r}, h) = \frac{315}{64\pi h^9} \begin{cases} 6(h^2 - r^2)(4r^2 - (h^2 - r^2)), & 0 \le r \le h \\ 0, & r > h \end{cases}$$

Тем не менее, для лучшего эффекта несжимаемой жидкости при расчёте плотности лучше воспользоваться специальной функцией ядра: $\mathrm{W}_{density}(\mathbf{r}, h) = \begin{cases} (1 - \frac{r}{h})^2, & 0 \le r \le h \\ 0, & r > h \end{cases}, . :$

нахождение соседних частиц;

вычисление плотности;

вычисление давления;

расчёт сил: давление, вязкость и поверхностное натяжение;

добавление гравитации и расчёт ускорения;

пересчет скорости и положения;

проверку ограничений (стенки);

отрисовку.

6.1.2 Минимальные требования (базовая часть)

Базовая реализация проекта, в которой должны разбираться все участники, должна содержать:

- визуализацию и моделирование заданной сцены с жидкостью;
- загрузку начального состояния сцены из файла.

6.1.3 Расширенный интерфейс (дополнительная часть)

Расширенный интерфейс должен добавлять хотя бы 2 различные возможности к базовому интерфейсу. Ниже перечислены возможные варианты расширения интерфейса, однако этим списком они не ограничены:

- меню выбора сцены;
- меню выбора способа моделирования (различные ядровые функции);
- редактор сцены;
- настройки свойств жидкости;
- управление моделированием:
  - пауза/продолжение;
  - ускорение/замедление;
  - перемотка;
- интерфейс сохранения/загрузки текущего состояния сцены;
- отображение поверхности жидкости;
- и т.д.

6.1.4 Расширенные возможности моделирования (дополнительная часть)

Расширенное моделирование должно добавлять хотя бы 2 различные возможности к базовому моделированию жидкости:

- оптимизация расчёта:
  - быстрое нахождение соседних частиц;
  - автоматически подстраиваемая под каждую частицу длина сглаживания (чтобы в её окрестности всегда находилось примерно одно и то же кол-во частиц);
- многофазные жидкости (разнородные частицы, вязкость и поверхностное натяжение действует только между частицами одного типа);
- взаимодействие с движущимися объектами (мяч, пропеллер и т.п.);
- и т.д.

6.1.5 Работа с базой данных (дополнительная часть)

Модуль для работы с базой данных должен предоставлять хотя бы 2 различных возможности:

- база сцен;
- база жидкостей и других объектов сцены;
- и т.д.

6.2 Игра «Жизнь»

6.2.1 Описание

Игра «Жизнь» — это клеточный автомат, придуманный английским математиком Джоном Конвеем в 1970 году.

Игра происходит во вселенной — плоскости или поверхности, разбитой на клетки. Каждая клетка в каждый момент времени может находиться в одном из двух состояний: быть «живой» или «мёртвой» (пустой). У каждой клетки есть соседи, определяемые окрестностью Мура.

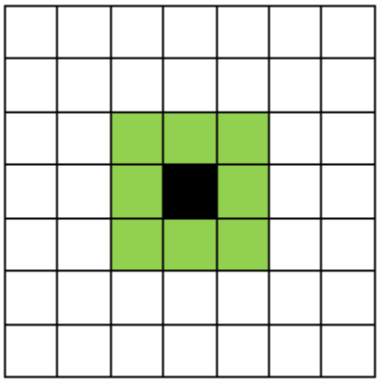

Fig. 6.2: Окрестность Мура.

Время в игре разбито на шаги, за каждый шаг все клетки одновременно обновляют своё состояние по следующим правилам:

- в пустой (мёртвой) клетке зарождается жизнь, если рядом находится ровно 3 живые клетки;
- если у живой клетки ровно 2 или 3 живых соседа, она остаётся жить; иначе она умирает (от одиночества или перенаселённости).

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

В игре жизнь присутсвует множество интересных структур. Самым популярным, вероятно, является планер.

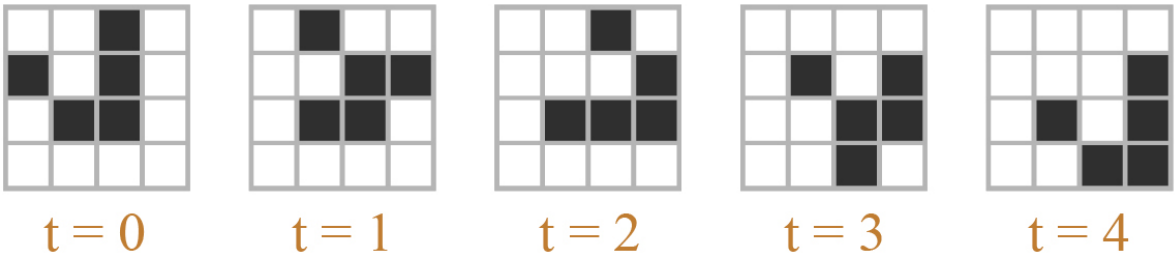

Fig. 6.3: Планер и его движение, определяемое правилами игры.

6.2.2 Минимальные требования (базовая часть)

Базовая реализация проекта, в которой должны разбираться все участники, должна содержать:

- визуализацию и моделирование заданной сцены;
- загрузку начального состояния сцены из файла.

6.2.3 Расширенный интерфейс (дополнительная часть)

Расширенный интерфейс должен добавлять хотя бы 2 различные возможности к базовому интерфейсу. Ниже перечислены возможные варианты расширения интерфейса, однако этим списком они не ограничены:

- меню выбора сцены;
- меню выбора модификаций игры «Жизнь»;
- настройки поля-вселенной;
- редактор сцены с панелью заготовленных объектов;
- управление моделированием:
  - пауза/продолжение;
  - ускорение/замедление;
  - перемотка;
- интерфейс сохранения/загрузки;
- и т.д.

6.2.4 Расширенные возможности моделирования (дополнительная часть)

Расширенное моделирование должно добавлять хотя бы 2 различные возможности к базовому моделированию:

- оптимизация расчёта: быстрое нахождение соседних частиц и расчёт только окрестностей живых клеток;
- модификации игры жизнь:
  - полноцветная игра;
  - шестиугольная сетка;
  - мозаика Пенроуза;
- и т.д.

6.2.5 Работа с базой данных (дополнительная часть)

Модуль для работы с базой данных должен предоставлять хотя бы 2 различных возможности:

- база сцен;
- база объектов;
- и т.д.

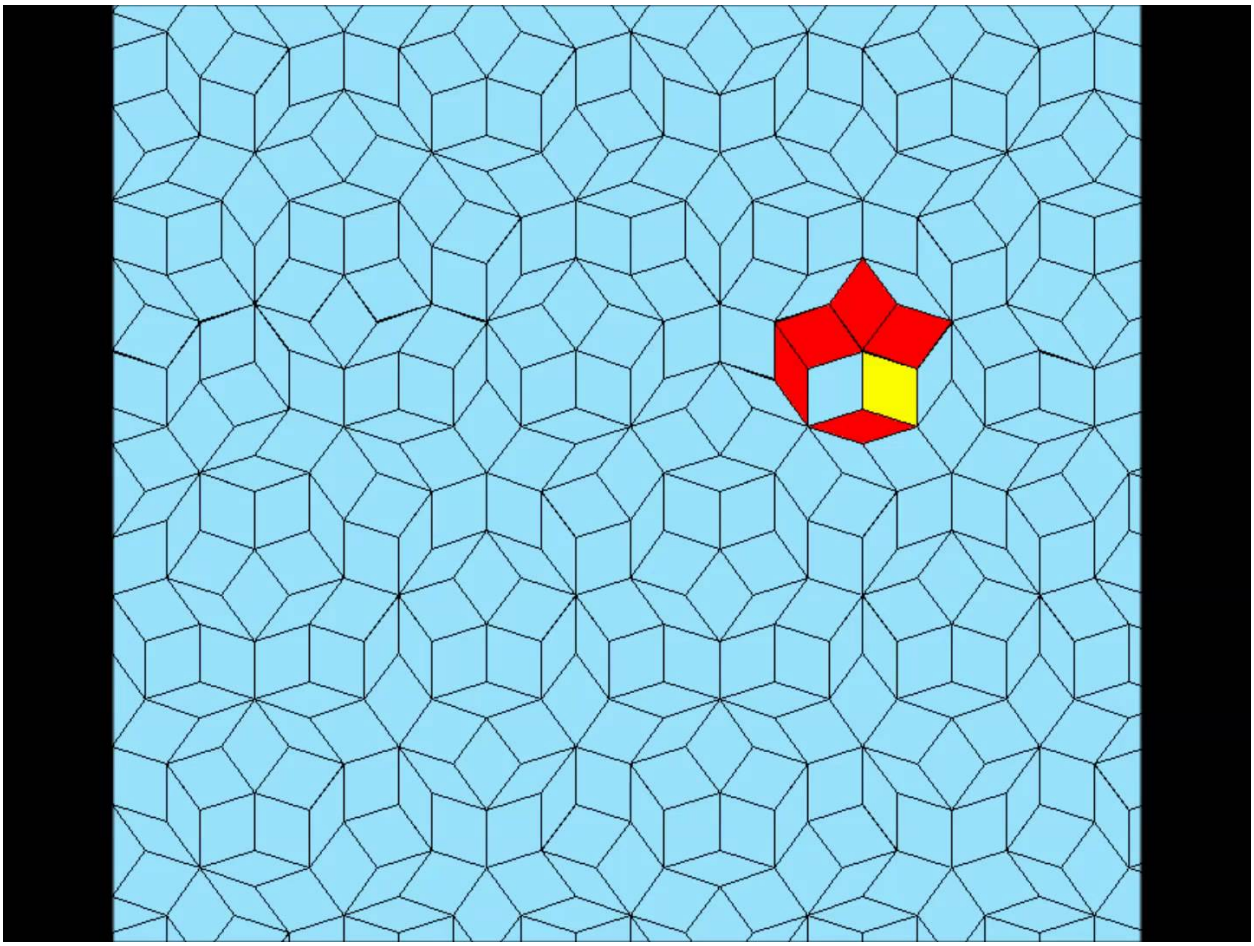

Fig. 6.4: Планер на мозаике Пенроуза.