Отчёт к заданию 2 “Поиск выхода из лабиринта” Гутов 428Б

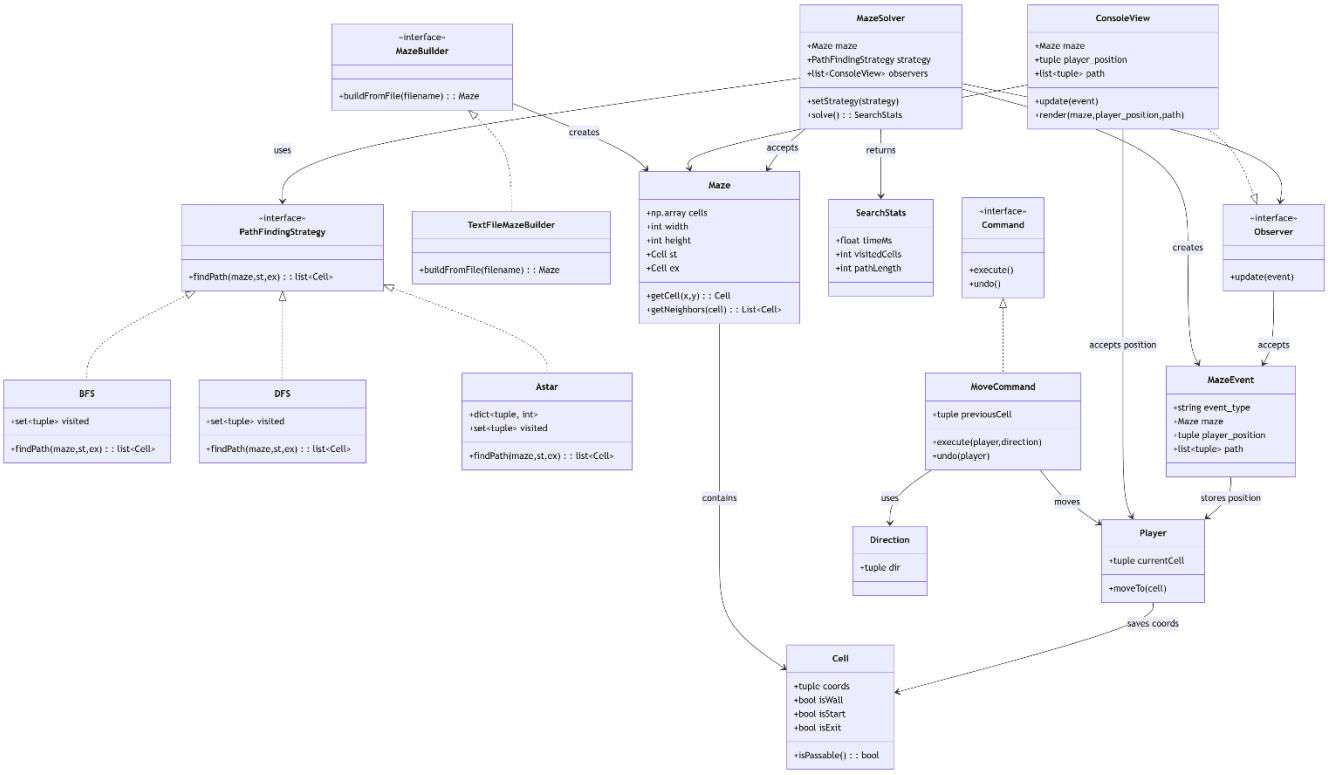
Ход работы

Было реализовано приложение, позволяющее принимать лабиринты из текстовых файлов в формате txt и автоматически находить в них маршрут от условленного старта до выхода разными стратегиями (алгоритмами) (BFS, DFS и A*). В приложение также были включены система графического отображения лабиринта и маршрута (в консоли) и функция ручного пошагового перемещения по лабиринту для более наглядной демонстрации работы программы. Функционал стратегий был проверен путём повторных экспериментов с различными лабиринтами, построенными при помощи ии. Вся структура программы была реализована в объектно – ориентированной парадигме.

Использованные паттерны

Полиморфизм, Наследование; GoF: Strategy, Command, Observer, Builder; Data Transfer Object (DTO).

Mermaid диаграмма



Выборочные листинги ключевых классов

```python
#Классы клетки и лабиринта

class cell:

    def __init__(self, coords, isWall = False, isStart = False,
                 isExit = False):
        self.coords = coords
        self.isWall = isWall
        self.isStart = isStart
        self.isExit = isExit

    def isPassable(self):
        if self.isWall:
            return False
        return True

```

```python
class Maze:

    def __init__(self, cells, width, height, st, ex):
        self.cells = cells
        self.width = width
        self.height = height
        self.st = st
        self.ex = ex

    def getCell(self,x,y):
        try:
            return self.cells[x][y]
        except:
            return None

    def getNeighbors(self,cell):
        x,y = cell.coords
        res = []
        for i,j in (x,y+1),(x,y-1),(x-1,y),(x+1,y):
            cellij = self.getCell(i,j)
            if i <= self.width-1 and j <= self.height-1 and 0 <= i and 0 <= j and cellij is not None:
                if cellij.isPassable():
                    res.append(cellij)
                else:
                    res.append(None)
            else:
                res.append(None)

        return res

```

```
class MazeBuilder(abc.ABC):

    @abc.abstractmethod
    def buildFromFile(filename): pass

#Наследуем от него класс постройки из текстового файла

class TextFileMazeBuilder(MazeBuilder):

    def buildFromFile(filename):
        with open(filename, "r") as file:

            rows = file.read().splitlines()

            #print(rows)

            width = 0
            height = 0
            for row in rows:
                height += 1
                if len(row) > width:
                    width = len(row)

            #print(width, height)

            st = (0,0)
            ex = (width,height)

            cells = np.full((width,height), None, dtype=object)

            flagst = False
            flagex = False
            for y in range(height):
                for x in range(width):
                    isWall = False
                    isStart = False
                    isExit = False

                    if rows[-(y+1)][x] == '#':
                        isWall = True
                    elif rows[-(y+1)][x] == 'S':
                        isStart = True
                        st = (x,y)
                        flagst = True
                        #print('Старт в',x,y)
                    elif rows[-(y+1)][x] == 'E':
                        isExit = True
                        ex = (x,y)
                        flagex = True
                        #print('Выход в',x,y)
                    elif rows[-(y+1)][x] != ' ':
                        raise ValueError("Неверный формат лабиринта! Пожалуйста, используйте только символы #,S,E и пробелы")

                    cells[x][y] = Cell((x,y), isWall, isStart, isExit)

            if flagst and flagex:

                return Maze(cells, width, height, cells[st[0]][st[1]], cells[ex[0]][ex[1]])

            raise ValueError('В лабаиринте должны быть вход и выход (S и E)')
```

```python
#Интерфейс поиска пути

class PathFindingStrategy(abc.ABC):

    @abc.abstractmethod
    def findPath(self, maze, st, ex): pass

#Поиск в глубину

class DFS(PathFindingStrategy):

    def findPath(self,maze,st,ex):

        stack = [st]

        self.visited = {st.coords} #по координатам надёжнее, а то вдруг адрес изменится

        pathmap = {}

        while stack:
            cell = stack.pop()

            if cell.coords == ex.coords:

                #маршрут выстраивается в обратном порядке и разворачивается
                path = []
                while cell.coords != st.coords:
                    path.append(cell)
                    cell = pathmap[cell.coords]
                path.append(st)
                path = path[::-1]
                return path

            for n in maze.getNeighbors(cell):
                if n != None and n.coords not in self.visited:
                    self.visited.add(n.coords)
                    pathmap[n.coords] = cell
                    stack.append(n)

        return None
```

```python
#Класс статистики поиска пути и класс оркестратор

class SearchStats():

    def __init__(self, timeMs, visitedCells, pathLength):
        self.timeMs = timeMs
        self.visitedCells = visitedCells
        self.pathLength = pathLength

class MazeSolver():

    def __init__(self, maze, strategy):
        self.maze = maze
        self.strategy = strategy
        self.observers = [ConsoleView(maze)]

        for observer in self.observers:
            observer.update(MazeEvent('maze_loaded',maze,maze.st.coords))

    def setStrategy(self,strategy):
        self.strategy = strategy

    def solve(self):

        start = time.perf_counter()
        path = self.strategy.findPath(self.maze,self.maze.st,self.maze.ex)
        end = time.perf_counter()

        elapsed = end - start

        visitedCells = len(self.strategy.visited)

        if path is not None:
            pathLength = len(path)

            for observer in self.observers:
                observer.update(MazeEvent('path_found',self.maze,path[-1].coords,path))

        else:
            pathLength = 0
            for observer in self.observers:
                observer.update(MazeEvent('path_found',self.maze,None,path))

        return SearchStats(elapsed*1000, visitedCells, pathLength)
```

```python
#Класс для событий

class MazeEvent():

    def __init__(self,event_type, maze, player_position = None, path = []):
        if player_position is None:
            player_position = maze.st.coords
        self.event_type = event_type
        self.maze = maze
        self.player_position = player_position
        self.path = path

#Интерфейс наблюдатель

class Observer(abc.ABC):

    @abc.abstractmethod
    def update(self, event):

        if not isinstance(event, (str, MazeEvent)):
            raise TypeError('Только строки и обьекты события')

        elif isinstance(event, MazeEvent) and event.event_type not in ('path_found','move','maze_loaded'):
            raise TypeError('Только события "path_found","move","maze_loaded"')

#Класс консольного просмотра
```

```python
class ConsoleView(Observer):

    def __init__(self, maze, player_position = (0,0), path = []):

        self.maze = maze
        self.player_position = player_position
        self.path = path

    def update(self, event):

        super().update(event) #проверка через сам интерфейс

        if isinstance(event, str):
            print('')
            print(event+'\n')
            self.render(self.maze, self.player_position, self.path)

        else:
            print('')
            print(event.event_type+'\n')
            if event.player_position is not None:
                self.player_position = event.player_position
            if event.path is not None and event.path:
                self.path = event.path
            self.render(event.maze, self.player_position, self.path)

    def render(self, maze, player_position, path):

        os.system('cls' if os.name == 'nt' else 'clear')

        #из-за системы координат надо всё опять транспонировать

        res = []
        for row in maze.cells.T[::-1]:
            subres = []
            for cell in row:
                if cell.isWall:
                    subres += '#'
                elif cell.isStart:
                    subres += 'S'
                elif cell.isExit:
                    subres += 'E'
                else:
                    subres += ' '
            res.append(subres)

        for cell in path:
            x,y = cell.coords
            if res[-(y+1)][x] != 'S':
                res[-(y+1)][x] = '*'

        res[-(player_position[1]+1)][player_position[0]] = 'X'

        for row in res:
            print(''.join(row))
```

```python
#Класс направление

class Direction():

    def __init__(self, x,y):
        self.dir = (x,y)

```

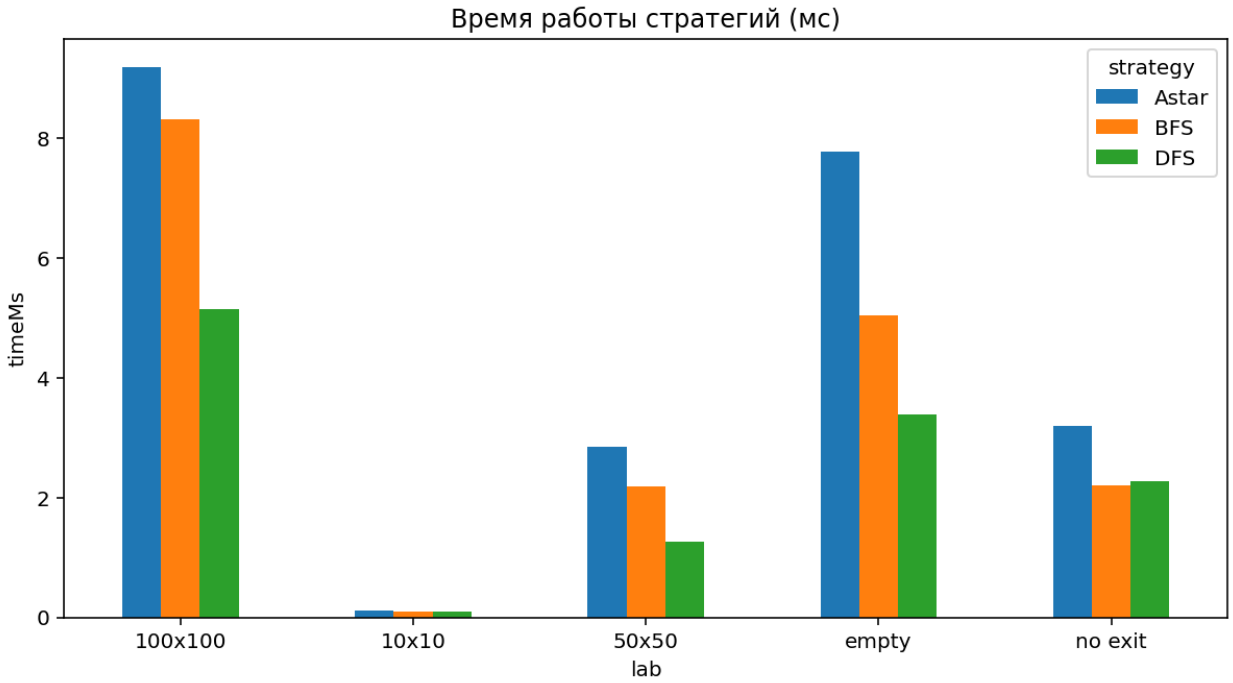
Результаты экспериментов

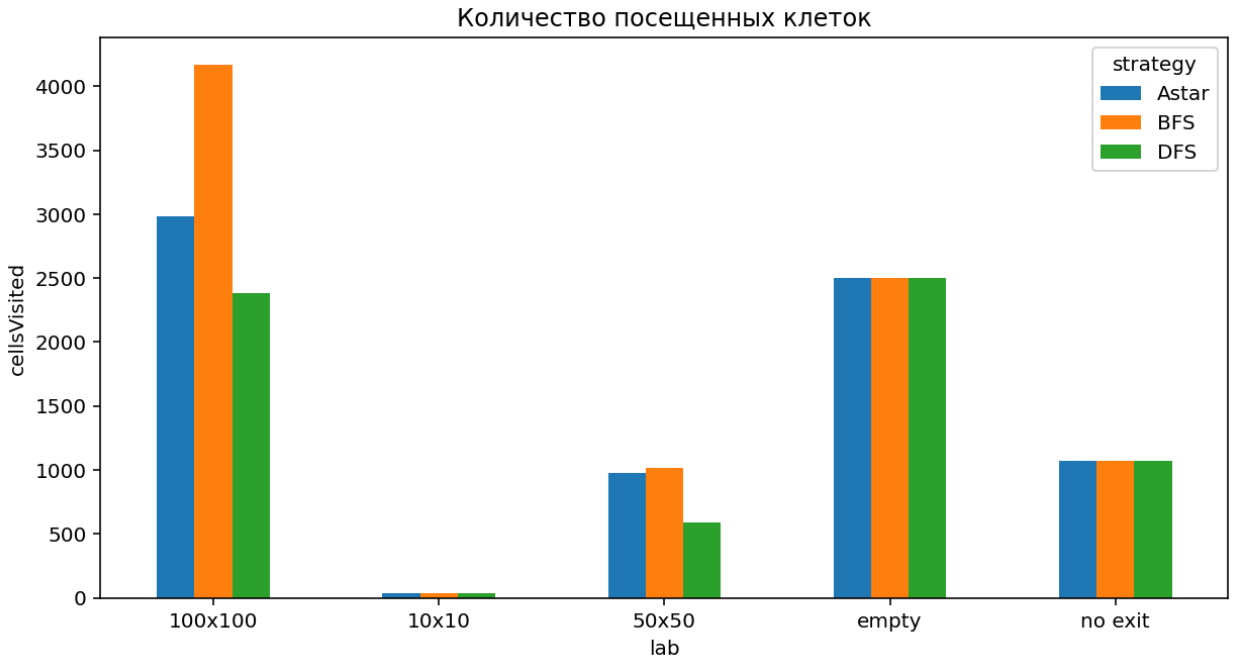
Таблица:

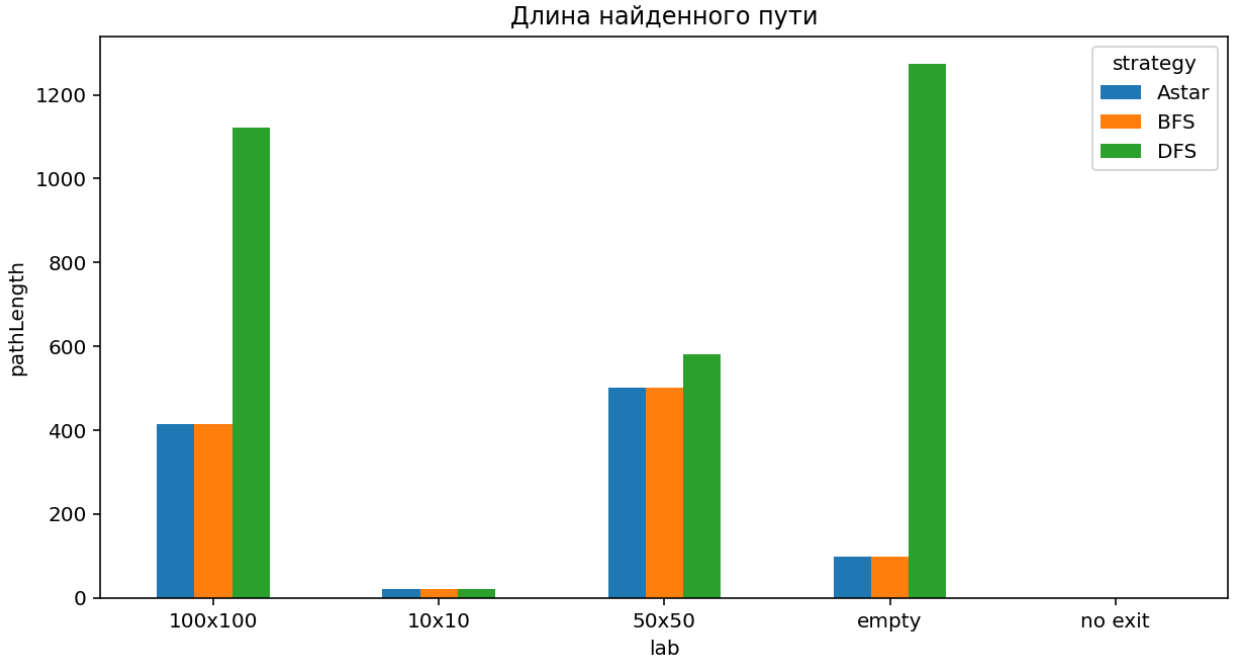
лабиринт	стратегия	время_мс	посещено_клеток (ср)	длина_пути (ср)
10x10	BFS	0.08966999594122171	36	21
50x50	BFS	2.187670022249222	1020	501
100x100	BFS	8.31503001973033	4172	414
empty	BFS	5.047210049815476	2500	99
no exit	BFS	2.197379991412163	1072	0
10x10	DFS	0.09059999138116837	37	21
50x50	DFS	1.264189975336194	593	581
100x100	DFS	5.151620041579008	2384	1122
empty	DFS	3.3813500544056296	2500	1275
no exit	DFS	2.2671299753710628	1072	0
10x10	Astar	0.11618996504694223	33	21
50x50	Astar	2.8455600142478943	973	501
100x100	Astar	9.197140019387007	2980	414
empty	Astar	7.771430024877191	2500	99
no exit	Astar	3.192879958078265	1072	0

Графики:

Примечание: размер лабиринтов empty и no exit – 50x50 клеток







Анализ эффективности алгоритмов и применимости паттернов

Алгоритмы:

По результатам экспериментов (всего было проведено 10 измерений для каждой комбинации лабиринт + алгоритм) видно, что алгоритм DFS (поиск в глубину) быстрее остальных справляется с задачей поиска пути (на всех лабиринтах за исключением безвыходного) и посещает меньше всего клеток в процессе, что означает, что данный алгоритм по минимуму расходует ресурсы компьютера. Однако же, что становится заметнее на более крупных

лабиринтах, маршрут, выстраиваемый DFS, имеет наибольшую длину среди маршрутов, построенных данными алгоритмами.
Вывод по DFS: Этот алгоритм можно использовать для нахождения маршрутов в небольших лабиринтах, или в лабиринтах, где конечная длина маршрута не важна. Он отлично подходит для устройств с ограниченной вычислительной мощностью, что, по-моему мнению, делает его гораздо полезнее, чем кажется.

Алгоритмы BFS (поиск в ширину) и Astar (комбинированный, с эвристикой) оба в конечном итоге достигали одинаковой, оптимальной длины пути, меньше чем у DFS. Причём, BFS оказался быстрее Astar, посещая большее количество клеток. Это делает BFS объективно более эффективным чем DFS и Astar на среднего размера лабиринтах (предположу, что в диапазоне 100x100 – 500x500), однако, как заметно на графиках, чем больше размер лабиринта, тем ближе время Astar к DFS по времени и количеству посещённых клеток, в то время как у BFS оба эти параметра растут сравнительно быстро. Таким образом, на лабиринтах размерами где-то 5000x5000 и больше явно эффективнее было бы использовать Astar.

Вывод по BFS:
Алгоритм более затратный чем DFS, но на не слишком больших лабиринтах может сравниться с ним по скорости, при этом давая гораздо более оптимальный маршрут.

Вывод по Astar:
На небольших лабиринтах он незначительно проигрывает по времени и затратам вычислительных ресурсов остальным алгоритмам, но его эффективность остаётся высокой даже при самых больших лабиринтах, что делает этот алгоритм идеальным балансом между скоростью и точностью вычисления маршрута.

Паттерны:

Полиморфизм

Благодаря паттерну полиморфизма можно сначала указать через интерфейс предполагаемую функциональность класса, а затем уже реализовать её внутри класса. Это также позволяет в будущем добавить в программу больше классов-наследников схожих функций. Полезно для соблюдения целостности структуры программы. Один из основных паттернов, применимых везде.

Например, класс TextFileMazeBuilder наследуется от интерфейса MazeBuilder. Функция создания лабиринта из файла buildFromFile(filename) уже указана, но не реализована в самом интерфейсе. TextFileMazeBuilder как наследник интерфейса обязан предоставить реализацию функции, и реализует он её,

соответственно, через обработку текстовых файлов, при этом можно также добавить в программу, например, ImageMazeBuilder, наследник того же интерфейса, который будет реализовывать функцию создания лабиринта по-своему, на этот раз из изображений, а не из текста.

Наследование

Этот паттерн позволяет указать общую функциональность и какие параметры принимают функции наследников (не всегда) абстрактного класса. Полезно для избежания излишних повторов в коде и соблюдения целостности структуры программы. Также основной паттерн.

Например, в интерфейсе Observer метод update(event) не содержит полной функциональности, но уже заранее содержит проверку на ошибки в подаче входных параметров, чтобы не тратить время на прописывание одного и того же в будущих наследниках интерфейса несколько раз. Наследник ConsoleView добавляет к функции update свой функционал для отрисовки лабиринта, пути и позиции игрока в консоли, но также и вызывает функционал класса родителя командой super().update(event)

Strategy

Данный паттерн позволяет избежать излишнего использования операторов if и else посредством разделения функций алгоритма на части отдельной иерархии, выполняющейся исходя из контекста. Экономит вычислительные ресурсы. Демонстрация его работы доказывает, он применим в разного рода коммерческих проектах, а его наглядность помогает в работе как программистам, так и заказчикам.

Пример:

Интерфейс PathFindingStrategy имеет 3 реализации – BFS, DFS и Astar. Все они имеют общую функцию findPath(), и MazeSolver, вызывая эту функцию у своей strategy, не смотрит, какая реализация задействована. Для него это не имеет значения.

Command

Этот паттерн заключается в инкапсулировании запроса в виде объекта. Это очень полезно для эффективного логирования результатов работы программы. Применимо в приложениях со встроенными интерфейсами взаимодействия пользователя с программой.

Пример реализации этого паттерна – система ручного пошагового передвижения пользователя по лабиринту в программе. Конкретнее, существует интерфейс Command с функциями execute() и undo(), и его наследник - класс MoveCommand, который осуществляет непосредственное перемещение игрока. Класс принимает на вход execute() объект игрока и объект направления перемещения, а при самом процессе выполнения команды запоминает (логирует) предыдущее местоположение для отмены действия. В undo() принимается только объект игрока, и его местоположение меняется на предыдущее.

Observer

Данный паттерн предполагает создание механизма подписки. Проще говоря, одни объекты следят за событиями в других объектах. Паттерн позволяет чётко разделить функциональность классов, помогает целостности структуры кода и читабельности, в какой-то степени тоже экономит вычислительные ресурсы. Также очень применимый паттерн в работе с пользовательскими интерфейсами.

Пример: Класс MazeSolver содержит список наблюдателей observers, которых он уведомляет о событиях (по типу “лабиринт загружен” / “путь найден”), передавая их в формате MazeEvent. Он не знает о том, что происходит с этими событиями после передачи наблюдателям. На данный момент единственный реализованный наблюдатель, ConsoleView, принимает события MazeEvent и обрабатывает их (выводит в консоль новое изображение лабиринта в формате render).

Builder

Паттерн, позволяющий разделить создание сложного объекта между простыми, упрощая читабельность кода и положительно влияя на целостность его структуры. Применим в проектах, предусматривающих сложную и часто вариативную реализацию тех или иных алгоритмов.

Пример в коде: подкласс TextFileMazeBuilder интерфейса MazeBuilder, реализующий всю функцию создания объекта лабиринта из текста внутри себя.

Data Transfer Object (DTO)

Паттерн, предусматривающий объединение разнородной информации в один объект (структуру) для передачи между процессами. Это упрощает работу с информацией и конвертирование её в разные форматы. Применим в сферах связанных с работой с большими данными.

Пример: объект SearchStats, возвращаемый после операции solve() классом MazeSolver. Этот объект используется для переноса результатов экспериментов с лабиринтами в формат csv.

Вывод

ООП и использованные паттерны помогли наладить множество мелочей в коде. Прежде всего, код остался читаемым даже 600 строк спустя:

Паттерны, особенно GoF (в данном случае Strategy, Command, Observer и Builder) представили содержимое программы в наглядной и легко модифицируемой по частям форме. Самое главное, изменения в программу можно вносить динамически, и добавление новых классов для расширения функциональности, например, ImageMazeBuilder, не вызовет никаких проблем в программе.

Полноценное ведение записей измерений программой также возможно благодаря связной работе паттернов. В данном случае самый полезный – DTO.

ООП даёт программе ту самую гибкость которой не достаёт спагетти-коду.