ЛАБОРАТОРНАЯ РАБОТА № 2
«ИССЛЕДОВАНИЕ НЕИНФОРМИРОВАННЫХ МЕТОДОВ ПОИСКА РЕШЕНИЙ ЗАДАЧ В ПРОСТРАНСТВЕ СОСТОЯНИЙ»

Цель работы

Исследование неинформированных методов поиска решений задач в пространстве состояний, приобретение навыков программирования интеллектуальных агентов, планирующих действия на основе методов слепого поиска решений задач.

Постановка задачи

1. Изучить по лекционному материалу и учебным пособиям [1-3] методы слепого поиска решений задач в пространстве состояний.

2. Изучить структуры данных Stack, Queue и PriorityQueue, предоставленные в модуле util.py.

3. Изучить методы среды AI Pacman: problem.getStartState() , problem.isGoalState(), problem.getSuccessors(). Для этого проверить выполнение команды python pacman.py -l tinyMaze -p SearchAgent для случая, когда реализация функции depthFirstSearch содержит только вызовы операторов печати print("Start:", problem.getStartState()) print("Is the start a goal?", problem.isGoalState(problem.getStartState())) print("Start's successors:", problem.getSuccessors(problem.getStartState()))

Разобраться с типами значений, возвращаемых методами и использовать их при написании кода, реализующего алгоритмы поиска DFS, BFS, UCS.

4. Определить в соответствии с заданиями 1-3 раздела 2.3 функции, реализующие алгоритмы поиска DFS, BFS, UCS. При реализации алгоритмов

поиска рекомендуется использовать псевдокод из раздела 2.2.2. Для реализации списка OPEN в алгоритме UCS следует использовать очередь с приоритетами PriorityQueue.

5. Зафиксировать результаты использования функций для всех лабиринтов, указанных в заданиях 1-3. Ответить письменно на предлагаемые в заданиях 1-3 вопросы.

6. Выполнить с помощью autograder.py автооценивание заданий 1-3. При обнаружении ошибок отредактировать код. Результаты автооценивания внести в отчет.

7. Оценить эффективность используемых методов поиска по критериям временной и пространственной сложности.

Ход работы

1. Поиск в глубину.

Была проверена правильность работы SearchAgent, для чего была выполнена команда python3.6 pacman.py -l tinyMaze -p SearchAgent -a fn=tinyMazeSearch. Результат выполнения команды продемонстрирован на рисунке 1.

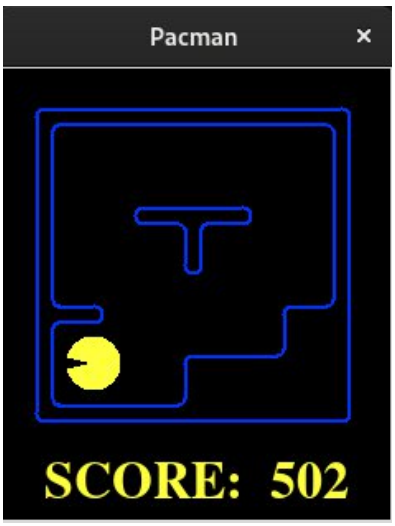

Рисунок 1 – Результат поиска в маленьком лабиринте

Далее, был реализован алгоритм поиска в глубину, код которого представлен в листинге 1.

Листинг 1 – Алгоритм поиска в глубину

```
def depthFirstSearch(problem):
    """
    Поиск самого глубокого узла в дереве поиска.

    Ваш алгоритм поиска должен возвращать список действий, которые
    ведут к цели. Убедитесь, что реализуете алгоритм поиска на графе

    Прежде чем кодировать,полезно выполнить функцию  с этими простыми
    командами,чтобы понять смысл задачи (problem), передаваемой на вход:

    print("Start:", problem.getStartState())
    print("Is the start a goal?", problem.isGoalState(problem.getStartState()))
    print("Start's successors:", problem.getSuccessors(problem.getStartState()))
    """
    "*** ВСТАВЬТЕ ВАШ КОД СЮДА ***"

    frontier = util.Stack()

    visited = []

    start = problem.getStartState()
    startNode = (start, [])

    frontier.push(startNode)

    while not frontier.isEmpty():
        currentState, actions = frontier.pop()

        if currentState not in visited:
            visited.append(currentState)

            if problem.isGoalState(currentState):
                return actions
            else:
                successors = problem.getSuccessors(currentState)

                for sState, sAction, sCost in successors:
                    newAction = actions + [sAction]
                    newNode = (sState, newAction)
                    frontier.push(newNode)

    return actions
```

Затем данный алгоритм был проверен для поиска решений для маленького, среднего и большого лабиринтов. Результаты прохождения лабиринтов представлены соответственно на рисунках 2, 3 и 4.

Рисунок 2 – Поиск в глубину в маленьком лабиринте

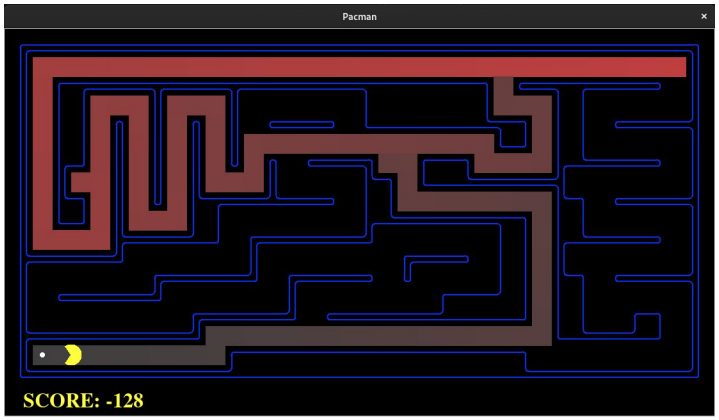

Рисунок 3 – Поиск в глубину в среднем лабиринте

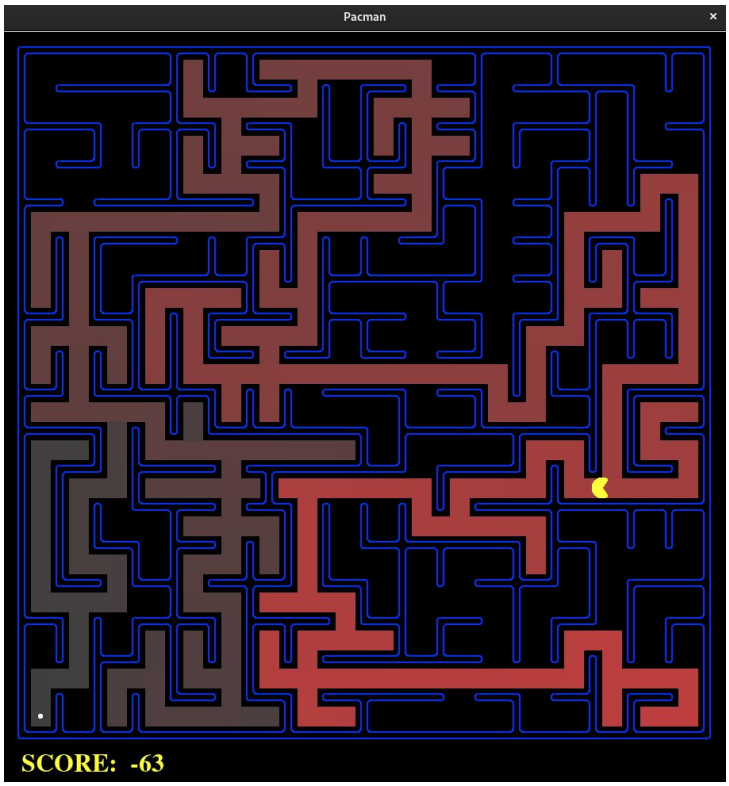

Рисунок 4 – Поиск в глубину в большом лабиринте

После чего написанное решение было проверено при помощи автооценивателя, для чего была выполнена команда python3.6 autograder.py -q q1. Результат прохождения тестов представлен на рисунке 5.

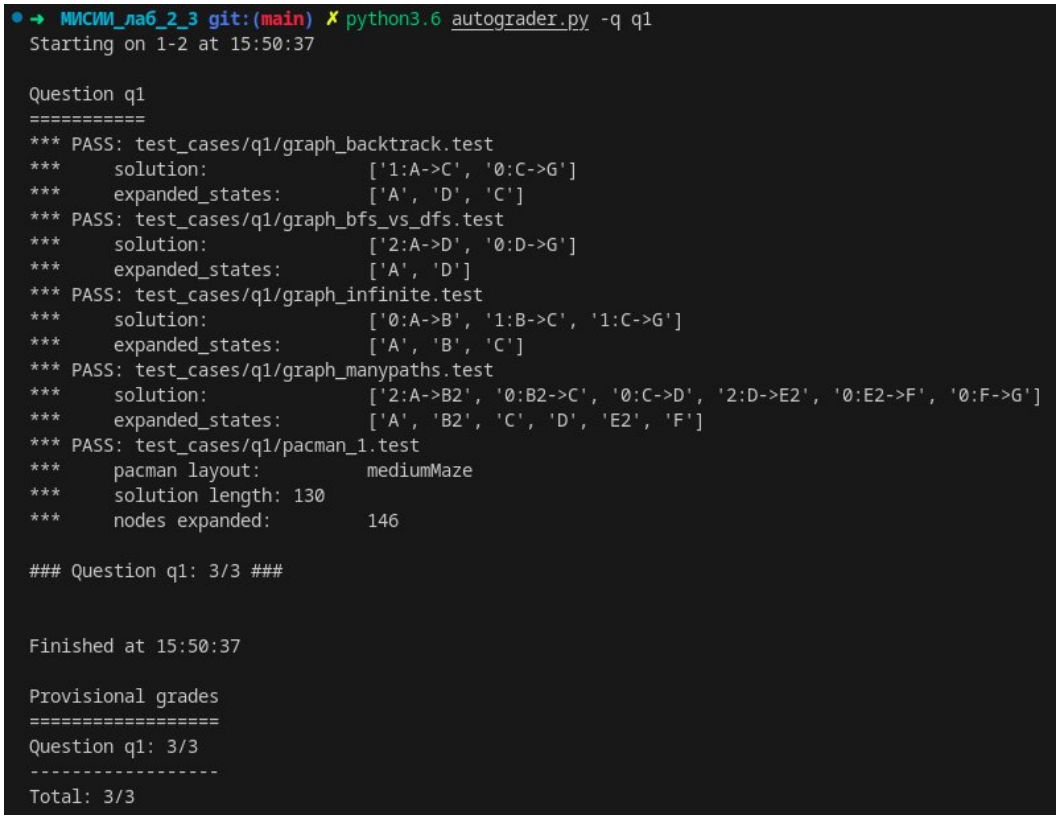

Рисунок 5 – Прохождение тестов при помощи автооценивателя

2. Поиск в ширину.

Был реализован алгоритм поиска в глубину, код которого представлен в листинге 2.

Листинг 2 – Алгоритм поиска в ширину

```
def breadthFirstSearch(problem):
    """Находит самые поверхностные узлы в дереве поиска """
    "*** ВСТАВЬТЕ ВАШ КОД СЮДА ***"

    frontier = util.Queue()

    visited = []

    start = problem.getStartState()
    startNode = (start, [], 0)

    frontier.push(startNode)
```

```
while not frontier.isEmpty():
    currentState, actions, currentCost = frontier.pop()
    if currentState not in visited:
        visited.append(currentState)

        if problem.isGoalState(currentState):
            return actions
        else:
            successors = problem.getSuccessors(currentState)

            for sState, sAction, sCost in successors:
                newAction = actions + [sAction]
                newCost = currentCost + sCost
                newNode = (sState, newAction, newCost)

                frontier.push(newNode)

return actions
```

После чего данный алгоритм был проверен для поиска решений в маленьком, среднем и большом лабиринтах, для чего были выполнены следующие команды:

python3.6 pacman.py -l tinyMaze -p SearchAgent -a fn=breadthFirstSearch

python3.6 pacman.py -l mediumMaze -p SearchAgent -a fn=breadthFirstSearch

python3.6 pacman.py -l bigMaze –z .8 -p SearchAgent -a fn=breadthFirstSearch

На рисунках 6, 7 и 8 представлены результаты выполнения команд.

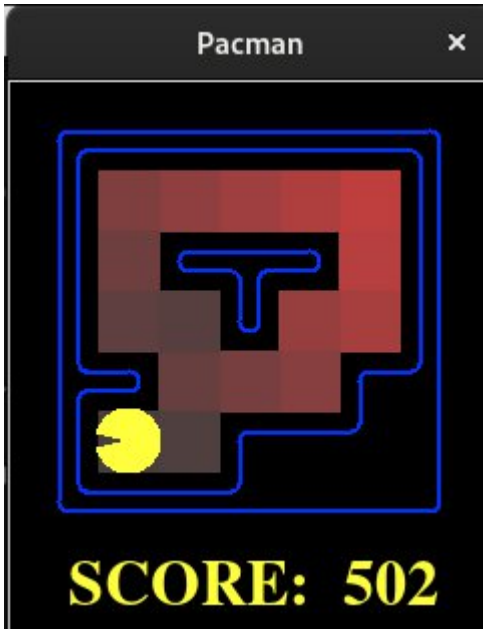

Рисунок 6 – Поиск в ширину в маленьком лабиринте

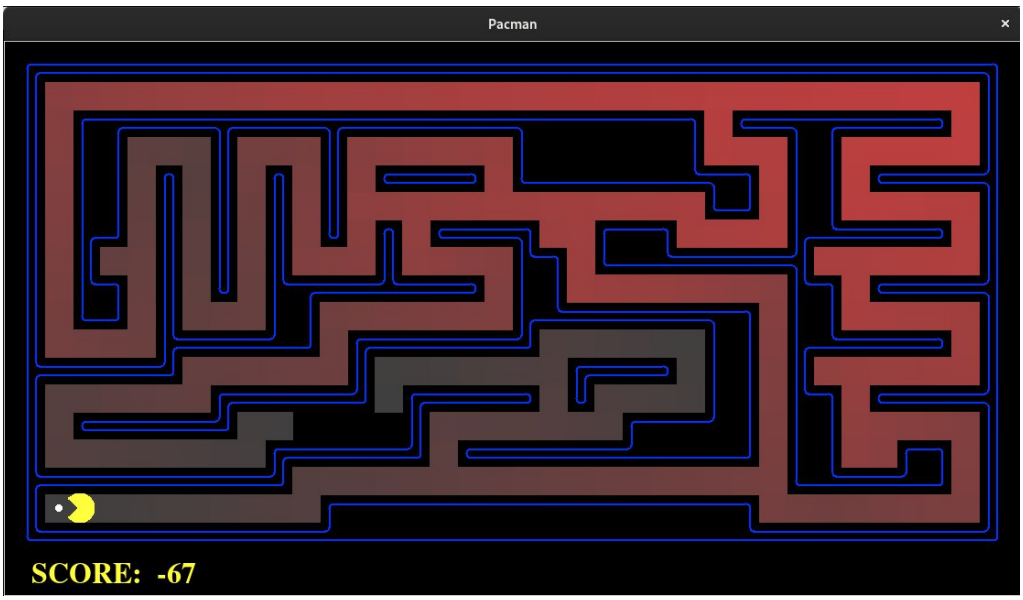

Рисунок 7 – Поиск в ширину в среднем лабиринте

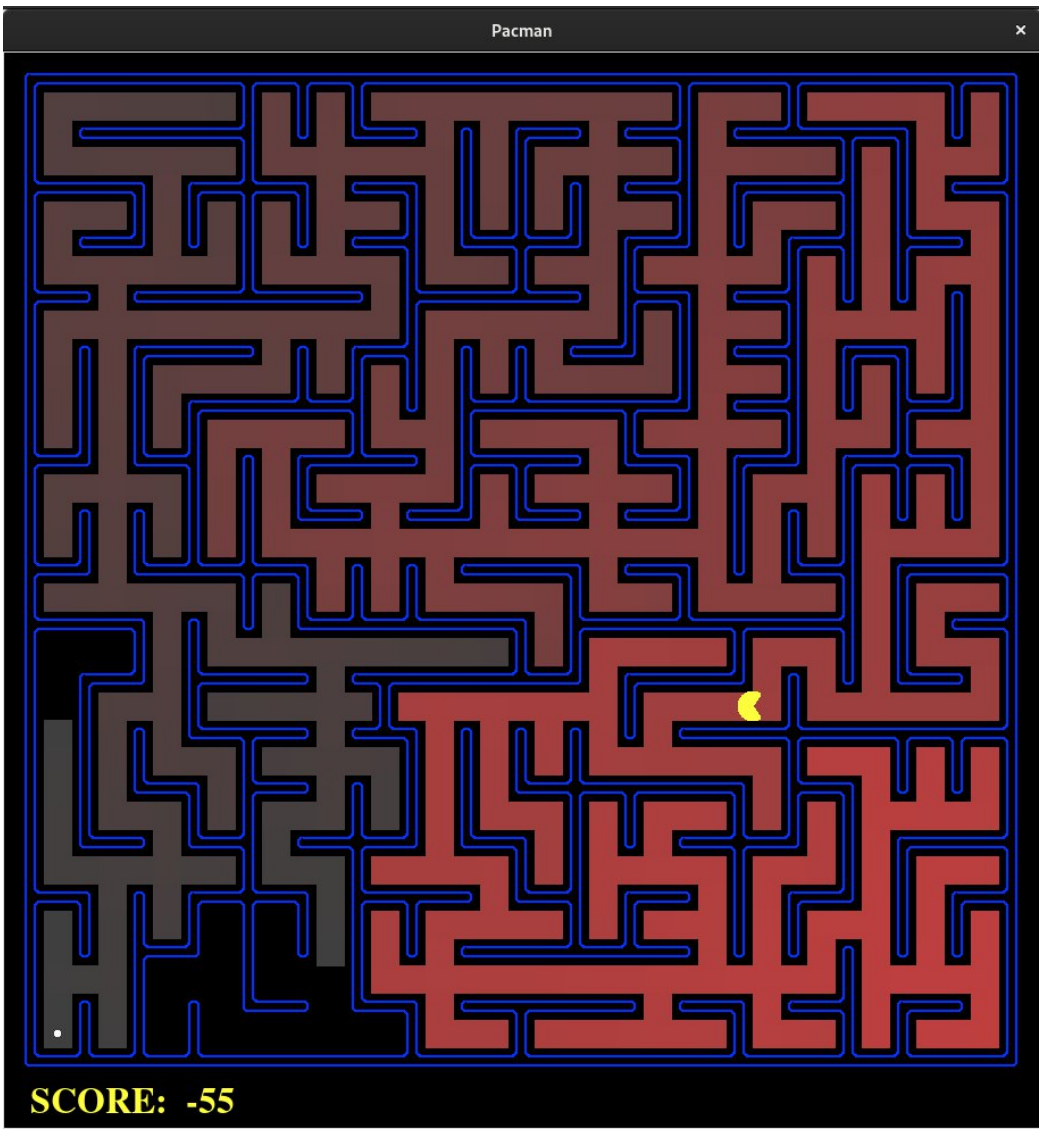

Рисунок 7 – Поиск в ширину в большом лабиринте

После чего написанный алгоритм был проверен для прохождения «игры в восемь». На рисунке 8 представлен представлен последний ход игры, соответственно можно сделать вывод, что данный алгоритм подходит и для решения данной игры.

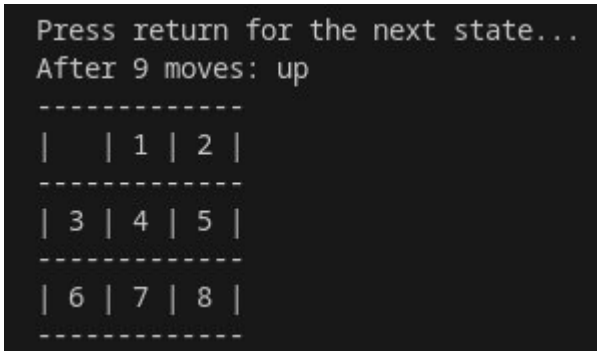

Рисунок 8 – Результат прохождения игры в восемь

Затем данный алгоритм был проверен при помощи автооценивателя, на рисунке 9 представлен результат успешного прохождения тестов.

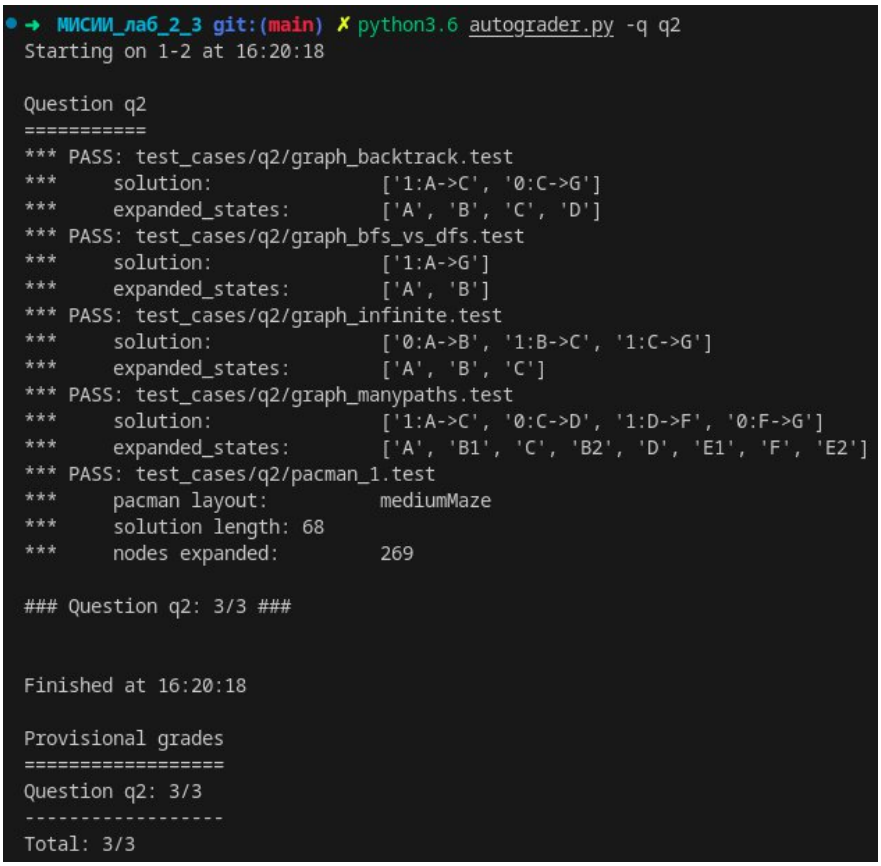

Рисунок 9 – Прохождение тестов при помощи автооценивателя

3. Поиск на основе алгоритма равных цен.

Был реализован алгоритм равных цен для поиска пути на графе, код которого представлен в листинге 3.

Листинг 3 – Алгоритм равных цен

```
def uniformCostSearch(problem):
    """Находит узел минимальной стоимости """
    "*** ВСТАВЬТЕ ВАШ КОД СЮДА ***"

    frontier = util.PriorityQueue()
    visited = {}

    start = problem.getStartState()
    startNode = (start, [], 0)

    frontier.push(startNode, 0)

    while not frontier.isEmpty():
        currentState, actions, currentCost = frontier.pop()

      if (currentState not in visited) or (currentCost < visited[currentState]):
            visited[currentState] = currentCost

            if problem.isGoalState(currentState):
                return actions
            else:
                successors = problem.getSuccessors(currentState)

                for sState, sAction, sCost in successors:
                    newAction = actions + [sAction]
                    newCost = currentCost + sCost
                    newNode = (sState, newAction, newCost)

                    frontier.update(newNode, newCost)

    return actions
```

Далее, данный алгоритм был протестирован на трех лабиринтах, где все агенты являются агентами, функционирующими на основе алгоритма UCS, которые отличаются только используемой функцией стоимости. Для этого были выполнены следующие команды:

python3.6 pacman.py -l mediumMaze -p SearchAgent -a fn=ucs

python3.6 pacman.py -l mediumDottedMaze -p StayEastSearchAgent

python3.6 pacman.py -l mediumScaryMaze -p StayWestSearchAgent

Результат продемонстрирован на рисунках 10, 11 и 12.

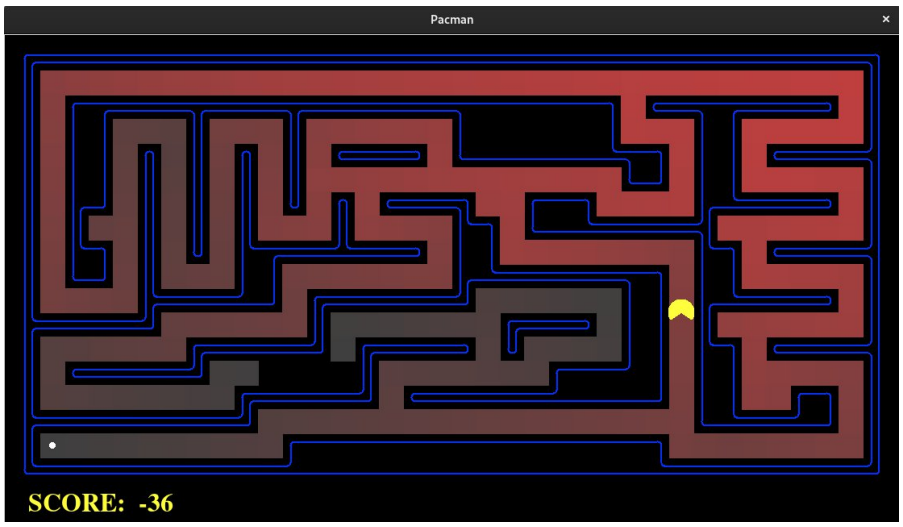

Рисунок 10 – Поиск на основе алгоритма равных цен в среднем лабиринте

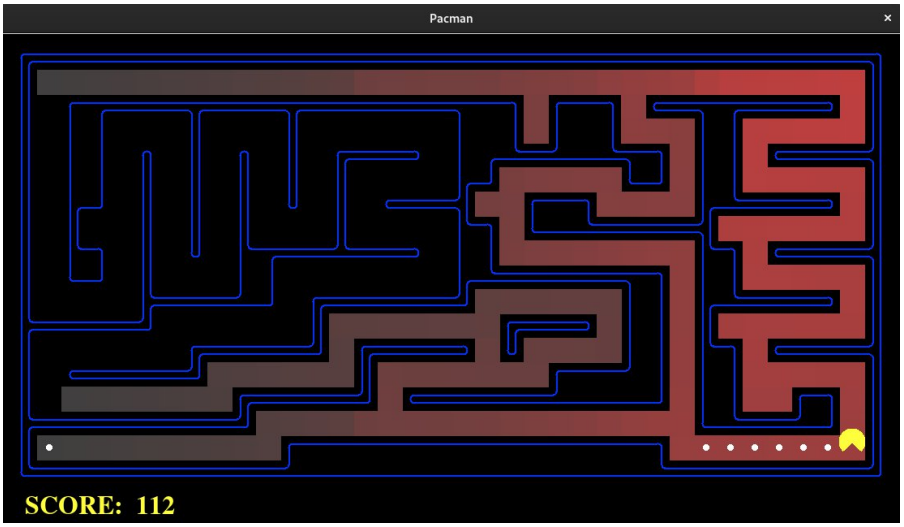

Рисунок 11 – Поиск в лабиринте с гранулами

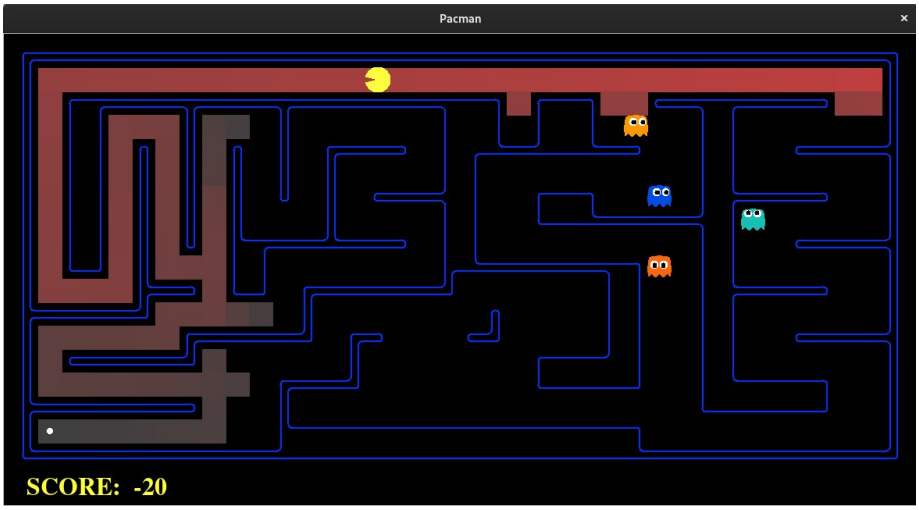

Рисунок 12 – Поиск в лабиринте с приведениями

После чего написанное решение было проверено при помощи автооценивателя. Результат успешного прохождения всех тестов продемонстрирован на рисунке 13.

```
Question q3
===========
*** PASS: test_cases/q3/graph_backtrack.test
***     solution:               ['1:A->C', '0:C->G']
***     expanded_states:        ['A', 'B', 'C', 'D']
*** PASS: test_cases/q3/graph_bfs_vs_dfs.test
***     solution:               ['1:A->G']
***     expanded_states:        ['A', 'B']
*** PASS: test_cases/q3/graph_infinite.test
***     solution:               ['0:A->B', '1:B->C', '1:C->G']
***     expanded_states:        ['A', 'B', 'C']
*** PASS: test_cases/q3/graph_manypaths.test
***     solution:               ['1:A->C', '0:C->D', '1:D->F', '0:F->G']
***     expanded_states:        ['A', 'B1', 'C', 'B2', 'D', 'E1', 'F', 'E2']
*** PASS: test_cases/q3/ucs_0_graph.test
***     solution:               ['Right', 'Down', 'Down']
***     expanded_states:        ['A', 'B', 'D', 'C', 'G']
*** PASS: test_cases/q3/ucs_1_problemC.test
***     pacman layout:          mediumMaze
***     solution length: 68
***     nodes expanded:         269
*** PASS: test_cases/q3/ucs_2_problemE.test
***     pacman layout:          mediumMaze
***     solution length: 74
***     nodes expanded:         260
*** PASS: test_cases/q3/ucs_3_problemW.test
***     pacman layout:          mediumMaze
***     solution length: 152
***     nodes expanded:         173
*** PASS: test_cases/q3/ucs_4_testSearch.test
***     pacman layout:          testSearch
***     solution length: 7
***     nodes expanded:         14
*** PASS: test_cases/q3/ucs_5_goalAtDequeue.test
***     solution:               ['1:A->B', '0:B->C', '0:C->G']
***     expanded_states:        ['A', 'B', 'C']

### Question q3: 3/3 ###


Finished at 17:38:13

Provisional grades
==================
Question q3: 3/3
------------------
Total: 3/3
```

Рисунок 13 – Прохождение тестов при помощи автооценивателя

Выводы

В ходе выполнения лабораторной работы были исследованы неинформированные методы поиска решений задач в пространстве состояний, приобретены навыки программирования интеллектуальных агентов, планирующих действия на основе методов слепого поиска решений задач. Также были разработаны алгоритмы для различных видов поиска путей, такие как поиск в глубину, в ширину, а также поиска на основе алгоритма равных цен. Все разработанные алгоритмы были проверены автооценивателем и ошибок выявлено не было.