

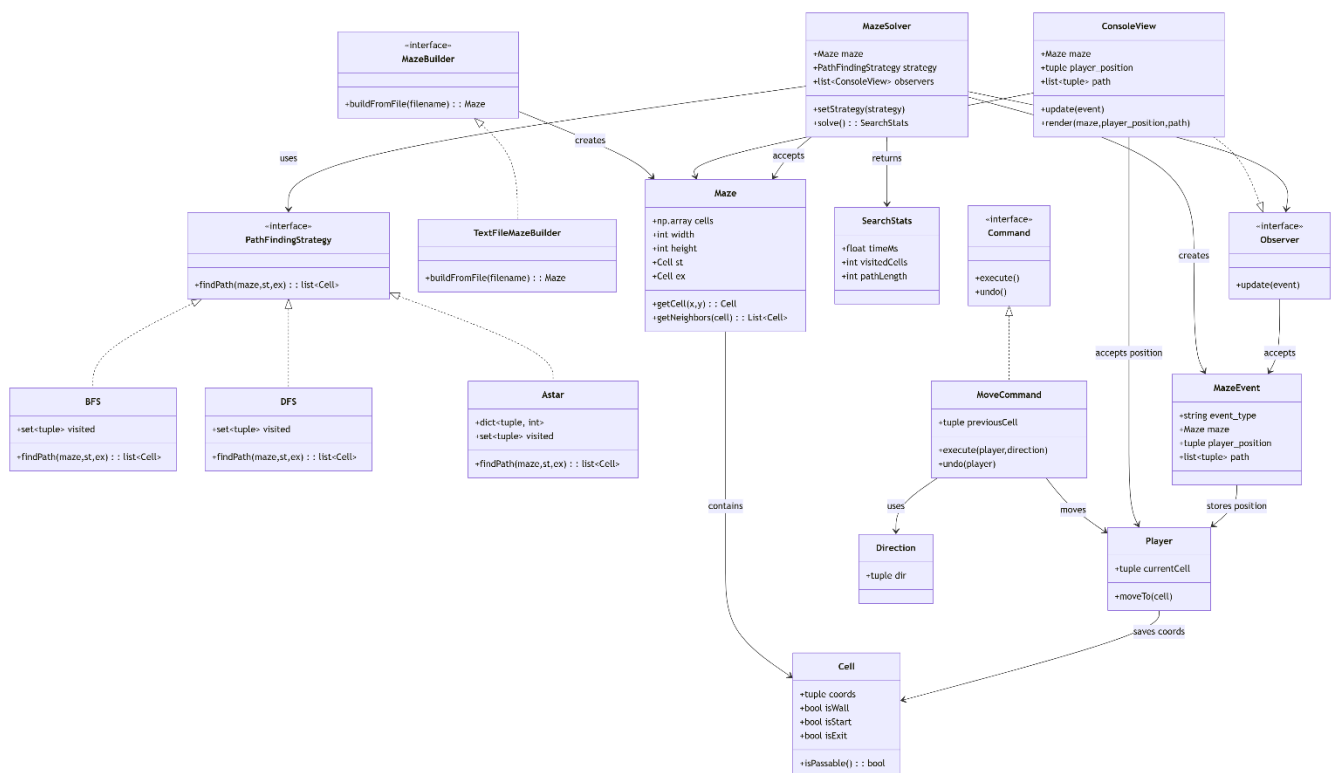
Ход работы

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

```
10 #Классы клетки и лабиринта
11
12 class Cell:
13
14     def __init__(self, coords, isWall = False, isStart = False,
15                 isExit = False):
16         self.coords = coords
17         self.isWall = isWall
18         self.isStart = isStart
19         self.isExit = isExit
20
21     def isPassable(self):
22         if self.isWall:
23             return False
24         return True
25
```

```
26 class Maze:
27
28     def __init__(self, cells, width, height, st, ex):
29         self.cells = cells
30         self.width = width
31         self.height = height
32         self.st = st
33         self.ex = ex
34
35     def getCell(self, x, y):
36         try:
37             return self.cells[x][y]
38         except:
39             return None
40
41     def getNeighbors(self, cell):
42         x, y = cell.coords
43         res = []
44         for i, j in (x, y+1), (x, y-1), (x-1, y), (x+1, y):
45             cellij = self.getCell(i, j)
46             if i <= self.width-1 and j <= self.height-1 and 0 <= i and 0 <= j and cellij is not None:
47                 if cellij.isPassable():
48                     res.append(cellij)
49                 else:
50                     res.append(None)
51             else:
52                 res.append(None)
53
54         return res
55
```

```

88 class MazeBuilder(abc.ABC):
89
90     @abc.abstractmethod
91     def buildFromFile(filename): pass
92
93     #Наследуем от него класс постройки из текстового файла
94
95     class TextFileMazeBuilder(MazeBuilder):
96
97         def buildFromFile(filename):
98             with open(filename, "r") as file:
99
100                 rows = file.read().splitlines()
101
102                 #print(rows)
103
104                 width = 0
105                 height = 0
106                 for row in rows:
107                     height += 1
108                     if len(row) > width:
109                         width = len(row)
110
111                 #print(width, height)
112
113                 st = (0,0)
114                 ex = (width,height)
115
116                 cells = np.full((width,height), None, dtype=object)
117
118                 flagst = False
119                 flagex = False
120                 for y in range(height):
121                     for x in range(width):
122                         isWall = False
123                         isStart = False
124                         isExit = False
125
126                         if rows[-(y+1)][x] == '#':
127                             isWall = True
128                         elif rows[-(y+1)][x] == 'S':
129                             isStart = True
130                             st = (x,y)
131                             flagst = True
132                             #print('Старт в',x,y)
133                         elif rows[-(y+1)][x] == 'E':
134                             isExit = True
135                             ex = (x,y)
136                             flagex = True
137                             #print('Выход в',x,y)
138                         elif rows[-(y+1)][x] != ' ':
139                             raise ValueError("Неверный формат лабиринта! Пожалуйста, используйте только символы #,S,E и пробелы")
140
141                 cells[x][y] = Cell((x,y), isWall, isStart, isExit)
142
143                 if flagst and flagex:
144
145                     return Maze(cells, width, height, cells[st[0]][st[1]], cells[ex[0]][ex[1]])
146
147                 raise ValueError('В лабиринте должны быть вход и выход (S и E)')

```

```

157 #Интерфейс поиска пути
158
159 class PathFindingStrategy(abc.ABC):
160
161     @abc.abstractmethod
162     def findPath(self, maze, st, ex): pass
163
164 #Поиск в глубину
165
166 class DFS(PathFindingStrategy):
167
168     def findPath(self, maze, st, ex):
169
170         stack = [st]
171
172         self.visited = {st.coords} #по координатам надёжнее, а то вдруг адрес изменится
173
174         pathmap = {}
175
176         while stack:
177             cell = stack.pop()
178
179             if cell.coords == ex.coords:
180
181                 #маршрут выстраивается в обратном порядке и разворачивается
182                 path = []
183                 while cell.coords != st.coords:
184                     path.append(cell)
185                     cell = pathmap[cell.coords]
186                 path.append(st)
187                 path = path[::-1]
188                 return path
189
190             for n in maze.getNeighbors(cell):
191                 if n != None and n.coords not in self.visited:
192                     self.visited.add(n.coords)
193                     pathmap[n.coords] = cell
194                     stack.append(n)
195
196         return None

```

```

299 #Класс статистики поиска пути и класс оркестратор
300
301 class SearchStats():
302
303     def __init__(self, timeMs, visitedCells, pathLength):
304         self.timeMs = timeMs
305         self.visitedCells = visitedCells
306         self.pathLength = pathLength
307
308 class MazeSolver():
309
310     def __init__(self, maze, strategy):
311         self.maze = maze
312         self.strategy = strategy
313         self.observers = [ConsoleView(maze)]
314
315         for observer in self.observers:
316             observer.update(MazeEvent('maze_loaded', maze, maze.st.coords))
317
318     def setStrategy(self, strategy):
319         self.strategy = strategy
320
321     def solve(self):
322
323         start = time.perf_counter()
324         path = self.strategy.findPath(self.maze, self.maze.st, self.maze.ex)
325         end = time.perf_counter()
326
327         elapsed = end - start
328
329         visitedCells = len(self.strategy.visited)
330
331         if path is not None:
332             pathLength = len(path)
333
334             for observer in self.observers:
335                 observer.update(MazeEvent('path_found', self.maze, path[-1].coords, path))
336
337         else:
338             pathLength = 0
339             for observer in self.observers:
340                 observer.update(MazeEvent('path_found', self.maze, None, path))
341
342         return SearchStats(elapsed*1000, visitedCells, pathLength)
343
344 #Класс для событий
345
346 class MazeEvent():
347
348     def __init__(self, event_type, maze, player_position = None, path = []):
349         if player_position is None:
350             player_position = maze.st.coords
351         self.event_type = event_type
352         self.maze = maze
353         self.player_position = player_position
354         self.path = path
355
356 #Интерфейс наблюдатель
357
358 class Observer(abc.ABC):
359
360     @abc.abstractmethod
361     def update(self, event):
362
363         if not isinstance(event, (str, MazeEvent)):
364             raise TypeError('Только строки и объекты события')
365
366         elif isinstance(event, MazeEvent) and event.event_type not in ('path_found', 'move', 'maze_loaded'):
367             raise TypeError('Только события "path_found", "move", "maze_loaded"')
368
369 #Класс консольного просмотра

```

```

390 class ConsoleView(Observer):
391
392     def __init__(self, maze, player_position = (0,0), path = []):
393
394         self.maze = maze
395         self.player_position = player_position
396         self.path = path
397
398     def update(self, event):
399
400         super().update(event) #проверка через сам интерфейс
401
402         if isinstance(event, str):
403             print('')
404             print(event+'\n')
405             self.render(self.maze, self.player_position, self.path)
406
407         else:
408             print('')
409             print(event.event_type+'\n')
410             if event.player_position is not None:
411                 self.player_position = event.player_position
412             if event.path is not None and event.path:
413                 self.path = event.path
414                 self.render(event.maze, self.player_position, self.path)
415
416     def render(self, maze, player_position, path):
417
418         os.system('cls' if os.name == 'nt' else 'clear')
419
420         #из-за системы координат надо всё опять транспонировать
421
422         res = []
423         for row in maze.cells.T[::-1]:
424             subres = []
425             for cell in row:
426                 if cell.isWall:
427                     subres += '#'
428                 elif cell.isStart:
429                     subres += 'S'
430                 elif cell.isExit:
431                     subres += 'E'
432                 else:
433                     subres += ' '
434             res.append(subres)
435
436         for cell in path:
437             x,y = cell.coords
438             if res[-(y+1)][x] != 'S':
439                 res[-(y+1)][x] = '*'
440
441         res[-(player_position[1]+1)][player_position[0]] = 'X'
442
443         for row in res:
444             print(''.join(row))
445
446         #Класс направление
447
448     class Direction():
449
450     def __init__(self, x,y):
451         self.dir = (x,y)
452
453

```

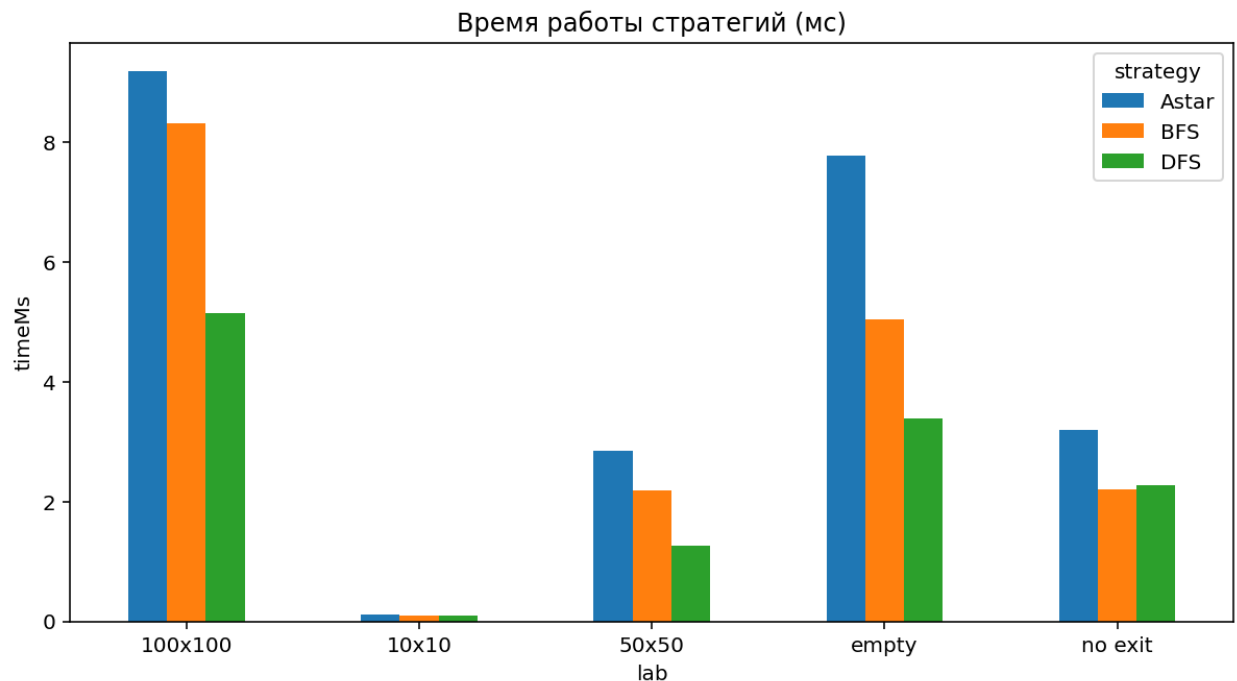
Результаты экспериментов

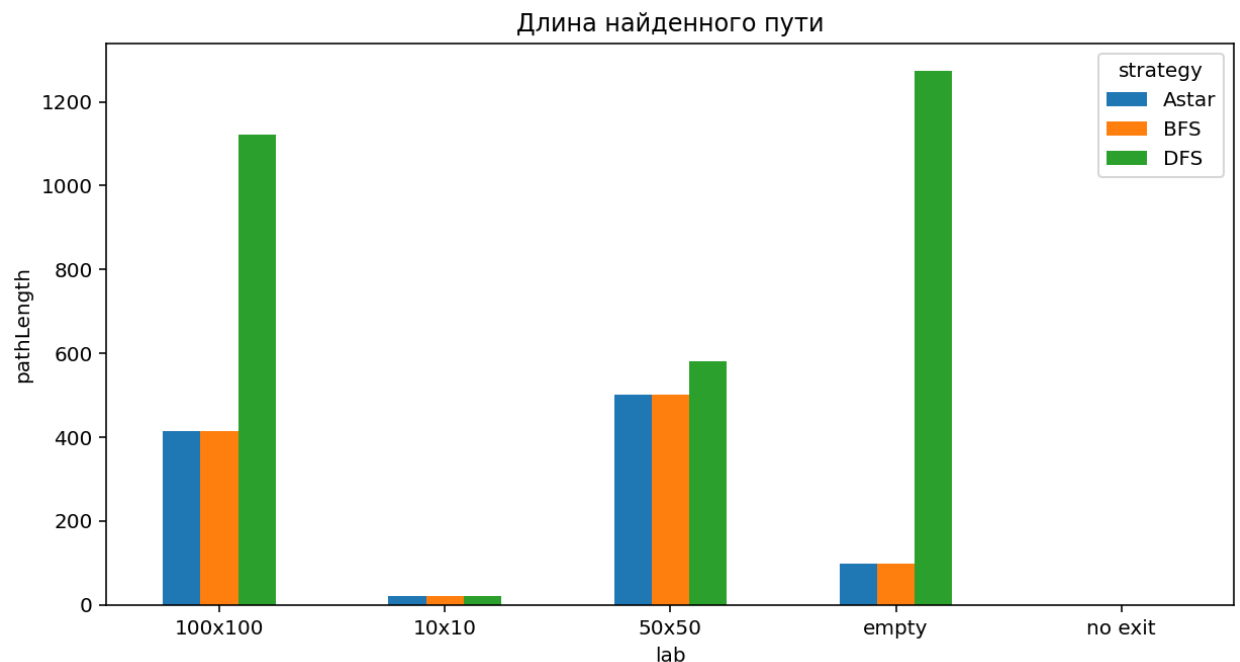
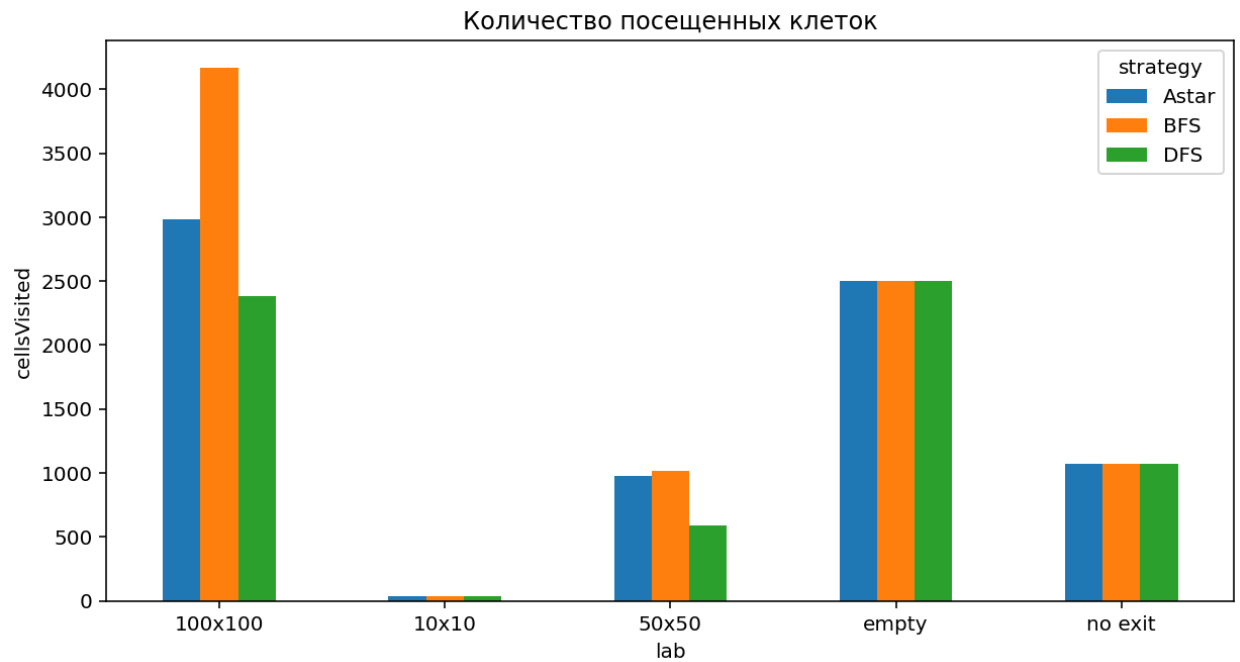
Таблица:

| 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.