Group tasks

1. За представянето на лабиринта в програмата ще използвам двумерен масив с булеви стойности, защото има само два типа клетки. На мястото на клетките, които са стени има стойност False, а в останалите стойност True. Ще създам и двумерен масив със същата големина, в който ще пазя дали дадена клетка вече е посетена, тъй като е безмислено да имаме цикли в нашите пътища. Ще създам и клас cellNode, в който пазим координатите на клетка и пътя от началото до нея.
2. В общия случай критерият за това дали дадена клетка е финална е: тази клетка има индекс на ред с едно по-малък от размера на дадения масив за лабиринта и индекс на колона с едно по-малък от размера на първия подмасив в масива на лабиринта. За конкретния случай, търсим клетка с ред 4 и колона 5.
3. Ще добавяме съседните клетки на разглежданата в опашка, чрез функцията loadQueue. Трябва да проверим три неща: дали ще излезем извън границите, дали съседната клетка е стена и дали тя вече е била посетена. Пътят на новата клетка се образува от пътя до предишната плюс посоката, в която току що сме тръгнали.
4. За евристика ще използвааме Манхатънско разстояние.
5. Пазим пътя в член-данната path на cellNode.
6. Дължината на пътя я намираме чрез метода cost na cellNode, това е просто дължината на член-данната path
7. За решаване на задачата ще използваме търсене в ширина, знаем че когато преходите са с еднаква цена с този метод откриваме минимални пътища. Алгоритъмът ще спре когато получим cellNode, за който isFinal() ще върне True.
8. Пътят, който получихме е DRRRRRDDD, с дължина 9.

Individual work

1. Другия алгоритъм, който ще използвам е A*. Отново ще се нуждая опашка, но този път ще е приоритетна. В cellNode добавих функция h(), която е сбор от изминатия път до клетката и евристичната функция манхатън. Дефинирах и функция __lt__, която определя кога един cellNode е по-малък от друг, а именно когато той има по-малка стойност, когато двата викнат h(). Така когато опашката извика своя get() метод, тя ще вади този cellNode с най-малка стойност на h(). Добавянето на елементи в опашката става по-същия начин, както при bfs(). Отново алгоритъмът спира, когато намерим cellNode, за който isFinal() връща True.
2. Това са всички клетки, които сме вкарвали и изкарвали от опашката:
   Going out of the queue: row: 0 col: 0
   Going into the queue: row: 1 col: 0
   Going out of the queue: row: 1 col: 0
   Going into the queue: row: 1 col: 1
   Going into the queue: row: 2 col: 0
   Going out of the queue: row: 1 col: 1
   Going into the queue: row: 1 col: 2

Going out of the queue: row: 2 col: 0
Going into the queue: row: 3 col: 0
Going out of the queue: row: 1 col: 2
Going into the queue: row: 1 col: 3
Going into the queue: row: 0 col: 2
Going into the queue: row: 2 col: 2
Going out of the queue: row: 3 col: 0
Going into the queue: row: 4 col: 0
Going out of the queue: row: 1 col: 3
Going into the queue: row: 1 col: 4
Going into the queue: row: 0 col: 3
Going out of the queue: row: 2 col: 2
Going into the queue: row: 3 col: 2
Going out of the queue: row: 4 col: 0
Going into the queue: row: 4 col: 1
Going out of the queue: row: 1 col: 4
Going into the queue: row: 1 col: 5
Going into the queue: row: 0 col: 4
Going into the queue: row: 2 col: 4
Going out of the queue: row: 3 col: 2
Going into the queue: row: 3 col: 3
Going into the queue: row: 4 col: 2
Going out of the queue: row: 2 col: 4
Going into the queue: row: 2 col: 5
Going out of the queue: row: 3 col: 3
Going into the queue: row: 4 col: 3
Going out of the queue: row: 4 col: 1
Going out of the queue: row: 1 col: 5
Going into the queue: row: 0 col: 5
Going out of the queue: row: 2 col: 5
Going into the queue: row: 3 col: 5
Going out of the queue: row: 4 col: 3
Going out of the queue: row: 3 col: 5
Going into the queue: row: 4 col: 5
Going out of the queue: row: 4 col: 2
Going out of the queue: row: 4 col: 5

3. За да нарисуваме какво прави алгоритъмът A* ще ни трябва библиотеката tkinter. Първоначално рисуваме цялото поле като празните клетки са светло сини, а тези със стена са червени. Когато клетка влезе в опашката ще я оцветяваме в зелено, за целта си дефинираме член-функцията colorVisited() и я викаме в loadQueue(). Когато вадим клетка от опашката ще я боядисваме в жълто това се прави while цикъла на aStar(). Умишлено промените стават през половин секунда, за да са видими за човешкото око, това се постига благодарение на функцията after(). Накрая рисуваме пътя с черни квадратчета.

4. И двата алгоритъма в най-лошия случай ще обходят всички клетки, A* може да се забави повече при вкарването на клетки в опашката, тъй като тя е приоритетна, но в някои случаи може да обходи по-малко клетки, защото се стреми да ходи към посока финалната клетка и избягва пътищата, които изглеждат, че няма да доведът до добър резултат, докато bfs търси неинформирано. От гледна точка на памет bfs ще има нужда от толкова, колкото е големината на последното обхождано ниво, в някои случаи A* ще се нуждае от много по-малко памет, защото ще вкарва по малко елементи в опашката.