Министерство Цифрового Развития, Связи и Массовых Коммуникаций Российской Федерации Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования   
“Московский технический университет связи и информатики”

Кафедра «Информационные технологии»

Лабораторная работа №3:   
«Алгоритм A\* («A star»)**»**

Выполнил: студент гр. БПЗ1902

Кварацхелия Д.Т.

Проверил: Харрасов К.Р.

Москва 2021 г.

Цель работы:

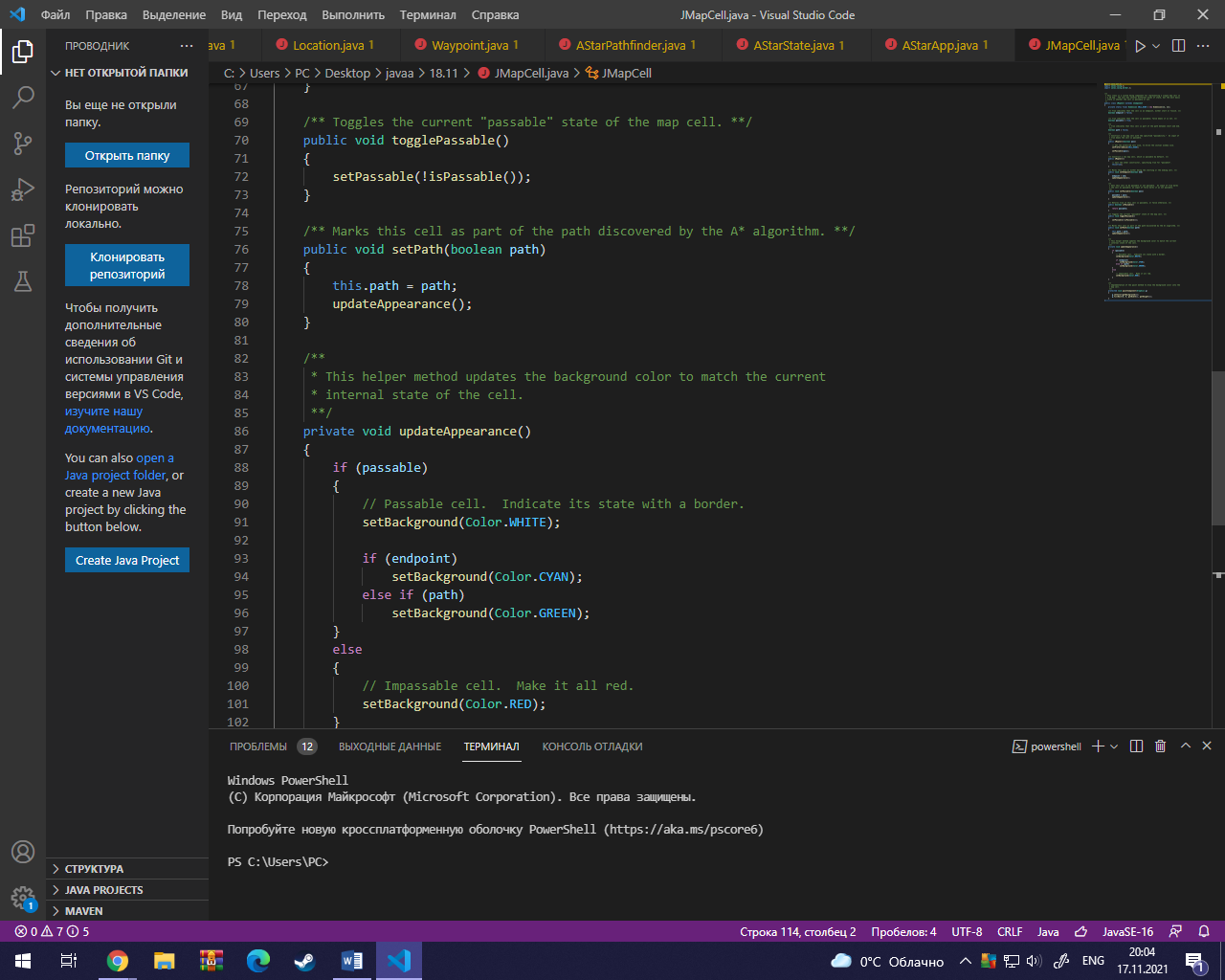
В данной лабораторной работе необходимо изучить программу поиска пути, и исключить возникновение проблем при создании препятствий и последующим поиском путей вокруг них.

Задание:

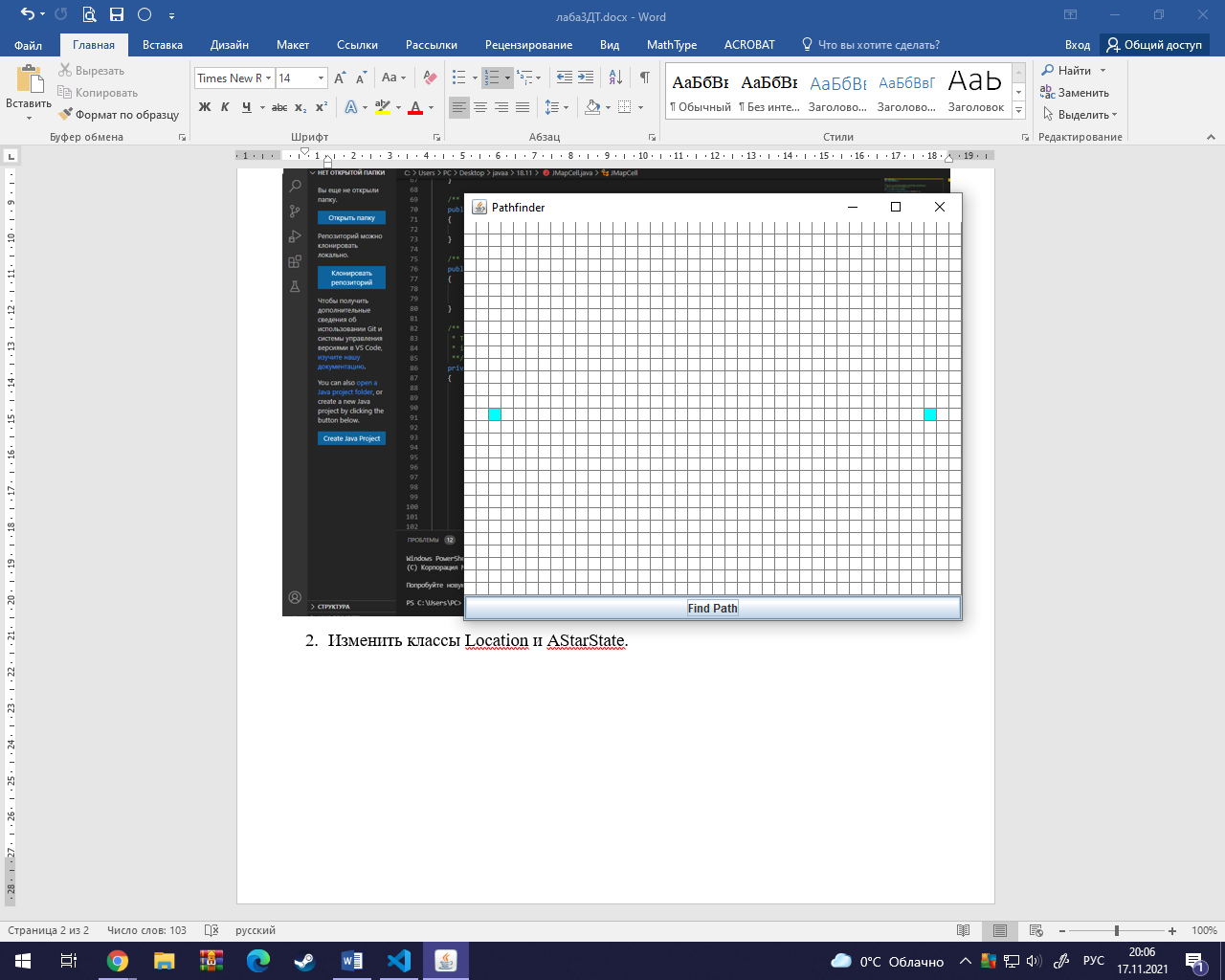
1. Скачать исходные файлы для данной лабораторной работы.
2. Изменить классы Location и AStarState.

Выполнение работы:

1. Скачать исходные файлы для данной лабораторной работы.



Приложение успешно компилируется в том виде, какое оно есть, но функция поиска пути не работает.



1. Изменить классы Location и AStarState.

Location:

Для начала необходимо подготовить класс Location для совместного использования с классами коллекции Java. Мы будем использовать контейнеры для хеширования для выполнения данного задания, для этого необходимо:   
• Обеспечить реализацию метода equals ().

 public boolean equals(Object obj) {

        if (this == obj)

            return true;

        if (obj == null)

            return false;

        if (getClass() != obj.getClass())

            return false;

        Location other = (Location) obj;

        if (xCoord != other.xCoord)

            return false;

        if (yCoord != other.yCoord)

            return false;

        return true;

    }

• Обеспечить реализацию метода hashcode().

public int hashCode() {

        final int prime = 31;

        int result = 1;

        result = prime \* result + xCoord;

        result = prime \* result + yCoord;

        return result;

    }

Код Location:

/\*\*

 \* This class represents a specific location in a 2D map.  Coordinates are

 \* integer values.

 \*\*/

public class Location

{

    /\*\* X coordinate of this location. \*\*/

    public int xCoord;

    /\*\* Y coordinate of this location. \*\*/

    public int yCoord;

    /\*\* Creates a new location with the specified integer coordinates. \*\*/

    public Location(int x, int y)

    {

        xCoord = x;

        yCoord = y;

    }

    /\*\* Creates a new location with coordinates (0, 0). \*\*/

    public Location()

    {

        this(0, 0);

    }

    @Override

    public int hashCode() {

        final int prime = 31;

        int result = 1;

        result = prime \* result + xCoord;

        result = prime \* result + yCoord;

        return result;

    }

    @Override

    public boolean equals(Object obj) {

        if (this == obj)

            return true;

        if (obj == null)

            return false;

        if (getClass() != obj.getClass())

            return false;

        Location other = (Location) obj;

        if (xCoord != other.xCoord)

            return false;

        if (yCoord != other.yCoord)

            return false;

        return true;

    }

}

Теперь мы можем использовать класс Location в качестве ключевого типа в контейнерах хеширования, таких как HashSet и HashMap.

AStarState:

Добавим два (нестатических) поля в класс AStarState с таким типом, одно для "открытых вершин" и другой для "закрытых вершин".   
Кроме того, инициализируем каждое из этих полей для ссылки на новую пустую коллекцию. После создания и инициализации полей, мы реализуем следующие методы в классе AStarState:   
1) public int numOpenWaypoints() Этот метод возвращает количество точек в наборе открытых вершин.

 public int numOpenWaypoints()

    {

        return open.size();

    }

2) public Waypoint getMinOpenWaypoint() Эта функция должна проверить все вершины в наборе открытых вершин, и после этого она должна вернуть ссылку на вершину с наименьшей общей стоимостью. Если в "открытом" наборе нет вершин, функция возвращает NULL. Не удаляйте вершину из набора после того, как вы вернули ее; просто верните ссылку на точку с наименьшей общей стоимостью.

 public Waypoint getMinOpenWaypoint()

    {

        if (numOpenWaypoints() == 0)

            return null;

        Waypoint minWaypoint = null;

        float minCost = Float.MAX\_VALUE;

        for (HashMap.Entry<Location, Waypoint> entry : open.entrySet()) {

            Location key = entry.getKey();

            Waypoint value = entry.getValue();

            if (value.getTotalCost() < minCost) {

                minWaypoint = value;

                minCost = value.getTotalCost();

            }

        }

        return minWaypoint;

    }

3) public boolean addOpenWaypoint(Waypoint newWP) Это самый сложный метод в классе состояний А\*. Данный метод усложняет то, что он должен добавлять указанную вершину только в том случае, если существующая вершина хуже новой. Вот что должен делать этот метод:

• Если в наборе «открытых вершин» в настоящее время нет вершины для данного местоположения, то необходимо просто добавить новую вершину.  
 • Если в наборе «открытых вершин» уже есть вершина для этой локации, добавьте новую вершину только в том случае, если стоимость пути до новой вершины меньше стоимости пути до текущей. (Убедитесь, что используете не общую стоимость.) Другими словами, если путь через новую вершину короче, чем путь через текущую вершину, замените текущую вершину на новую

public boolean addOpenWaypoint(Waypoint newWP)

    {

        Location location = newWP.getLocation();

        if (open.containsKey(location)) {

            Waypoint currWP = open.get(location);

            if (newWP.getPreviousCost() < currWP.getPreviousCost()) {

                open.put(location, newWP);

                return true;

            }

            return false;

        }

        open.put(location, newWP);

        return true;

    }

4) public boolean isLocationClosed(Location loc) Эта функция должна возвращать значение true, если указанное местоположение встречается в наборе закрытых вершин, и false в противном случае. Так как закрытые вершины хранятся в хэш-карте с расположениями в качестве ключевых значений, данный метод достаточно просто в реализации.

public boolean isLocationClosed(Location loc)

    {

        return close.containsKey(loc);

    }

5) public void closeWaypoint(Location loc) Эта функция перемещает вершину из набора «открытых вершин» в набор «закрытых вершин». Так как вершины обозначены местоположением, метод принимает местоположение вершины. Процесс должен быть простым:   
• Удалите вершину, соответствующую указанному местоположению из набора «открытых вершин».   
• Добавьте вершину, которую вы удалили, в набор закрытых вершин. Ключом должно являться местоположение точки.

 public void closeWaypoint(Location loc)

    {

        Waypoint waypoint = open.remove(loc);

        close.put(loc, waypoint);

    }

Код AStarState:

/\*\*

 \* This class stores the basic state necessary for the A\* algorithm to compute a

 \* path across a map.  This state includes a collection of "open waypoints" and

 \* another collection of "closed waypoints."  In addition, this class provides

 \* the basic operations that the A\* pathfinding algorithm needs to perform its

 \* processing.

 \*\*/

import  java.util.\*;

public class AStarState

{

    /\*\* This is a reference to the map that the A\* algorithm is navigating. \*\*/

    private Map2D map;

    private HashMap<Location, Waypoint> open = new HashMap<>();;

    private HashMap<Location, Waypoint> close = new HashMap<>();;

    /\*\*

     \* Initialize a new state object for the A\* pathfinding algorithm to use.

     \*\*/

    public AStarState(Map2D map)

    {

        if (map == null)

            throw new NullPointerException("map cannot be null");

        this.map = map;

    }

    /\*\* Returns the map that the A\* pathfinder is navigating. \*\*/

    public Map2D getMap()

    {

        return map;

    }

    /\*\*

     \* This method scans through all open waypoints, and returns the waypoint

     \* with the minimum total cost.  If there are no open waypoints, this method

     \* returns <code>null</code>.

     \*\*/

    public Waypoint getMinOpenWaypoint()

    {

        if (numOpenWaypoints() == 0)

            return null;

        Waypoint minWaypoint = null;

        float minCost = Float.MAX\_VALUE;

        for (HashMap.Entry<Location, Waypoint> entry : open.entrySet()) {

            Location key = entry.getKey();

            Waypoint value = entry.getValue();

            if (value.getTotalCost() < minCost) {

                minWaypoint = value;

                minCost = value.getTotalCost();

            }

        }

        return minWaypoint;

    }

    /\*\*

     \* This method adds a waypoint to (or potentially updates a waypoint already

     \* in) the "open waypoints" collection.  If there is not already an open

     \* waypoint at the new waypoint's location then the new waypoint is simply

     \* added to the collection.  However, if there is already a waypoint at the

     \* new waypoint's location, the new waypoint replaces the old one <em>only

     \* if</em> the new waypoint's "previous cost" value is less than the current

     \* waypoint's "previous cost" value.

     \*\*/

    public boolean addOpenWaypoint(Waypoint newWP)

    {

        Location location = newWP.getLocation();

        if (open.containsKey(location)) {

            Waypoint currWP = open.get(location);

            if (newWP.getPreviousCost() < currWP.getPreviousCost()) {

                open.put(location, newWP);

                return true;

            }

            return false;

        }

        open.put(location, newWP);

        return true;

    }

    /\*\* Returns the current number of open waypoints. \*\*/

    public int numOpenWaypoints()

    {

        return open.size();

    }

    /\*\*

     \* This method moves the waypoint at the specified location from the

     \* open list to the closed list.

     \*\*/

    public void closeWaypoint(Location loc)

    {

        Waypoint waypoint = open.remove(loc);

        close.put(loc, waypoint);

    }

    /\*\*

     \* Returns true if the collection of closed waypoints contains a waypoint

     \* for the specified location.

     \*\*/

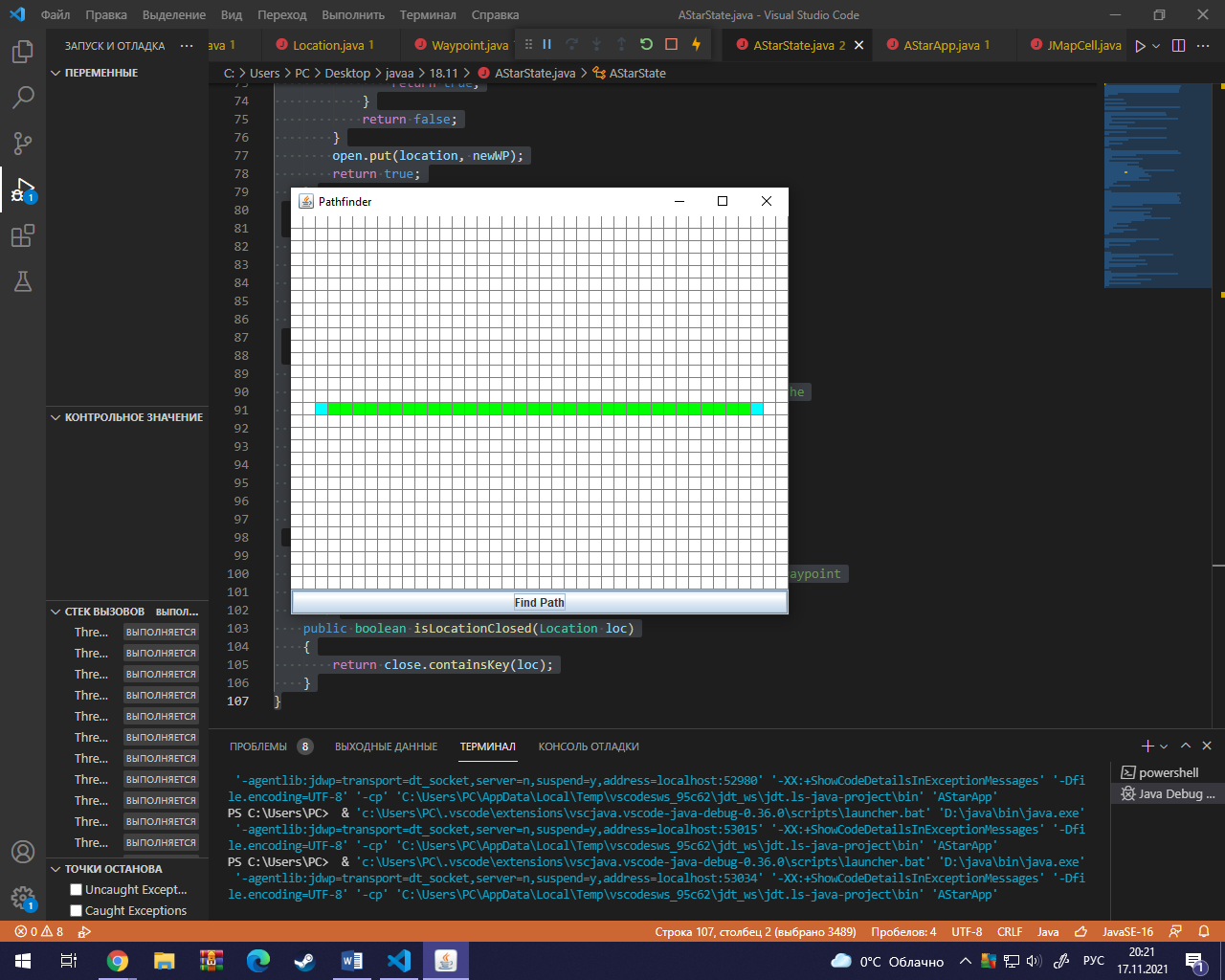
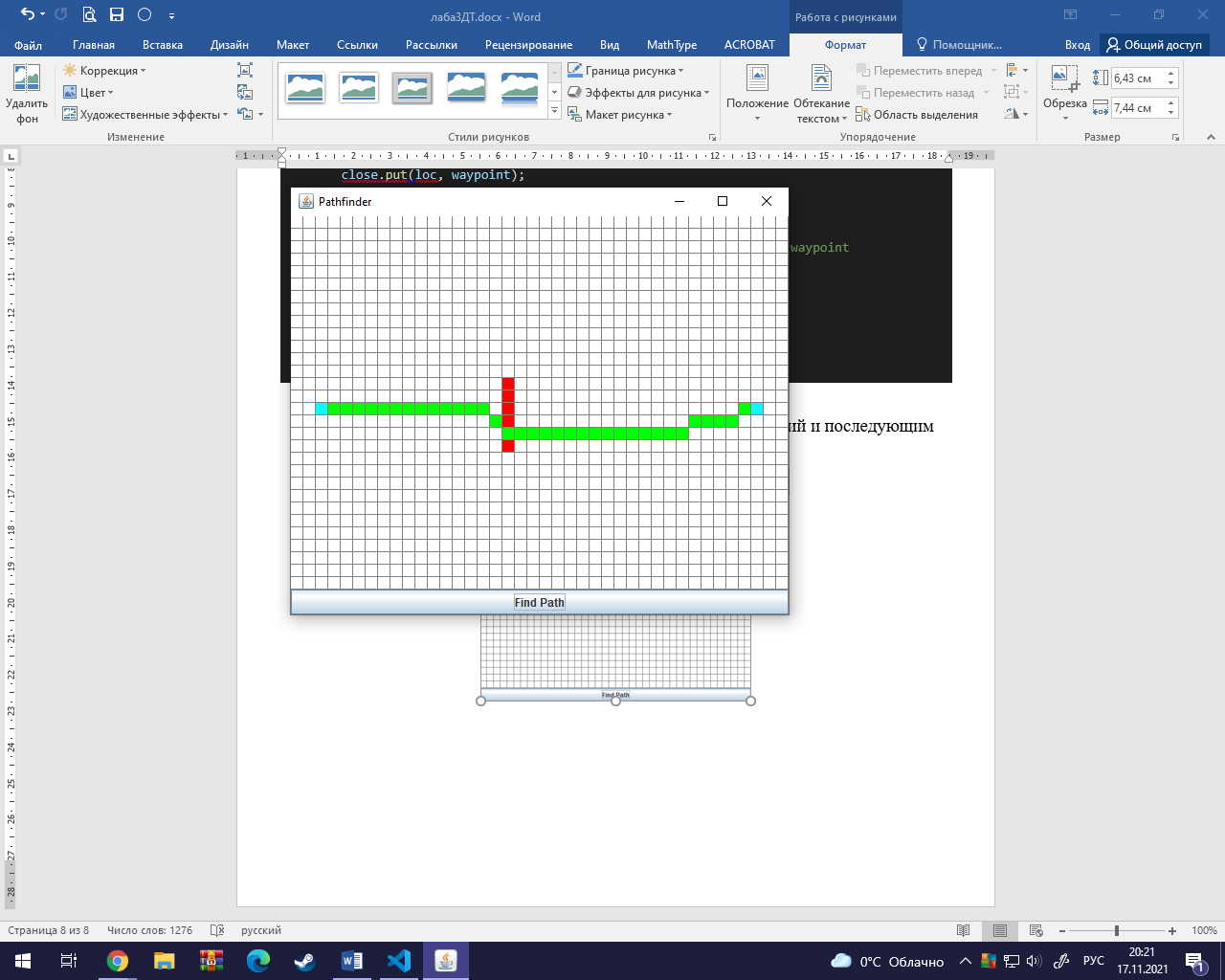
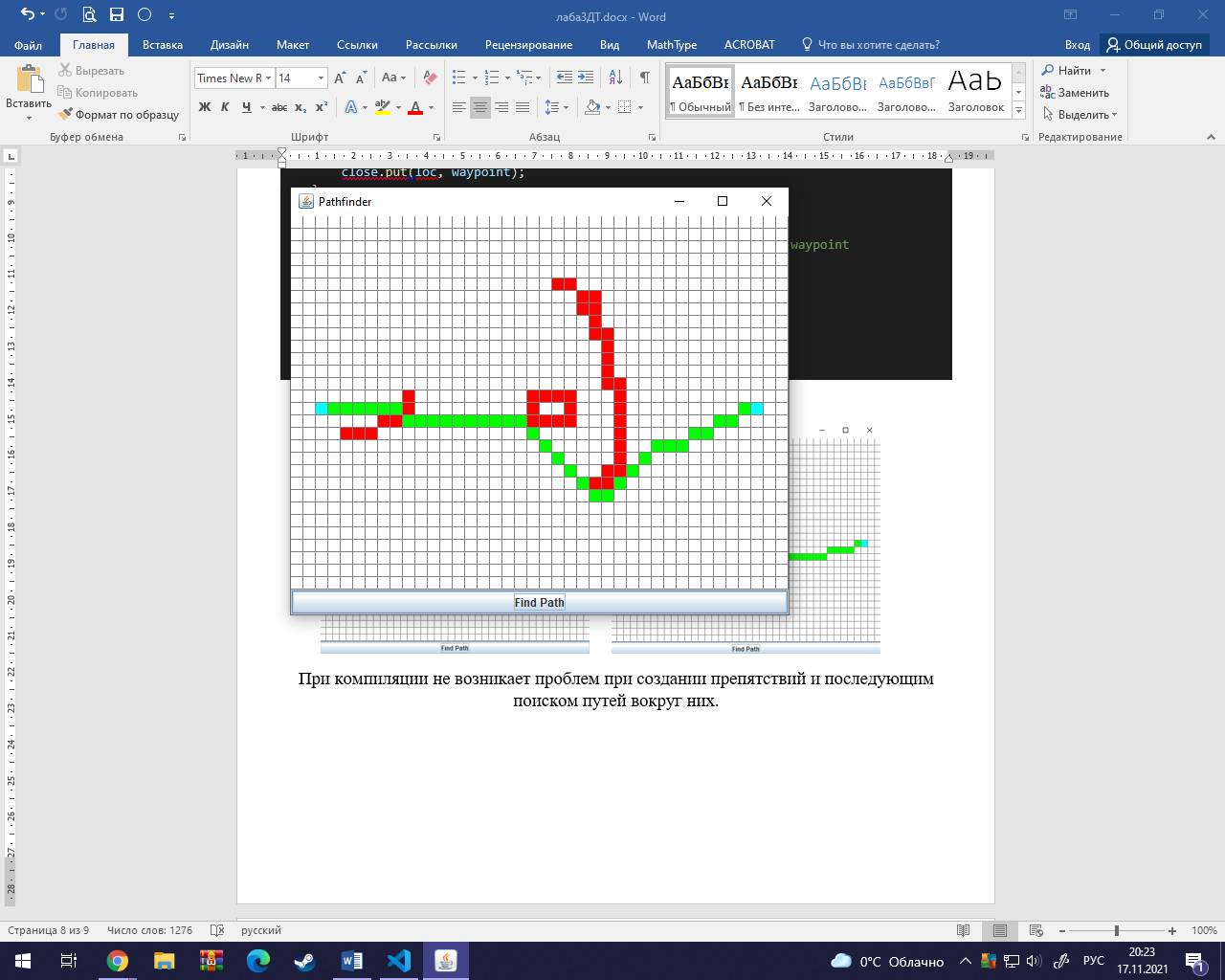
    public boolean isLocationClosed(Location loc)

    {

        return close.containsKey(loc);

    }

}

  
Компиляция и тестирование:

При компиляции не возникает проблем при создании препятствий и последующим поиском путей вокруг них.