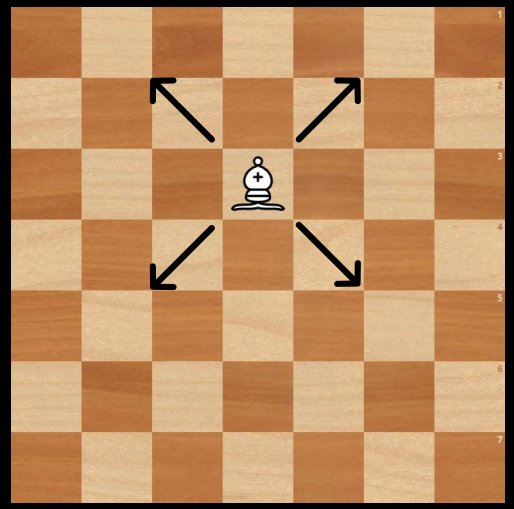
Отчет решения задачи от эксперта
Тема: DevOps

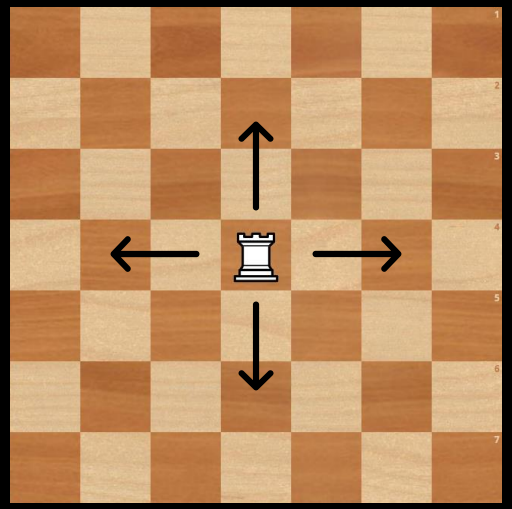
Выполнил: Ярошенко Никита Алексеевич

+7 (968) 468-20-66

Постановка задачи



Слон бьет по диагоналям.



Ладья бьет параллельно координатам.

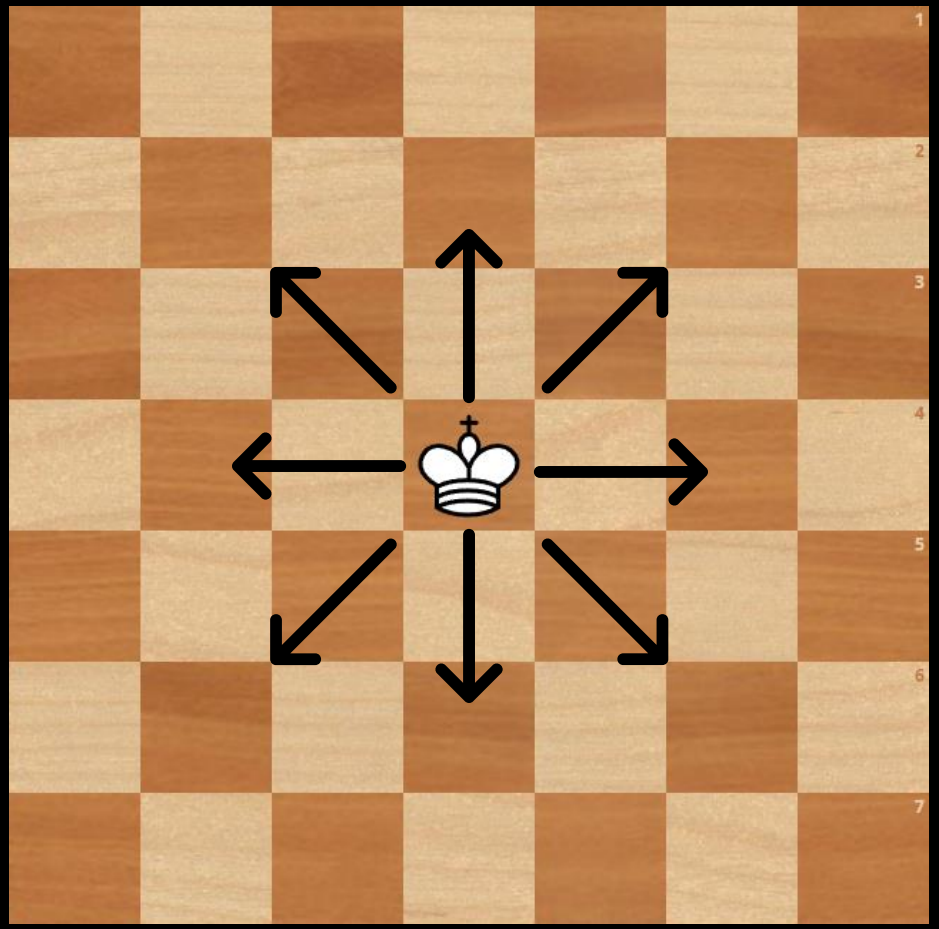
На шахматной доске стоят:

- Белый король,
- черный слон,
- черная ладья.

Определить, от какой фигуры есть угроза королю.
Варианты ответа:

- Шах от слона
- Шах от ладьи
- Нет шаха

Вербальная модель решения



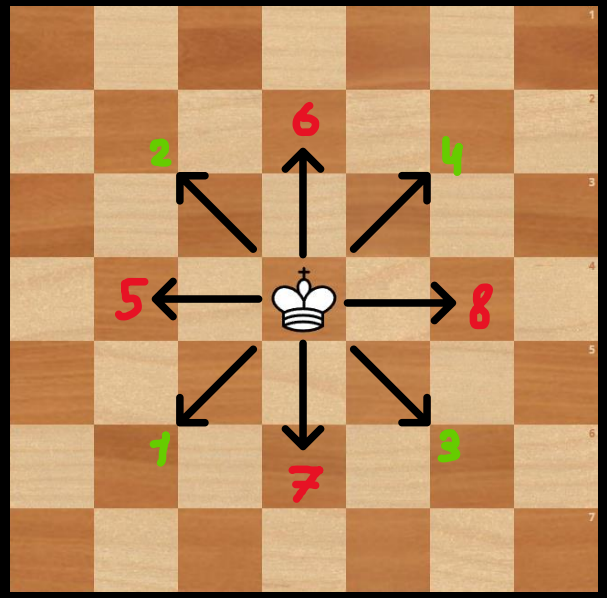
На шахматном поле нужно найти местоположение белого короля.

После этого от координат короля просматривать разные стороны по диагоналям, горизонталям и вертикалям.

Просмотр координат будет до первой найденной фигуры или конца шахматной доски.

Как только будет найдена фигура, будет сравнение с фигурами которые могут атаковать в этом направлении.

Математическая модель решения



Тем самым на диагоналях и горизонталях двигаемся с инкрементами D*(x, y),H*(x, y):
первые итерации D(-1, -1), H(-1, 0);
вторые итерации D(-1, 1), H(0, 1);
третьи итерации D(1, -1), H(0, -1);
четвертые итерации D(1, 1), H(1, 0).

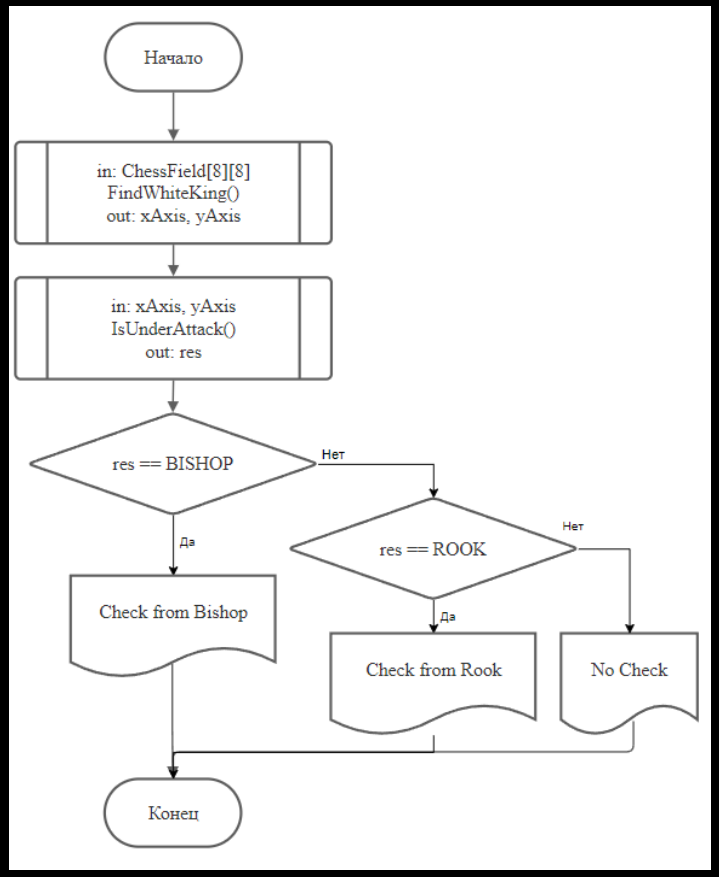
D - диагональ,
H - горизонталь и вертикаль

Поиск будет разделен на диагонали и горизонтали с вертикалями.
Поиск состоит из цикла с четырьмя итерациями, цикл состоит из вложенного цикла который складывает исходные координаты с заданными инкриментами до тех пор пока не встретит на своем пути фигуру или конец доски. Если фигура того же цвета что и король, то внутренний цикл заканчивается и наступает следующая итерация внешнего цикла. Если чужого, то сверяется с фигурами которые могут вести атаку с этой прямой.

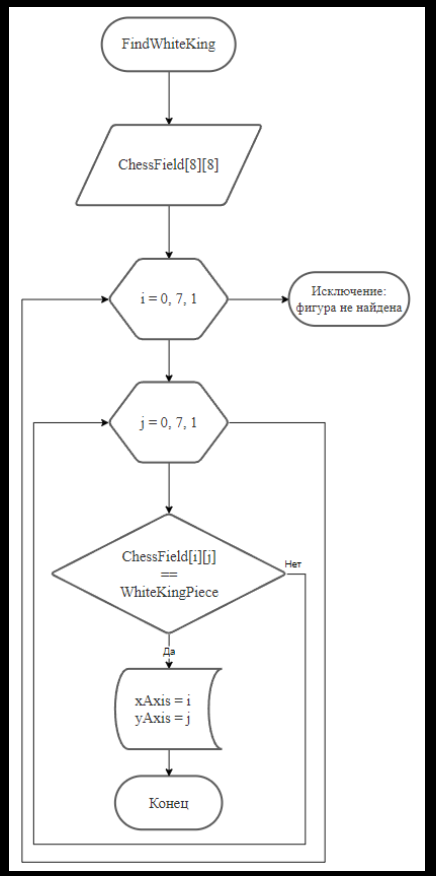
1. Для диагоналей, первые инкременты будут для X = -1, для Y = -1. После первого и третьего цикла инкремент Y увеличивается на 2, после второго цикла инкременты изменятся на X = 1, Y = -1.
2. Для горизонталей и вертикалей, первые инкременты будут X = -1, Y = 0. После первой и третьей итерации инкременты увеличатся на единицу, после второго цикла инкремент X = 0, Y = -1.

Блок-схема алгоритма

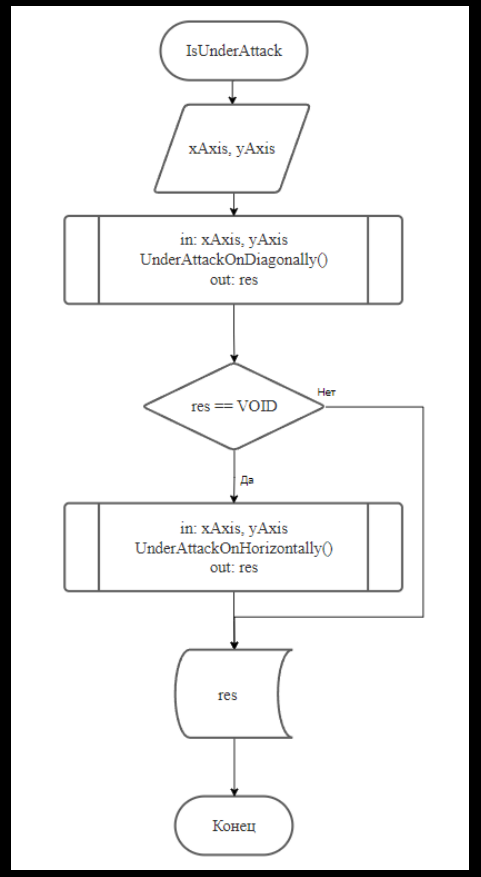
Основное приложение



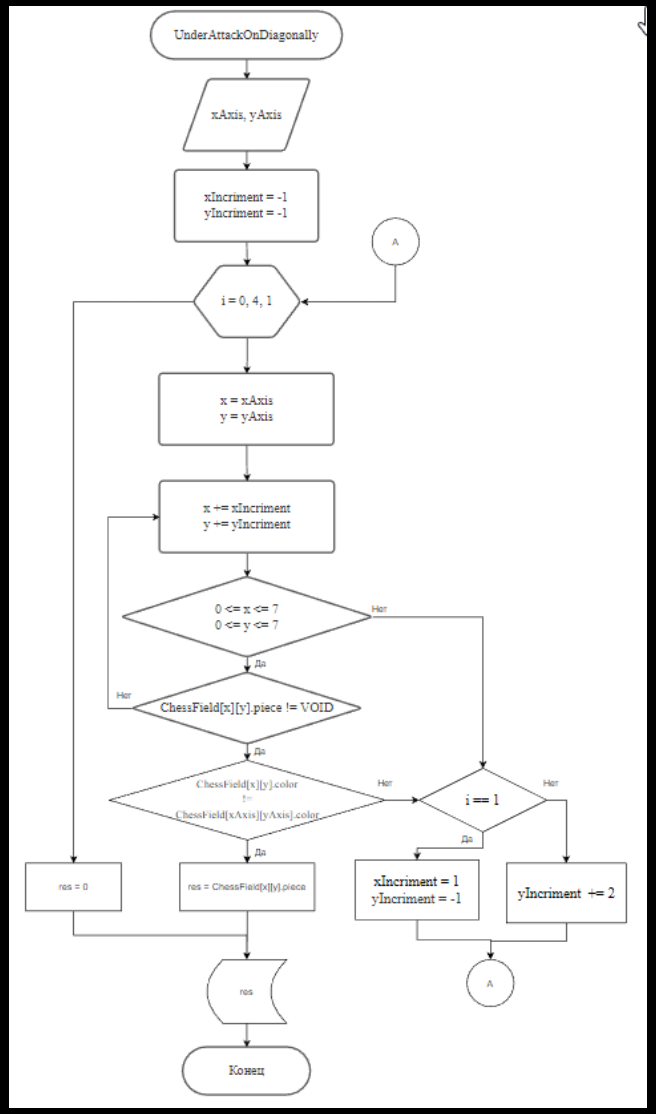
Подпрограмма поиска короля



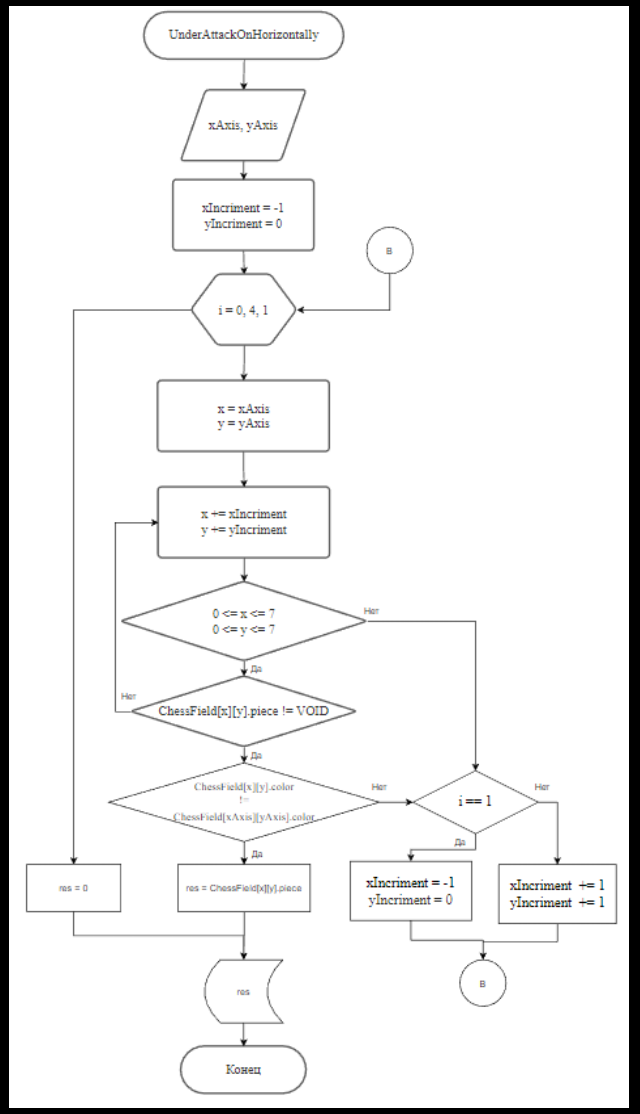
Подпрограмма поиска нападения на заданные координаты



Подпрограмма поиска угрозы по диагоналям от заданных координат



Подпрограмма поиска угрозы по горизонталям и вертикалям от заданных координат



Подробнее блок-схему алгоритма можно посмотреть по пути materials/report/BlockDiagramOfTheAlgorithm.png

Программа на языке среднего уровня C++

Во время разработки был создан класс Chess который содержит в себе математическую составляющую и функцию решения задачи.

Основная функция решения задачи:

```
Chess::SolutionCheck Chess::Solution() {
  std::pair<uint8_t, uint8_t> PositionKing = FindWhiteKing();
  int res = IsUnderAttack(PositionKing.first, PositionKing.second);
  if (res == Piece::PieceFigure::BISHOP) {
    return CheckFromBishop;
  } else if (res == Piece::PieceFigure::ROOK) {
    return CheckFromRook;
  }
  return NoCheck;
}
```

Функция нахождения белого короля:

```
std::pair<uint8_t, uint8_t> Chess::FindWhiteKing() {
  for (int xAxis{}; xAxis < 8; xAxis++) {
    for (int yAxis{}; yAxis < 8; yAxis++) {
      Piece &current = ChessField[xAxis][yAxis];
      if (current.GetPiece() == Piece::PieceFigure::KING &&
          current.GetColor() == Piece::PieceColor::WHITE) {
        return {xAxis, yAxis};
      }
    }
  }
  throw std::logic_error("No white king on the field");
}
```

Функция по нахождению угрозы от заданных координат:

```
int Chess::IsUnderAttack(uint8_t positionNumber, uint8_t
positionMarker) {
  Piece &target = GetPiece(positionNumber, positionMarker);
  int res = UnderAttackOnDiagonally(target.GetColor(),
positionNumber, positionMarker);
  if (res) return res;
  return UnderAttackOnHorizontally(target.GetColor(),
positionNumber, positionMarker);
}
```

Функция получения угрозы по диагоналям:

```
int Chess::UnderAttackOnDiagonally(Piece::PieceColor ColorPiece, int xAxisPos, int yAxisPos) {
  int IncrementX = -1, IncrementY = -1;
  for (int i{}; i < 4; i++) {
    unsigned int xAxis = xAxisPos, yAxis = yAxisPos;
    while (true) {
      xAxis += IncrementX;
      yAxis += IncrementY;
      if (xAxis > 7 || yAxis > 7) break;
      Piece &current(ChessField[xAxis][yAxis]);
      if (!*current) continue;
      if (current.GetPiece() == Piece::BISHOP &&
          current.GetColor() != ColorPiece) {
        return Piece::BISHOP;
      }
      break;
    }
    if (i == 1) {
      IncrementY = -1;
      IncrementX = 1;
    } else {
      IncrementY += 2;
    }
  }
  return Piece::VOID;
}
```

Функция получения угрозы по диагоналям:

```
int Chess::UnderAttackOnHorizontally(Piece::PieceColor ColorPiece, int xAxisPos, int yAxisPos) {
  int IncrementX = -1, IncrementY{};
  for (int i{}; i < 4; i++) {
    unsigned int xAxis = xAxisPos, yAxis = yAxisPos;
    while (true) {
      xAxis += IncrementX;
      yAxis += IncrementY;
      if (xAxis > 7 || yAxis > 7) break;
      Piece &current(ChessField[xAxis][yAxis]);
      if (!*current) continue;
      if (current.GetPiece() == Piece::ROOK &&
          current.GetColor() != ColorPiece) {
        return Piece::ROOK;
      }
      break;
    }
    if (i == 1) {
      IncrementY = -1;
      IncrementX = 0;
    } else {
      ++IncrementX;
      ++IncrementY;
    }
  }
  return 0;
}
```

Проверка решения (система тестирования)

Для тестирования были написаны тесты на библиотеке Google Test.

Для тестирования внутренних функций было написано 21 тест кейс для комплексного функционального тестирования был сделан 1 тест кейс с плавающим количеством тестов(на момент сдачи 20 штук в БД).

Пример одного тест кейса:

```
TEST(Chess, under_attack_on_diagonally_x_plus_y_minus) {
  Piece void_none,
      king_white(Piece::PieceFigure::KING, Piece::PieceColor::WHITE),
      bishop_black(Piece::PieceFigure::BISHOP, Piece::PieceColor::BLACK);
  Chess clear_field(void_none);
  uint8_t positionKingX = 4, positionKingY = 4;
  clear_field.GetPiece(positionKingX, positionKingY) = king_white;
  clear_field.GetPiece(5, 3) = bishop_black;
  EXPECT_EQ(clear_field.IsUnderAttack(positionKingX, positionKingY),
            Piece::PieceFigure::BISHOP);
}
```

Если идти с самого начала `Chess` означает к чему принадлежит тест кейс, `under_attack_on_diagonally_x_plus_y_minus` означает что именно тест кейс тестирует. Из названия можно понять что в этом тесте по диагонали идет нападение на фигуру. На координатах (4, 4)(в шахматной форме E5) ставится белый король, а на координатах (5, 3)(в шахматной форме D6) ставится черный слон. Далее в функцию `EXPECT_EQ` из библиотеки GTest кладутся 2 аргумента которые сравниваются. Соответственно в первый аргумент вызываем функцию проверки шаха, а во второй аргумент задаем ожидаемое значение, то есть слона.

В двадцать втором тест кейсе были написаны тесты через запрос в базу данных PostgreSQL для получения фигур шахматном столе и ожидаемое значение.

На примере одних тестовых значений из базы данных:

```
 9 |      24 |       17 |           1 | F           | 4
 9 |      24 |        5 |           2 | D           | 4
 9 |      24 |        7 |           2 | G           | 5
```

В девятом тесте в первой строке на координате F4 находится Король(17) белого цвета(1). На второй строке находится черная(2) ладья(5) на координатах D4. На третьей строке на координатах G5 находится черный(2) слон(7). Данные что означает каждая цифра дублированы в документации класса и в отдельных таблице БД PIECES и PIECES_COLOR.

А ожидаемое значение находится в таблице RESULT_VALEUS под соответствующим ID. Где ANSWER это решение 0 = Нет шаха, 1 = Есть шах от слона, 2 = Есть шах от ладьи

Так как алгоритм сначала ищет по диагоналям, то приоритетность у слона будет чуть больше чем у ладьи.

```
chess=# SELECT * FROM RESULT_VALUES WHERE ID = 9;
 id | answer
----+--------
  9 |      1
(1 row)
```

Для автоматического тестирования была выбрана технология GitLab CI

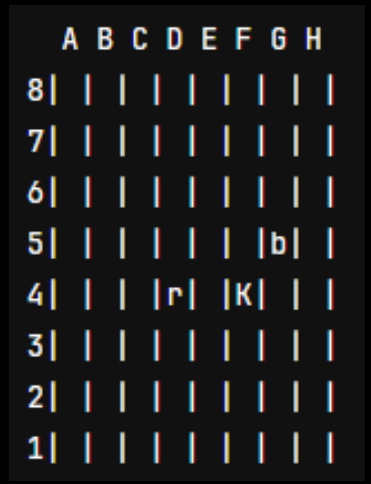
В пайплайне было создано 7 стадий:

- Сборка динамической библиотеки и сохранение библиотеки в артефакты
- Проверка на соблюдение в коде гугл стиля
- Перестройка базы данных для потенциально новых тестов
- Запуск гугл тестов для проверки функциональности и сохранение скомпилированного файла в артефактах
- Сборка документации и сохранение в артефакты
- Сборка исходных файлов в архив и сохранение в артефакты
- Отчет о покрытии и сохранение html отчета в артефактах

Во время запуска пайплайна можно задать переменную DEBUG = 1 и во время тестов будет выводится импровизированная шахматная доска

Справа пример вывода где: K - King; Q - Queen;
N - Knight, B - Bishop; R - Rook; P - Pawn.
A регистр букв отвечает за принадлежность к цвету:
высокий - Белый, маленький - черный

Как раз по выводу можем понять,
что на короля нападают слон и ладья



Заключение

В данной работе был показан пример автоматизации разработки с применением средств автоматизации GitLab-CI, Makefile, Doxygen.

Были сделаны все требуемые цели сборки. Для успешного пайплайна требуется придерживаться код style Google. Настроен пайплайн для проверки и сборка артефактов разных стадий разработки. Так же пайплайн можно запустить в режиме отладки переменной DEBUG.

Написаны тесты которые связанны с базой данных на основе PostgreSQL которые покрыли 100% от исходного кода. Реализована цель для формирования отчета о покрытом коде. Написаны комментарии в стиле Doxygen в заголовочных файлах для автоматической документации.

Все планируемые цели были достигнуты и реализованы.

Выводы

Перед началом разработки нужно сначала построить логическую и математические модели чтобы было более четкое понимание что именно нужно разработать.

Написание тестов помогают разработчику сразу определить где у него может появиться баг или ошибка исполнения.

Разработка стадий автоматизации в пайплайне, одна и самых важных вещей для комфортного отслеживания разработки продукта.

Цели по сборке отчетов, генерации документации облегчают жизнь разработчику.

Стек

Во время разработки будет использоваться стек:

Язык разработки: C++;

Операционная система: Ubuntu Server 20.04 LTS;

Система контроля версий: Git, платформа GitLab;

Система автоматизации локальных действий: Make;

Система автоматизации: GitLab-CI;

Библиотека для написания тестов: GTest;

Средства автоматизированного создания документации: Doxygen;

Система хранения тестовых данных: PostgreSQL.

На виртуальной машине должен быть создан пользователь gitlab-runner и установлен сам gitlab-runner. Через sudo зарегистрировать раннер к проекту.

У пользователя gitlab-runner должны быть зависимости: postgresql, postgresql-contrib, make, g++, clang-format, pkg-config, doxygen, libpqxx-dev, libgtest-dev, lcov. Так же через chown нужно дать доступ к директории /usr/lib.

В psql(postgresql) должен быть добавлен пользователь gitlab-runner как superuser с паролем '1'.

Стандарт компиляции исходного кода C++17.

`Doxyfile версия 1.9.5`

Версии всего ПО явно не указанного в проекте latest.

Стиль кода - Google style.

Источники информации

Библиотека GTest - https://github.com/google/googletest

Библиотека pqxx - https://libpqxx.readthedocs.io/en/stable

Средство автоматизации(GitLab Runner) - https://docs.gitlab.com/runner/