Федеральное государственное образовательное бюджетное учреждение высшего образования "Финансовый университет при правительстве Российской Федерации"

**Выполнение практических заданий по дисциплине «Практикум по программированию»**

Выполнил

Балашкин Андрей Михайлович

Группа ПИ21-4

Проверил

Волков Алексей Викторович

Преподаватель по дисциплине «Практикум по программированию»

Москва, 2021

Оглавление

[**Задание 1 (Супер Ним)** 3](#_Toc88398639)

[**Задача** 3](#_Toc88398640)

[**Решение** 3](#_Toc88398641)

[**Алгоритм решения (блок-схема)** 6](#_Toc88398642)

[**Задание 2 (Пятнашки)** 7](#_Toc88398643)

[**Задача** 7](#_Toc88398644)

[**Решение** 7](#_Toc88398645)

[**Алгоритм решения (блок-схема)** 12](#_Toc88398646)

[**Задание 3 (Морской бой)** 13](#_Toc88398647)

[**Задача** 13](#_Toc88398648)

[**Решение** 13](#_Toc88398649)

[**Алгоритм решения (блок-схема)** 20](#_Toc88398650)

[**Задание 4 (Мостики)** 21](#_Toc88398651)

[**Задача** 21](#_Toc88398652)

[**Решение** 21](#_Toc88398653)

[**Алгоритм решения (блок-схема)** 28](#_Toc88398654)

# **Задание 1 (Супер Ним)**

## **Задача**

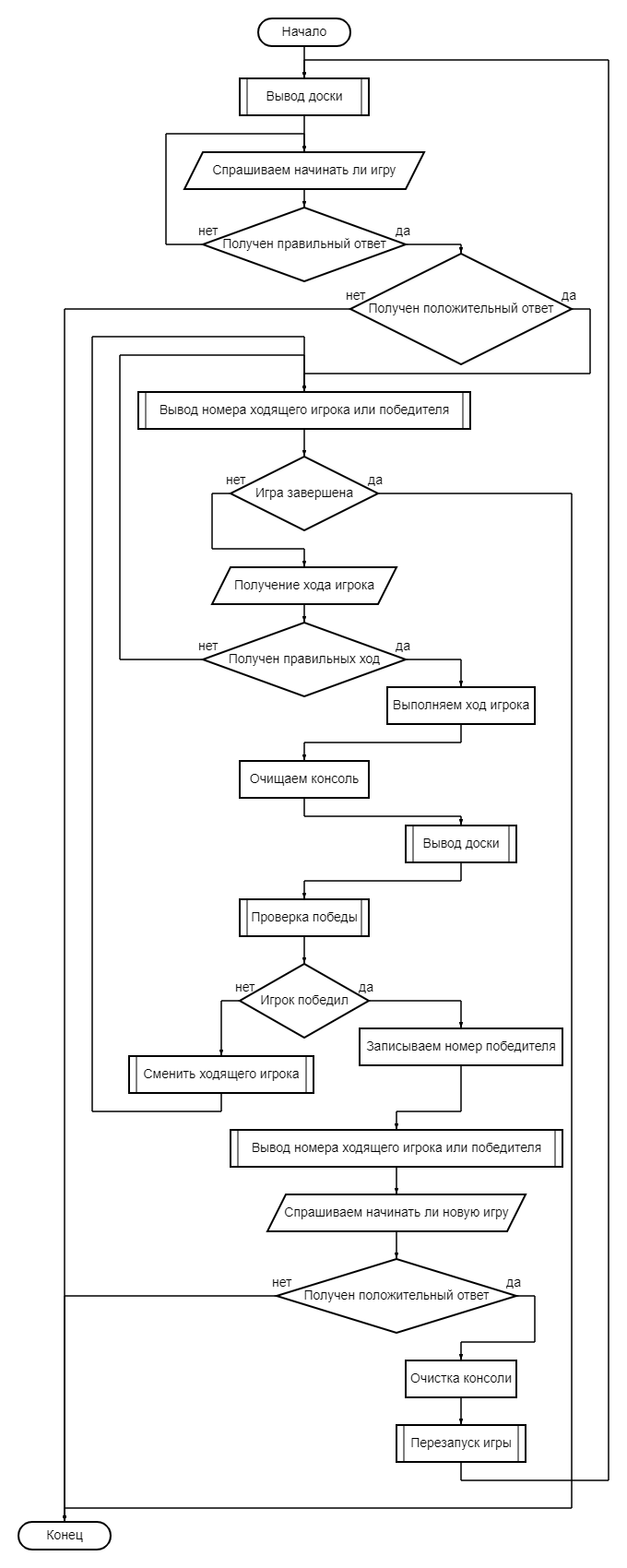
Реализовать программу, при помощи которой 2 игрока могут играть в игру «Супер ним». Правила игры следующие. На шахматной доске в некоторых клетках случайно разбросаны фишки или пуговицы. Игроки ходят по очереди. За один ход можно снять все фишки с какой-либо горизонтали или вертикали, на которой они есть. Выигрывает тот, кто заберет последние фишки. (описание правил игры: https://www.iqfun.ru/articles/super-nim.shtml )

Взаимодействие с программой производится через консоль. Игровое поле изображается в виде текстовых строк и перерисовывается при каждом изменении состояния поля. При запросе данных от пользователя программа сообщает, что ожидает от пользователя (в частности, координаты новой отметки на поле) и проверяет корректность ввода. Программа должна уметь автоматически определять, что партия окончена, и сообщать о победе одного из игроков. Сама программа НЕ ходит, т.е. не пытается выбирать строки или столбцы с целью победить в игре.

## **Решение**

**import** random, os *# Подключаем необходимые библиотеки*Letters = **'abcdefgh'** *# Возможные ходы по вертикали*Numbers = **'12345678'** *# Возможные ходы по горизонтали*Board = [**'-'**] \* 64 *# Доска*Player = **False** *# False - первый игрок, True - второй игрок*IsWin = **False** *# Выиграл ли кто-то*PlayerWin = 0 *# Победивший игрок***def** RestartGame(): *# Метод перезапуска игры* **global** Board, Player, IsWin, PlayerWin *# Задаем глобальные переменные* Board = [**'-'**] \* 64 *# Очищаем доску* Player = **False** *# Даем первый ход первому игроку* IsWin = **False** *# В начале игры никто не выиграл* PlayerWin = 0 *# То никакой игрок не выиграл***def** PrintBoard(): *# Метод вывода доски* print(**'\nИгра "Супер ним"'**) *# Вывод названия игры* print(**' a b c d e f g h'**) *# Вывод навигационного текста* **for** i **in** range(8): *# Цикл от 0 до 7 для прохода по строкам* print(str(i + 1), end=**' '**) *# Выводим навигационный текста (цифры от 1 до 8)* **for** j **in** range(8): *# Цикл от 0 до 7 для прохода по столбцам* print(Board[i \* 8 + j], end=**' '**) *# Выводим текущий элемент с пробелом в конце* print() *# Разделяем строки* print() *# Разделяем доску и другой текст***def** GenerateBoard(): *# Метод генерации доски со случайным кол-ом фишек* **for** i **in** range(random.randint(20, 60) + 1): *# Цикл от 0 до случайного числа (от 20 до 60)* a = random.randint(0, 63) *# Выбираем случайный индекс (место) для текущей фишки* **while** Board[a] == **'\*'**: *# Пока выбранное место занято* a = random.randint(0, 63) *# Заново выбираем индекс* Board[a] = **'\*'** *# После нахождения свободного места записываем в него фишку***def** PrintOrder(): *# Метод вывода номера игрока, который сейчас ходит* **global** IsWin, Player, PlayerWin *# Задаем глобальные переменные* **if not** IsWin: *# Если игра продолжается* a = 1 *# Номер игрока, который сейчас ходит (по умолчанию - 1)* **if** Player: *# Если же сейчас ходит второй игрок* a = 2 *# То записываем это* print(**'Ходит игрок'**, a) *# Выводим номер игрока, который ходит* print(**'Выберите букву или цифру (вертикаль или горизонталь соответственно): '**,  
 end=**''**) *# Выводим сообщение, в котором просим пользователя ввести свой ход* **else**: *# Если какой-то игрок уже выиграл* print(**'Игра завершена\nПобедил игрок'**, PlayerWin) *# То выводим соотв. сообщение и номер победившего игрока***def** CheckWin(): *# Функция проверки выигрыша* **for** i **in** Board: *# Цикл по всем элементам доски* **if** i == **'\*'**: *# Если нашлась хотя бы одна фишка* **return False** *# То игра еще идет* **else**: *# Если за весь цикл не нашлась фишка* **return True** *# То игра закончилась***def** Game(): *# Основной метод игры* **global** IsWin, Player, PlayerWin *# Задаем глобальные переменные* PrintBoard() *# Выводим доску* **while True**: *# Бесконечный цикл для начала игры  
 # a = 'да'* a = input(**'Начинаем игру? (y - да, n - нет)\n'**) *# Спрашиваем пользователя о начале игры* **if** a.lower() == **'y'**: *# Если пользователь ввел символ для начала игры* os.system(**'cls'**) *# Очищаем консоль* GenerateBoard() *# Генерируем доску* PrintBoard() *# Выводим полученную доску* **break** *# Останавливаем цикл while* **elif** a.lower() == **'n'**: *# Если пользователь не хочет играть и вводит соотв. символ* print(**':('**) *# Выводим соотв. сообщение* input(**'Для выхода из программы нажмите "Enter" на клавиатуре'**) *# Ждем нового ввода, чтобы программа сразу не закрывалась и пользователь мог все прочитать* exit() *# Выходим из программы* **else**: *# Если вводится другой символ или строка* print(**'Неправильный ввод! Можно ввести только "y" - да или "n" - нет'**,  
 end=**'\n\n'**) *# То выводим соотв. сообщение* PrintOrder() *# После вывода доски выводим номер игрока, который сейчас ходит* **while not** IsWin: *# Пока никто не выиграл* a = input() *# Получаем ход игрока* IsCorrectInput = **False** *# Переменная для определения корректности ввода* **if** a **in** Letters: *# Если полученный ход - это заранее заданная буква* b = Letters.index(a) *# Получаем индекс этой буквы в алфавите* **for** i **in** range(8): *# Цикл от 0 до 7 для прохода по столбцу* Board[i \* 8 + b] = **'-'** *# Убираем фишки с вертикали* IsCorrectInput = **True** *# Значит получен корректный ввод* **elif** a **in** Numbers: *# Если полученный ход - это заранее заданная цифра* **for** i **in** range(8): *# Цикл от 0 до 7 для прохода по строке* Board[(int(a) - 1) \* 8 + i] = **'-'** *# Убираем фишки с горизонтали* IsCorrectInput = **True** *# Значит получен корректный ввод* **else**: *# Если получен неизвестный символ* print(**'Неправильный ввод! Можно ввести только буквы "a-h" и цифры "1-8"'**) *# Выводим соотв. сообщение и возможные ходы* PrintOrder() *# Снова выводим кто сейчас ходит* **if** IsCorrectInput: *# Если был получен корректный ввод* os.system(**'cls'**) *# Очищаем консоль* PrintBoard() *# Выводим полученную доску* IsWin = CheckWin() *# Проверяем выиграл ли какой-то игрок и записываем это в глобальную переменную* **if** IsWin: *# Если игра закончена* PlayerWin = 1 *# Сохраняем выигравшего игрока (по умолчанию - 1)* **if** Player: *# Если ходил второй игрок, то он выиграл* PlayerWin = 2 *# Записываем номер второго игрока* PrintOrder() *# Выводим номер игрока, который будет ходить или выигравшего игрока* a = input(**'Начать новую игру? (y - да, n - нет) '**) *# Спрашиваем пользователя о начале новой игры* **if** a == **'y'**: *# Если пользователь вводит символ старта новой игры* os.system(**'cls'**) *# Очищаем консоль* RestartGame() *# Запускаем метод перезапуска игры* Game() *# Заново начинаем игру* **else**: *# Если пользователь ввел что-то другое* print(**':('**) *# Выводим соотв. сообщение* input(**'Для выхода из программы нажмите "Enter" на клавиатуре'**) *# Ждем ввода пользователя, чтобы он мог прочитать что-то* **else**: *# Если еще остались фишки* Player = **not** Player *# Передаем ход другому игроку* PrintOrder() *# Выводим номер игрока, который будет ходить или выигравшего игрока*Game() *# Начинаем игру*

## **Алгоритм решения (блок-схема)**



# **Задание 2 (Пятнашки)**

## **Задача**

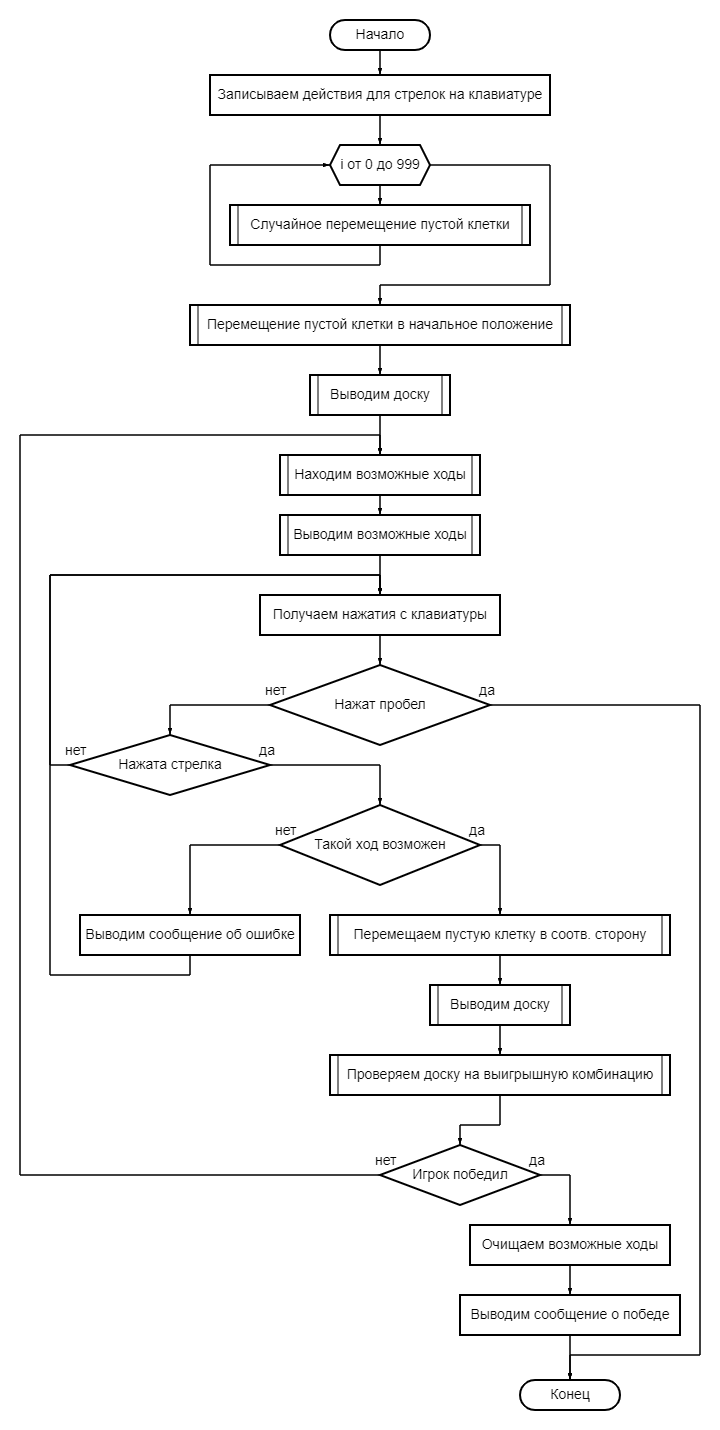
Реализовать программу, с которой можно играть в игру «Пятнашки». Правила игры следующие. Головоломка представляет собой 15 квадратных костяшек с числами от 1 до 15. Все костяшки заключены в квадратную коробку (поле) размером 4 на 4. При размещении костяшек в коробке остается одно пустое место, которое можно использовать для перемещения костяшек внутри коробки. Цель игры - упорядочить размещение чисел в коробке, разместив их по возрастанию слева направо и сверху вниз, начиная с костяшки с номером 1 в левом верхнем углу и заканчивая пустым местом в правом нижнем углу коробки.

Взаимодействие с программой производится через консоль. Игровое поле изображается в виде 4 текстовых строк и перерисовывается при каждом изменении состояния поля. При запросе данных от пользователя программа сообщает, что ожидает от пользователя (например, координаты очередного хода) и проверяет корректность ввода. Программа должна считать количество сделанных ходов, уметь автоматически определять недопустимые ходы, окончание партии и ее победителя. Сама программа НЕ ходит, т.е. не пытается упорядочить костяшки с целью выиграть игру.

## **Решение**

**import** random, os, keyboard *# Подключаем необходимые библиотеки*CurrentWays = [] *# Возможные ходы*IsWin = **False** *# Выиграл ли пользователь*WinBoard = [ *# Выигрышная доска* **'1'**, **'2'**, **'3'**, **'4'**,  
 **'5'**, **'6'**, **'7'**, **'8'**,  
 **'9'**, **'10'**, **'11'**, **'12'**,  
 **'13'**, **'14'**, **'15'**, **'□'**]  
Board = [ *# Текущая доска* **'1'**, **'2'**, **'3'**, **'4'**,  
 **'5'**, **'6'**, **'7'**, **'8'**,  
 **'9'**, **'10'**, **'11'**, **'12'**,  
 **'13'**, **'14'**, **'15'**, **'□'**]  
  
  
**def** GenerateBoard(): *# Метод генерации доски* **global** Board *# Задаем глобальную переменную* steps = [] *# Возможные ходы для квадрата* b = Board.index(**'□'**) *# Получает индекс квадрата* c = b // 4 *# Получаем номер строки, на которой находится квадрат* **if** b + 1 < (c + 1) \* 4: *# Проверяем возможные ходы* steps.append(**'r'**) *# Если ход возможен, то записываем его* **if** b - 1 >= c \* 4:  
 steps.append(**'l'**)  
 **if** b + 4 < 16:  
 steps.append(**'d'**)  
 **if** b - 4 > -1:  
 steps.append(**'u'**)  
  
 d = random.randint(0, len(steps) - 1) *# Выбираем случайный из найденных ход* **if** steps[d] == **'r'**: *# Ищем выбранный ход* Board[b] = Board[b + 1] *# Меняем символы местами* Board[b + 1] = **'□'  
 elif** steps[d] == **'l'**:  
 Board[b] = Board[b - 1]  
 Board[b - 1] = **'□'  
 elif** steps[d] == **'u'**:  
 Board[b] = Board[b - 4]  
 Board[b - 4] = **'□'  
 else**:  
 Board[b] = Board[b + 4]  
 Board[b + 4] = **'□'  
  
  
def** MoveToStart(): *# Метод перемещения квадрата в начальную позицию (справа снизу)* **global** Board *# Задаем глобальную переменную* b = Board.index(**'□'**) *# Находим индекс квадрата* **while** b + 4 < 16: *# Пока возможно двигаться вниз* Board[b] = Board[b + 4] *# Меняем символы местами* Board[b + 4] = **'□'** b += 4 *# Задаем новый индекс квадрата* **while** b + 1 < 16: *# Пока возможно двигаться вправо* Board[b] = Board[b + 1] *# Меняем символы местами* Board[b + 1] = **'□'** b += 1 *# Задаем новый индекс квадрата***def** CheckWays(): *# Метод проверки возможных ходов* **global** Board, CurrentWays *# Задаем глобальные переменные* CurrentWays = [] *# Очищаем текущие ходы* a = Board.index(**'□'**) *# Находим индекс квадрата* **if** a + 1 <= (a // 4) \* 4 + 3: *# Проверяем возможные ходы* CurrentWays.append(**'→'**) *# Если ход возможен, то записываем его* **if** a - 1 >= (a // 4) \* 4:  
 CurrentWays.append(**'←'**)  
 **if** 0 <= a - 4 <= 15:  
 CurrentWays.append(**'↑'**)  
 **if** 0 <= a + 4 <= 15:  
 CurrentWays.append(**'↓'**)  
  
  
**def** CheckWin(): *# Функция проверки победы пользователя* **global** Board, WinBoard *# Задаем глобальные переменные* **if** Board == WinBoard: *# Если текущая доска равна выигрышной* **return True** *# То пользователь победил* **else**:  
 **return False** *# Иначе нет***def** PrintBoard(): *# Метод вывода доски* os.system(**'cls'**) *# Очищаем консоль* print(**'Игра "Пятнашки"'**) *# Выводим название игры* **for** i **in** range(4): *# Цикл по строкам* **for** j **in** range(4): *# Цикл по столбцам* a = Board[i \* 4 + j] *# Получаем символ на текущей позиции* print(a + **' '** \* (2 - len(a)), end=**' '**) *# Выводим текущий символ. Если символ один, то к нему добавляется пробел* print() *# Разделяем строки***def** PrintWays(): *# Метод вывода возможных ходов* a = str(CurrentWays).replace(**'['**, **''**).replace(**']'**, **''**).replace(**"'"**, **''**) *# Записываем возможные ходы в строку* print(**'Вы можете передвинуть квадрат по следующим направлениям:'**, a) *# Выводим их* print(**'Нажмите на одну из этих стрелок на вашей клавиатуре'**) *# Выводим инструкцию***def** MoveLeft(): *# Метод движения квадрата влево* **global** IsWin, CurrentWays *# Задаем глобальные переменные* **if not** IsWin: *# Если игрок еще не победил* **if '←' in** CurrentWays: *# И такой ход возможен* a = Board.index(**'□'**) *# Получаем индекс квадрата* Board[a] = Board[a - 1] *# Меняем символы местами* Board[a - 1] = **'□'** PastMove() *# Вызываем метод для операций после хода* **else**: *# Если такого хода нет в возможных* print(**'Такой ход невозможен'**) *# Выводим сообщение* **else**: *# Если игрок уже выиграл* PastMoveIfWin() *# Вызываем метод для операций после выигрыша***def** MoveRight(): *# Метод движения квадрата вправо* **global** IsWin, CurrentWays *# Задаем глобальные переменные* **if not** IsWin: *# Если игрок еще не победил* **if '→' in** CurrentWays: *# И такой ход возможен* a = Board.index(**'□'**) *# Получаем индекс квадрата* Board[a] = Board[a + 1] *# Меняем символы местами* Board[a + 1] = **'□'** PastMove() *# Вызываем метод для операций после хода* **else**: *# Если такого хода нет в возможных* print(**'Такой ход невозможен'**) *# Выводим сообщение* **else**: *# Если игрок уже выиграл* PastMoveIfWin() *# Вызываем метод для операций после выигрыша***def** MoveUp():  
 **global** IsWin, CurrentWays *# Задаем глобальные переменные* **if not** IsWin: *# Если игрок еще не победил* **if '↑' in** CurrentWays: *# И такой ход возможен* a = Board.index(**'□'**) *# Получаем индекс квадрата* Board[a] = Board[a - 4] *# Меняем символы местами* Board[a - 4] = **'□'** PastMove() *# Вызываем метод для операций после хода* **else**: *# Если такого хода нет в возможных* print(**'Такой ход невозможен'**) *# Выводим сообщение* **else**: *# Если игрок уже выиграл* PastMoveIfWin() *# Вызываем метод для операций после выигрыша***def** MoveDown():  
 **global** IsWin, CurrentWays *# Задаем глобальные переменные* **if not** IsWin: *# Если игрок еще не победил* **if '↓' in** CurrentWays: *# И такой ход возможен* a = Board.index(**'□'**) *# Получаем индекс квадрата* Board[a] = Board[a + 4] *# Меняем символы местами* Board[a + 4] = **'□'** PastMove() *# Вызываем метод для операций после хода* **else**: *# Если такого хода нет в возможных* print(**'Такой ход невозможен'**) *# Выводим сообщение* **else**: *# Если игрок уже выиграл* PastMoveIfWin() *# Вызываем метод для операций после выигрыша***def** PastMove(): *# Метод для операций после хода* **global** IsWin *# Задаем глобальную переменную* PrintBoard() *# Выводим доску* IsWin = CheckWin() *# Проверяем выиграл ли пользователь* **if** IsWin: *# Если пользователь выиграл* PastMoveIfWin() *# Вызываем метод для операций после выигрыша* **else**: *# Если игра еще идет* CheckWays() *# Проверяем возможные ходы* PrintWays() *# Выводим возможные ходы***def** PastMoveIfWin(): *# Метод для операций после выигрыша* CurrentWays = [] *# Очищаем возможные ходы* print(**'Вы выиграли'**) *# Выводим сообщение о победе* print(**'Для выхода из игры нажмите пробел'**) *# Выводим инструкцию для выхода из игры***def** Game():  
 **global** IsWin *# Задаем глобальную переменную* keyboard.add\_hotkey(**'left'**, MoveLeft) *# Создаем действия для стрелок на клавиатуре* keyboard.add\_hotkey(**'right'**, MoveRight)  
 keyboard.add\_hotkey(**'up'**, MoveUp)  
 keyboard.add\_hotkey(**'down'**, MoveDown)  
  
 **for** i **in** range(1000): *# Делаем тысячу перестановок на доске* GenerateBoard() *# Вызывая метод генерации доски* MoveToStart() *# Запускаем метод перемещения квадрата в начальное положение* PrintBoard() *# Выводим доску* CheckWays() *# Находим возможные ходы* PrintWays() *# Выводим возможные ходы* keyboard.wait(**'space'**, suppress=**True**) *# Принимаем нажатие клавиш на клавиатуре пока не нажали пробел*Game() *# Запускаем игру*

## **Алгоритм решения (блок-схема)**



# **Задание 3 (Морской бой)**

## **Задача**

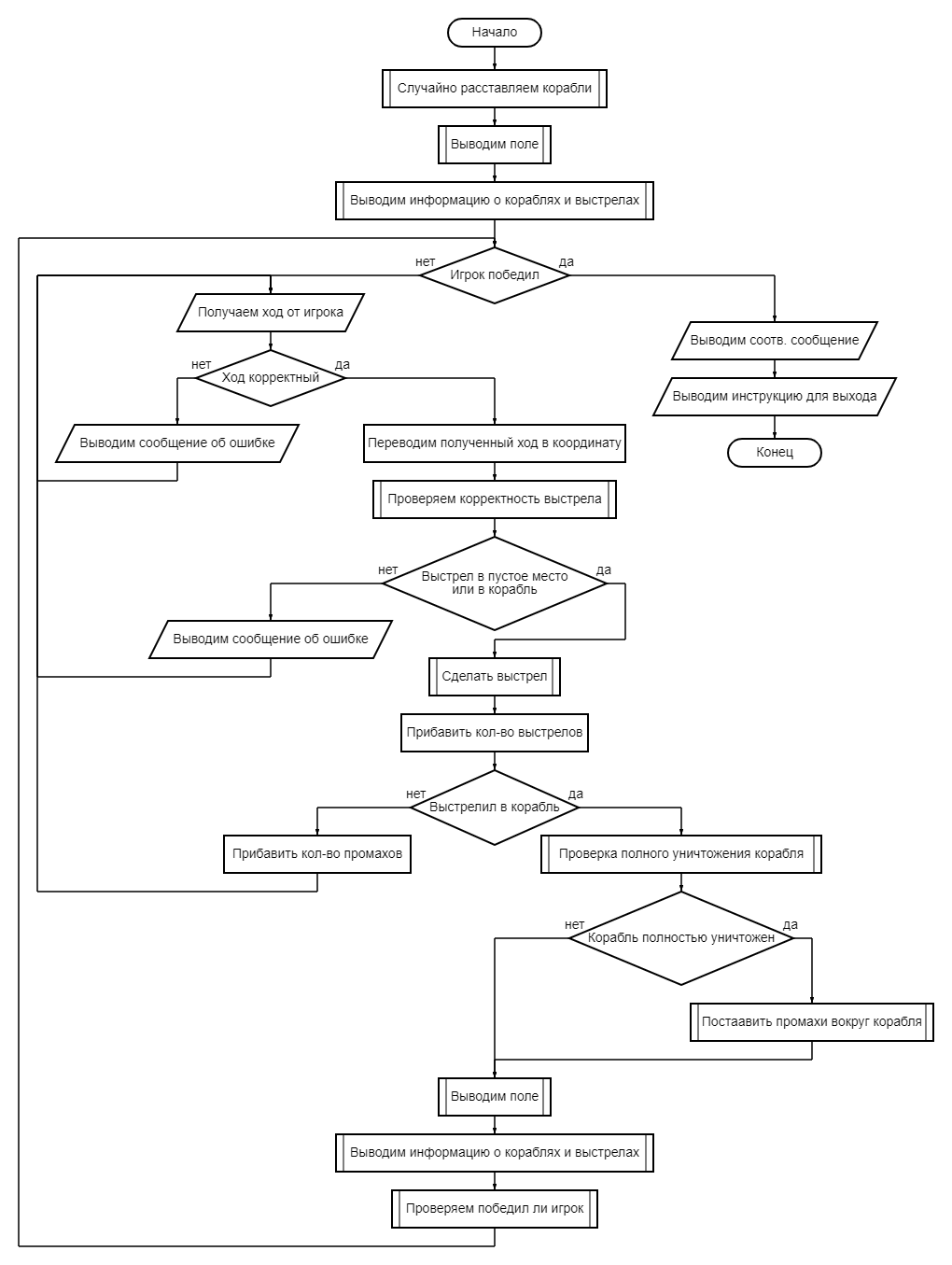
Реализовать программу, с которой можно играть в игру «Морской бой». Программа автоматически случайно расставляет на поле размером 10 на 10 клеток: 4 1-палубных корабля, 3 2-палубных корабля, 2 3-палубных корабля и 1 4-х палубный. Между любыми двумя кораблями по горизонтали и вертикали должна быть как минимум 1 незанятая клетка. Программа позволяет игроку ходить, производя выстрелы. Сама программа НЕ ходит т.е. не пытается топить корабли расставленные игроком.

Взаимодействие с программой производится через консоль. Игровое поле изображается в виде 10 текстовых строк и перерисовывается при каждом изменении состояния поля. При запросе данных от пользователя программа сообщает, что ожидает от пользователя (в частности, координаты очередного «выстрела») и проверяет корректность ввода. Программа должна уметь автоматически определять потопление корабля и окончание партии и сообщать об этих событиях.

## **Решение**

import random, os *# Подключаем необходимые библиотеки*Letters = **'abcdefghij'** *# Возможные ходы по горизонтали*Numbers = **'0123456789'** *# Возможные ходы по вертикали*Board = [**'-'**] \* 100 *# Игровое поле*Exists = 0 *# Кол-во кораблей на поле*CountMiss = 0 *# Кол-во промахов*CountShoot = 0 *# Кол-во выстрелов*IsWin = False *# Победил ли игрок*def GenerateBoard(): *# Создание игрового поля* GenerateShips(1, 4) *# Запускаем методы создания кораблей (один четырехпалубный)* GenerateShips(2, 3) *# два трехпалубных* GenerateShips(3, 2) *# три двухпалубных* GenerateShips(4, 1) *# четыре однопалубных*def GenerateShips(count: int, l: int): *# Метод создания кораблей (кол-во кораблей, кол-во палуб)* global Exists *# Задаем глобальную переменную* for i in range(count): *# Цикл по кол-ву кораблей* x = random.randint(0, 99) *# Выбираем случайную позицию* rotate = random.randint(0, 1) *# Выбираем направление размещения (0 - горизонтально, 1 - вертикально)* while not SetShip(l, x, rotate): *# Пока не получается разместить такой корабль* x = random.randint(0, 99) *# Выбираем новую случайную позицию* rotate = random.randint(0, 1) *# Выбираем новое направление размещения* Exists += count *# Записываем кол-во созданных кораблей*def SetShip(len: int, x: int, r: int): *# Функция размещения корабля на поле (кол-во палуб, позиция, направление размещения)* ok = True *# Получилось ли разместить такой корабль* if Board[x] != **'-'**: *# Если начало корабля занято* ok = False *# То такой корабль нельзя разместить* if ok: *# Если корабль все еще можно разместить* chk = True *# Есть ли место для размещения корабля* for i in range(len): *# Цикл по кол-ву палуб* a = 0 *# Позиция палубы* if r == 0: *# Если корабль должен стоять горизонтально* a = x + i *# Прибавить к начальной позиции текущее кол-во палуб* else: *# Если корабль должен стоять вертикально* a = x + 10 \* i *# Прибавить к начальной позиции текущее кол-во палуб (вниз - это + 10 к позиции)* if not CheckPoint(a): *# Если есть место для текущей палубы* chk = False *# То нет места для размещения корабля* break *# Перестать проверять следующие палубы, это не имеет смысла* if chk: *# Если есть место для размещения корабля* for i in range(len): *# Цикл по кол-ву палуб* if r == 0: *# Если корабль стоит горизонтально* Board[x + i] = **'□'** *# То разместить текущую палубу* elif r == 1: *# Если корабль стоит вертикально* Board[x + 10 \* i] = **'□'** *# То разместить текущую палубу* else: *# Если нет маста для размещения корабля* ok = False *# То не получилось разместить такой корабль* return ok *# Возвращаем получилось ли разместить такой корабль*def CheckPoint(x: int): *# Проверка свободности точки для палубы* ok = True *# Свободна ли точка* if x < 0 or x > 99: *# Если точка лежит за пределами поля* ok = False *# То точка недоступна  
  
 # Проверка точек* if x > 0 and x < 100:  
 if Board[x - 1] != **'-'**:  
 ok = False  
 if x < 99 and x > -1:  
 if Board[x + 1] != **'-'**:  
 ok = False  
 if x > 9 and x < 100:  
 if Board[x - 10] != **'-'**:  
 ok = False  
 if x < 90 and x > -1:  
 if Board[x + 10] != **'-'**:  
 ok = False  
 if x > 10 and x < 100:  
 if Board[x - 1 - 10] != **'-'**:  
 ok = False  
 if x < 91 and x > -1:  
 if Board[x - 1 + 10] != **'-'**:  
 ok = False  
 if x > 8 and x < 100:  
 if Board[x + 1 - 10] != **'-'**:  
 ok = False  
 if x < 89 and x > -1:  
 if Board[x + 1 + 10] != **'-'**:  
 ok = False  
  
 return ok *# Возвращаем свободность точки*def PrintBoard(): *# Метод вывода игрового поля* os.system(**'cls'**) *# Очищаем консоль* print(**'Игра "Морской бой"'**) *# Выводим название игры* print(**' a b c d e f g h i j'**) *# Вывод навигационного текста* for i in range(10): *# Цикл для вывода строк* k = str(i) *# Переменная для навигационного текста* print(k + **' '** \* (2 - len(k)), end=**' '**) *# Вывод навигационного текста* for j in range(10): *# Цикл для вывода столбцов* print(Board[i \* 10 + j].replace(**'□'**, **'-'**), end=**' '**) *# Вывод игрового поля* print() *# Разделение строк*def PrintInfo(): *# Метод вывода информации* print(**'Выстрелы:'**, CountShoot) *# Вывод кол-ва выстрелов* print(**'Промахи:'**, CountMiss) *# Вывод кол-ва промахов* print(**'Осталось'**, Exists, **'кораблей'**) *# Вывод кол-ва оставшихся кораблей* print(**'Куда стреляем? Пример: a1 (первый символ - буква, второй - цифра)'**) *# Спрашиваем ввод и выводим пример хода*def CheckShoot(x: int): *# Функция проверки выстрела* if Board[x] in **'\*x'**: *# Если в эту точку уже стреляли или уже попали в палубу или корабль* return **'В эту точку Вы уже стреляли. Выберите другую'** *# То возвращаем соотв. сообщение* else: *# Если точка пустая или стоит корабль* return **''** *# Возвращаем пустую строку*def Shoot(x: int): *# Метод выстрела* global CountShoot, CountMiss *# Задаем глобальные переменные* CountShoot += 1 *# Прибавляем кол-во выстрелов* if Board[x] == **'-'**: *# Если выстрелили в пустую клетку* Board[x] = **'\*'** *# Ставим на ней промах* CountMiss += 1 *# Прибавляем кол-во промахов* else: *# Если выстрелили в корабль* Board[x] = **'x'** *# То убираем палубу корабля* CheckDead(x) *# Проверяем этот корабль на полное уничтожение*def CheckDead(x: int): *# Метод проверки корабля на полное уничтожение* global Exists *# Задаем глобальную переменную* Dead = True *# Уничтожен ли корабль  
  
 # Проверяем соседние точки на наличие палуб  
 # Если найдены палубы, то корабль еще жив* if x < 99:  
 if Board[x + 1] == **'□'**:  
 Dead = False  
 if x > 0:  
 if Board[x - 1] == **'□'**:  
 Dead = False  
 if x < 90:  
 if Board[x + 10] == **'□'**:  
 Dead = False  
 if x > 9:  
 if Board[x - 10] == **'□'**:  
 Dead = False  
  
 if Dead: *# Если корабль уничтожен полностью* Exists -= 1 *# Убираем один живой корабль* a = [x] *# Точки, вокруг которых надо поставить промахи* for i in range(1, 4): *# Проверка палуб справа* if x + i < 10 \* (x // 10 + 1) and Board[x + i] == **'x'**:  
 a.append(x + 1)  
 else:  
 break  
 for i in range(1, 4): *# Проверка палуб слева* if x - i >= 10 \* (x // 10) and Board[x - i] == **'x'**:  
 a.append(x - i)  
 else:  
 break  
 for i in range(1, 4): *# Проверка палуб снизу* if x + 10 \* i < 100 and Board[x + 10 \* i] == **'x'**:  
 a.append(x + 10 \* i)  
 else:  
 break  
 for i in range(1, 4): *# Проверка палуб сверху* if x - 10 \* i >= 0 and Board[x - 10 \* i] == **'x'**:  
 a.append(x - 10 \* i)  
 else:  
 break  
 for i in a: *# Проход по найденным палубам* SetMisses(i) *# Вызываем метод размещения промахов вокруг найденных точек*def SetMisses(x: int): *# Метод размещения промахов вокруг палуб  
 # Замена пустых точек вокруг палубы на промахи* if x > 0 and x - 1 >= 10 \* (x // 10):  
 Board[x - 1] = Board[x - 1].replace(**'-'**, **'\*'**)  
 if x < 99 and x + 1 < 10 \* (x // 10 + 1):  
 Board[x + 1] = Board[x + 1].replace(**'-'**, **'\*'**)  
 if x > 10 and x - 1 - 10 >= 10 \* (x // 10 - 1):  
 Board[x - 1 - 10] = Board[x - 1 - 10].replace(**'-'**, **'\*'**)  
 if x <= 90 and x - 1 + 10 >= 10 \* (x // 10 + 1):  
 Board[x - 1 + 10] = Board[x - 1 + 10].replace(**'-'**, **'\*'**)  
 if x <= 88 and x + 1 + 10 < 10 \* (x // 10 + 2):  
 Board[x + 1 + 10] = Board[x + 1 + 10].replace(**'-'**, **'\*'**)  
 if x > 9 and x + 1 - 10 > 10 \* (x // 10 - 1):  
 Board[x + 1 - 10] = Board[x + 1 - 10].replace(**'-'**, **'\*'**)  
 if x <= 89:  
 Board[x + 10] = Board[x + 10].replace(**'-'**, **'\*'**)  
 if x >= 10:  
 Board[x - 10] = Board[x - 10].replace(**'-'**, **'\*'**)  
  
  
def CheckWin(): *# Функция проверки победы* global Exists *# Задаем глобальную переменную* if Exists > 0: *# Если остались живые корабли* return False *# То игрок еще не победил* else: *# Если все корабли уничтожены* return True *# Игрок победил*def Game(): *# Основной метод игры* global IsWin *# Задаем глобальную переменную* GenerateBoard() *# Создаем новое игровоеполе* PrintBoard() *# Выводим поле* PrintInfo() *# Выводим информацию* while not IsWin: *# Пока игрок не победил* x = input() *# Получаем от игрока ход* IsCorrectShoot = True *# Корректность введенного хода* if len(x) == 2: *# Если ход состоит из двух символов* if x[0] not in Letters or x[1] not in Numbers: *# Если первый символ не буква, второй не цифра* IsCorrectShoot = False *# То ход неправильный* else: *# Если длина хода не равна двум* IsCorrectShoot = False *# То ход неправильный* if IsCorrectShoot: *# Если получен корректный ход* x = 10 \* int(x[1]) + Letters.index(x[0]) *# Переводим ход в координату* IsShootCorrect = CheckShoot(x) *# Проверяем корректность выстрела* if IsShootCorrect == **''**: *# Если игрок выстрелил в корабль или в пустую клетку* Shoot(x) *# Производим выстрел* PrintBoard() *# Выводим доску* PrintInfo() *# Выводим информацию* IsWin = CheckWin() *# Проверяем победу игрока* else: *# Если игрок выстрелил в уже стрелянную клетку* print(IsShootCorrect) *# Выводим соотв. сообщение* else: *# Если получен некорректный ход* print(**'Неправильный ввод'**) *# Выводим сообщение об ошибке* if IsWin: *# Если игрок победил* print(**'Вы победили'**) *# Выводим соотв. сообщение* print(**'Для выхода из игры нажмини Enter'**) *# Выводим инструкцию для выхода их игры* input() *# Ждем вывод, чтобы консоль сразу не закрывалась*Game() *# Запускаем игру*

## **Алгоритм решения (блок-схема)**



# **Задание 4 (Мостики)**

## **Задача**

Реализовать программу, при помощи которой 2 игрока могут играть в игру «Мостики». Правила игры следующие. В ходе игры каждый из игроков старается построить мост с одного своего берега на другой по камням, образующим массив 4 на 5 (4 камня вдоль берега игрока и 5 камней между берегами). У первого игрока - крестики в качестве камней и берега крестиков (левый и правый край поля), у второго игрока – нолики и берега ноликов (верхний и нижний край поля). Игру можно начинать в любой точке поля. За один ход игрок может соединить два своих соседних камня вертикальным или горизонтальным мостиком (обозначаются в текстовом режиме символами «-» и «|»). Мосты первого и второго игрока пересекаться не должны. Выигрывает тот, кто построит непрерывный мост с одного своего берега на другой. (описание правил игры: https://www.7ya.ru/article/Chem-zanyat-rebenka-13-igr-na-liste-bumagi-so-slovami-kartinkami/ )

Взаимодействие с программой производится через консоль. Игровое поле изображается в виде 9 текстовых строк и перерисовывается при каждом изменении состояния поля. При запросе данных от пользователя программа сообщает, что ожидает от пользователя (например, координаты очередного хода) и проверяет корректность ввода. Программа должна уметь автоматически определять недопустимые ходы (приводящие к пересечению мостов соперников) и окончание партии и ее победителя. Сама программа НЕ ходит, т.е. не пытается строить мосты с целью выиграть игру.

## **Решение**

import os, keyboard *# Подключаем необходимые библиотеки*Board = [  
 **'□'**, **'o'**, **' '**, **'o'**, **' '**, **'o'**, **' '**, **'o'**, **' '**,  
 **'x'**, **' '**, **'x'**, **' '**, **'x'**, **' '**, **'x'**, **' '**, **'x'**,  
 **' '**, **'o'**, **' '**, **'o'**, **' '**, **'o'**, **' '**, **'o'**, **' '**,  
 **'x'**, **' '**, **'x'**, **' '**, **'x'**, **' '**, **'x'**, **' '**, **'x'**,  
 **' '**, **'o'**, **' '**, **'o'**, **' '**, **'o'**, **' '**, **'o'**, **' '**,  
 **'x'**, **' '**, **'x'**, **' '**, **'x'**, **' '**, **'x'**, **' '**, **'x'**,  
 **' '**, **'o'**, **' '**, **'o'**, **' '**, **'o'**, **' '**, **'o'**, **' '**,  
 **'x'**, **' '**, **'x'**, **' '**, **'x'**, **' '**, **'x'**, **' '**, **'x'**,  
 **' '**, **'o'**, **' '**, **'o'**, **' '**, **'o'**, **' '**, **'o'**, **' '**] *# Игровое поле*CurrentSymbol = **' '** *# Символ на месте квадрата*Player = False *# Ходящий игрок*IsWin = False *# Победил ли кто-то*PlayerTranslate = {  
 False: **'крестик'**,  
 True: **'нолик'**} *# Перевод ходящего игрока для вывода в консоль*def PrintBoard(): *# Метод вывода игрового поля* os.system(**'cls'**) *# Очищаем консоль* print(**' -Игра "Мостики"**\n**'**) *# Выводим название игры* print(**' Берег ноликов'**) *# Выводим название берега* print(**' \_\_\_\_\_\_\_\_\_ '**) *# Выводим берег* for i in range(9): *# Цикл по строкам игрового поля* if i != 4: *# На всех строках, кроме пятой, выводим игровое поле* print(**' |'**, end=**''**) *# Выводим берег* for j in range(9): *# Цикл по столбцам* print(Board[i \* 9 + j], end=**''**) *# Выводим элементы игрового поля* print(**'|'**) *# Выводим берег* else: *# На пятой строке выводим игровое поле и название берегов* print(**' Берег крестиков |'**, end=**''**) *# Выводим название берега и берег* for j in range(9): *# Цикл по столбцам* print(Board[i \* 9 + j], end=**''**) *# Выводим элементы игрового поля* print(**'| Берег крестиков'**) *# Выводим берег и название берега* print(**' ¯¯¯¯¯¯¯¯¯'**) *# Выводим берег* print(**' Берег ноликов**\n**'**) *# Выводим название берега* PrintInfo() *# Выводим информацию о ходящем игроке и символе вместо квадрата или победы*def PrintInfo(): *# Метод вывода информации о ходящем игроке и символе вместо квадрата или победы* global IsWin, CurrentSymbol, Player *# Задаем глобальные переменные* if not IsWin: *# Если никто не победил* print(**'На месте квадрата стоит: "'** + CurrentSymbol + **'"'**) *# Выводим символ вместо квадрата* print(**'Ходит'**, PlayerTranslate[Player]) *# Выводим ходящего игрока* print(**'Для передвижения квадрата используйте стрелки на клавиатуре'**) *# Выводим инструкцию по управлению* print(**'Чтобы подтвердить ход нажмите Enter'**) *# Выводим инструкцию для размещения моста* else: *# Если игрок победил* print(**'Выиграл'**, PlayerTranslate[Player]) *# Выводим победившего игрока* print(**'Для выхода из программы нажмите пробел'**) *# Выводим инструкцию для выхода из программы*def MoveRight(): *# Метод передвижения вправо* global CurrentSymbol, IsWin *# Задаем глобальные переменные* if not IsWin: *# Если никто не победил* a = Board.index(**'□'**) *# Получаем индекс квадрата* Board[a] = CurrentSymbol *# Ставим на место квадрата нужный символ* a += 2 *# Сдвигаем квадрат вправо на 2 (перепрыгиваем x и o)* if a > 80: *# Если новая позиция квадрата вне поля* a -= 82 *# То сдвигаемся к началу поля* CurrentSymbol = Board[a] *# Сохраняем новый символ* Board[a] = **'□'** *# Ставим вместо этого символа квадрат* PrintBoard() *# Выводим доску* else: *# Если игрок победил* print(**'Для выхода из программы нажмите пробел'**) *# Выводим инструкцию для выхода из программы*def MoveLeft(): *# Метод передвижения вправо* global CurrentSymbol, IsWin *# Задаем глобальные переменные* if not IsWin: *# Если никто не победил* a = Board.index(**'□'**) *# Получаем индекс квадрата* Board[a] = CurrentSymbol *# Ставим на место квадрата нужный символ* a -= 2 *# Сдвигаем квадрат влево на 2 (перепрыгиваем x и o)* if a < 0: *# Если новая позиция квадрата вне поля* a += 82 *# То сдвигаем к концу поля* CurrentSymbol = Board[a] *# Сохраняем новый символ* Board[a] = **'□'** *# Ставим вместо этого символа квадрат* PrintBoard() *# Выводим доску* else: *# Если игрок победил* print(**'Для выхода из программы нажмите пробел'**) *# Выводим инструкцию для выхода из программы*def MoveUp(): *# Метод передвижения вправо* global CurrentSymbol, IsWin *# Задаем глобальные переменные* if not IsWin: *# Если никто не победил* a = Board.index(**'□'**) *# Получаем индекс квадрата* Board[a] = CurrentSymbol *# Ставим на место квадрата нужный символ* a -= 9 *# Сдвигаем квадрат на одну клетку вверх* if a < 0: *# Если новая позиция квадрата вне поля* a += 81 *# То сдвигаем к концу поля* if Board[a] in **'xo'**: *# Если на новом месте стоит x или o* if a == 9 \* (a // 9): *# Если новую позицию нельзя сместить влево* a += 1 *# То смещаем вправо* else: *# Если можно сместить влево* a -= 1 *# То смещаем влево* CurrentSymbol = Board[a] *# Сохраняем новый символ* Board[a] = **'□'** *# Ставим вместо этого символа квадрат* PrintBoard() *# Выводим доску* else: *# Если игрок победил* print(**'Для выхода из программы нажмите пробел'**) *# Выводим инструкцию для выхода из программы*def MoveDown(): *# Метод передвижения вправо* global CurrentSymbol, IsWin *# Задаем глобальные переменные* if not IsWin: *# Если никто не победил* a = Board.index(**'□'**) *# Получаем индекс квадрата* Board[a] = CurrentSymbol *# Ставим на место квадрата нужный символ* a += 9 *# Сдвигаем квадрат на одну клетку вниз* if a > 80: *# Если новая позиция вне поля* a -= 81 *# То сдвигаем к началу полу* if Board[a] in **'xo'**: *# Если на новом месте стоит x или o* if a == 9 \* (a // 9 + 1) - 1: *# Если новую позицию нельзя сместить вправо* a -= 1 *# То смещаем влево* else: *# Если можно сместить вправо* a += 1 *# То смещаем вправо* CurrentSymbol = Board[a] *# Сохраняем новый символ* Board[a] = **'□'** *# Ставим вместо этого символа квадрат* PrintBoard() *# Выводим доску* else: *# Если игрок победил* print(**'Для выхода из программы нажмите пробел'**) *# Выводим инструкцию для выхода из программы*def SetBridge(): *# Метод размещения моста* global Player, CurrentSymbol *# Задаем глобальные переменные* IsBridgeSet = False *# Поставлен ли мост* if CurrentSymbol not in **'-|'**: *# Если на текущей позиции нет моста* a = Board.index(**'□'**) *# Получаем индекс квадрата* if a > 9 \* (a // 9) and a < 9 \* (a // 9 + 1) - 1: *# Если слева и справа от квадрата есть x или o* if Board[a - 1] == **'o'** and Board[a + 1] == **'o'** and Player \  
 or Board[a - 1] == **'x'** and Board[a + 1] == **'x'** and (not Player): *# Проверка соседних точек* Board[a] = **'-'** *# То ставим мост* IsBridgeSet = True *# Мост поставлен* if a - 9 > 0 and a + 9 < 81 and (not IsBridgeSet): *# Если сверху и снизу от квадрата есть x или o* if Board[a - 9] == **'o'** and Board[a + 9] == **'o'** and Player \  
 or Board[a - 9] == **'x'** and Board[a + 9] == **'x'** and (not Player): *# Проверка соседних точек* Board[a] = **'|'** *# То ставим мост* IsBridgeSet = True *# Мост поставлен* if IsBridgeSet: *# Если мост поставлен* CurrentSymbol = Board[0] *# Сохраняем первый символ* Board[0] = **'□'** *# Ставим на его место квадрат* CheckWin() *# Проверяем победу* PrintBoard() *# Выводим доску* else: *# Если мост не поставлен* print(**'Ошибка! Здесь нельзя поставить мост'**) *# Выводим сообщение об ошибке*def CheckWin(): *# Метод проверки победы* global IsWin, Player *# Задаем глобальные переменные* IsHaveDots = False *# Если есть необходимые мосты* if Player: *# Если ходил нолик* if **'|'** in Board[9:18] and **'|'** in Board[27:36] and **'|'** in Board[45:54] and **'|'** in Board[63:72]:  
 IsHaveDots = True *# Если есть необходимые моста, то записываем это* else: *# Если ходил крестик* a = Board[10] + Board[28] + Board[46] + Board[64] *# Сохраняем точки* b = Board[12] + Board[30] + Board[48] + Board[66]  
 c = Board[14] + Board[32] + Board[50] + Board[68]  
 d = Board[16] + Board[34] + Board[52] + Board[70]  
 if **'-'** in a and **'-'** in b and **'-'** in c and **'-'** in d: *# Если в этих точках есть необходимые* IsHaveDots = True *# То записываем это* if IsHaveDots: *# Если есть необходимые мосты* Steps = [] *# Непрерывные пути* for i in range(len(Board)): *# Цилл по всему игровому полю* ok = False *# Правильный ли мост* if i > 9 and i < 72: *# Если для вертикального моста есть позиции сверху и снизу* if Board[i] == **'|'**: *# Если мост вертикальный* if Player and Board[i - 9] == **'o'** or (not Player) and Board[i - 9] == **'x'**: *# Если мост для нужного игрока* ok = True *# То мост правильный* if i > 1 and i < 79: *# Если для горизонтального моста есть позиции слева и сверху* if Board[i] == **'-'**: *# Если мост горизонтальный* if Player and Board[i - 1] == **'o'** or (not Player) and Board[i - 1] == **'x'**: *# Если мост для нужного игрока* ok = True *# То мост правильный* if ok: *# Если мост правильный* a = FindNear(i) *# Находим соседние мосты* Steps.append([i] + a) *# Добавляем в напрерывные пути найденные точки* k = 1 *# Индекс поиска следующих соседних точек (0 - начальная, 1 - найденная)* while k < len(Steps[-1]): *# Пока есть еще непроверенные точки* a = FindNear((Steps[-1])[k]) *# Находим соседние точки для найденной* for j in a: *# Цикл по найденным соседним точкам* if j not in Steps[-1]: *# Если такой точки нет в найденных* Steps[-1].append(j) *# То добавляем ее в текущий непрерывный путь* k += 1 *# Переходим к следующей найденной точке* if k > 35: *# Если нашлось много точек* k = len(Steps[-1]) *# То остановим цикл* for i in Steps: *# Цикл по найденным непрерывным путям* if Player: *# Если ходил нолик* if 10 in i or 12 in i or 14 in i or 16 in i: *# Проверяем наличие необходимых мостов для одного берега* if 70 in i or 68 in i or 66 in i or 64 in i: *# Проверяем наличие необходимых мостов для другого берега* IsWin = True *# То игрок победил* break *# Останавливаем цикл* else: *# Если ходил крестик* if 10 in i or 28 in i or 46 in i or 64 in i: *# Проверяем наличие необходимых мостов для одного берега* if 16 in i or 34 in i or 52 in i or 70 in i: *# Проверяем наличие необходимых мостов для другого берега* IsWin = True *# То игрок победил* break *# Останавливаем цикл* if not IsWin: *# Если игрок не победил* Player = not Player *# Меняем ходящего игрока*def FindNear(x: int): *# функция проверки соседних мостов* Result = [] *# Найденные мосты  
  
 # Проверка моста слева* if x >= 9 \* (x // 9) + 2:  
 if Board[x] == **'-'**:  
 if Board[x - 2] == **'-'**:  
 Result.append(x - 2)  
 *# Проверка моста справа* if x < 9 \* (x // 9 + 1) - 2:  
 if Board[x] == **'-'**:  
 if Board[x + 2] == **'-'**:  
 Result.append(x + 2)  
 *# Проверка моста сверху* if x > 17:  
 if Board[x] == **'|'**:  
 if Board[x - 18] == **'|'**:  
 Result.append(x - 18)  
 *# Проверка моста снизу* if x < 63:  
 if Board[x] == **'|'**:  
 if Board[x + 18] == **'|'**:  
 Result.append(x + 18)  
 *# Проверка моста снизу справа* if x < 71:  
 if Board[x] == **'-'**:  
 if Board[x + 9 + 1] == **'|'**:  
 Result.append(x + 9 + 1)  
 else:  
 if Board[x + 9 + 1] == **'-'**:  
 Result.append(x + 9 + 1)  
 *# Проверка моста снизу слева* if x + 9 - 1 > 9 \* (x // 9) and x < 71:  
 if Board[x] == **'-'**:  
 if Board[x + 9 - 1] == **'|'**:  
 Result.append(x + 9 - 1)  
 else:  
 if Board[x + 9 - 1] == **'-'**:  
 Result.append(x + 9 - 1)  
 *# Проверка моста сверху справа* if x > 9 and x < 9 \* (x // 9 + 1) - 1:  
 if Board[x] == **'-'**:  
 if Board[x - 9 + 1] == **'|'**:  
 Result.append(x - 9 + 1)  
 else:  
 if Board[x - 9 + 1] == **'-'**:  
 Result.append(x - 9 + 1)  
 *# Проверка моста сверху слева* if x > 9 and x > 9 \* (x // 9):  
 if Board[x] == **'-'**:  
 if Board[x - 9 - 1] == **'|'**:  
 Result.append(x - 9 - 1)  
 else:  
 if Board[x - 9 - 1] == **'-'**:  
 Result.append(x - 9 - 1)  
  
 return Result *# Возвращаем найденные мосты*def Game(): *# Основной метод игры* keyboard.add\_hotkey(**'right'**, MoveRight) *# Создаем действия для стрелок на клвиатуре* keyboard.add\_hotkey(**'left'**, MoveLeft)  
 keyboard.add\_hotkey(**'up'**, MoveUp)  
 keyboard.add\_hotkey(**'down'**, MoveDown)  
 keyboard.add\_hotkey(**'enter'**, SetBridge) *# Создаем действие для размещения моста* PrintBoard() *# Выводим доску* keyboard.wait(**'space'**, suppress=True) *# Принимаем нажатие клавиш на клавиатуре пока не нажали пробел*Game() *# Запускаем игру*

## **Алгоритм решения (блок-схема)**

