Институт приборостроения, автоматизации и информационных технологий

Кафедра информационных систем и цифровых технологий

Дисциплина «Качество и тестирование программного обеспечения»

Отчет к лабораторной работе № 2

«Структурное тестирование программного обеспечения»

Выполнил:

Василения Иван Валерьевич

Принял:

Олькина Елена Викторовна

Орёл, 2025г

* Кофеварка должна уметь варить кофе
* Что подразумевается под «уметь»?
* Какие конкретно функции должна выполнять кофеварка для получения сваренного кофе?
* С какой формой кофе будет работать кофеварка (цельнозерновой, молотый, кофейные капсулы)?
* Кофеварка должна иметь кнопку включения, ручку и крышку над емкостью для наливания воды
* Каков должен быть объем емкости для воды?
* Куда должен помещаться кофе?
* Кофеварка должна работать от электричества
* Какие требования выдвигаются к потреблению энергии?
* Какой вид тока питает кофеварку?
* Кнопка должна включаться, только если крышка закрыта
* Кнопка должна включаться?
* Воду в кофеварку можно наливать, только если крышка закрыта
* Я так не думаю.
* Кофеварка должен поддерживать протокол HTCPCP ( RFC 2324). Заказчик особенно настаивает на этом требовании и отказывается его убирать.
* Ему стоило бы передумать, так как этот протокол – первоапрельская шутка. Не существует ни одной его реализации.
* Кофеварка должна быть красивой
* Требование субъективно, что понимается под красивой?

M = E − N + 2P,

где:

M = цикломатическая сложность,

E = количество рёбер в графе,

N = количество узлов в графе,

P = количество компонент связности.

def heuristic(a, b):

return sqrt((a.x - b.x)\*\*2 + (a.y - b.y)\*\*2)

def reconstruct\_path(node):

path = []

while node is not None:

path.append((node.x, node.y))

node = node.parent

return path[::-1]

def astar(graph, start\_node, end\_node):

open\_set = []

closed\_set = set()

if(start\_node is None or end\_node is None): return None

start\_node.g = 0

start\_node.h = heuristic(start\_node, end\_node)

start\_node.f = start\_node.g + start\_node.h

heapq.heappush(open\_set, (start\_node.f, start\_node))

while open\_set:

current\_f, current\_node = heapq.heappop(open\_set)

if current\_node == end\_node:

return reconstruct\_path(current\_node)

closed\_set.add(current\_node)

for neighbor in graph.get(current\_node, []):

if neighbor in closed\_set:

continue

tentative\_g = current\_node.g + heuristic(current\_node, neighbor)

if tentative\_g < neighbor.g:

neighbor.g = tentative\_g

neighbor.h = heuristic(neighbor, end\_node)

neighbor.f = neighbor.g + neighbor.h

neighbor.parent = current\_node

if neighbor not in open\_set:

heapq.heappush(open\_set, (neighbor.f, neighbor))

return None