



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени
Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Лабораторная работа № 4
по дисциплине
«Функциональное и логическое
программирование»

Тема Лабиринт

Студент Александров Э.И.

Группа ИУ7-53БВ

Преподаватель Строганов Ю.В.

Москва, 2024

Содержание

ВВЕДЕНИЕ	4
1 Аналитическая часть	5
2 Конструкторская часть	6
3 Технологическая часть	8
ЗАКЛЮЧЕНИЕ	11
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	12
Приложение А	13

ВВЕДЕНИЕ

Цель данной работы заключается в разработке программы, которая позволяет формировать карту лабиринта и находить выход из него. Задачи, поставленные перед программой, включают создание лабиринта с особыми клетками, такими как телепорты, счётные ловушки и стены, а также реализацию механики перемещения игрока по лабиринту. Задачи включают анализ предметной области, разработку алгоритма для решения лабиринта, его реализацию и тестирование.

1 Аналитическая часть

Лабиринты являются популярным элементом в играх и головоломках, предоставляя игрокам возможность исследовать и решать задачи. В данной работе мы рассматриваем лабиринт, состоящий из ячеек ($M \times N$ клеток), каждая из которых может иметь различные свойства. Основные элементы лабиринта включают:

- 1) Телепорты: позволяют игроку перемещаться мгновенно в другую точку лабиринта.
- 2) Счётные ловушки: активируются через определённое количество шагов, создавая дополнительные сложности для игрока.
- 3) Стены: препятствуют движению, добавляя элемент стратегии в прохождение лабиринта.
- 4) Прыжок: возможность перемещения через одну клетку на другую, находящуюся через две клетки от текущей.

Выход из лабиринта достигается через последовательное перемещение по ячейкам, с учётом их свойств.

Вывод

В аналитической части мы рассмотрели основные элементы лабиринта и их влияние на игровой процесс.

2 Конструкторская часть

В данной части отчёта мы подробно рассмотрим архитектуру программного обеспечения, алгоритмы, использованные для реализации программы, а также структуру данных, применяемую для представления лабиринта и его элементов.

1) Архитектура программы

Основные компоненты архитектуры включают:

- Модуль определения лабиринта: Этот модуль отвечает за создание и хранение структуры лабиринта. Лабиринт задаётся в виде двумерного массива ячеек, каждая из которых имеет свои свойства (например, пустая ячейка, стена, телепорт, счётная ловушка).
- Модуль перемещения: Этот модуль реализует логику перемещения игрока по лабиринту. Он определяет возможные направления движения и учитывает особенности ячеек, такие как стены и телепорты.
- Модуль проверки допустимости ячеек: Этот модуль отвечает за проверку, может ли игрок переместиться в определённую ячейку. Он проверяет, находится ли ячейка в пределах границ лабиринта и не является ли она стеной.
- Модуль поиска пути: Этот модуль реализует алгоритм поиска пути от начальной ячейки до конечной. Он использует рекурсивный подход для обхода ячеек и нахождения возможных путей, учитывая активные ловушки и телепорты.

2) Структура данных

Лабиринт представлен в виде списка ячеек, где каждая ячейка описывается с помощью структуры `cell(X, Y, Type)`, где:

- `X` и `Y` — координаты ячейки в лабиринте.
- `Type` — тип ячейки, который может быть одним из следующих:
 - `empty` — пустая ячейка.
 - `wall` — стена, через которую нельзя пройти
 - `teleport(X1, Y1)` — телепорт, который перемещает игрока в ячейку с координатами `(X1, Y1)`.
 - `counting_trap(K)` — счётная ловушка, которая срабатывает каждые `K` шагов.

3) Алгоритмы

- Определение возможных перемещений

Функция `move/2` определяет возможные перемещения игрока. Она реализует следующие направления:

- Вверх (уменьшение координаты `X`).
- Вниз (увеличение координаты `X`).
- Влево (уменьшение координаты `Y`).
- Вправо (увеличение координаты `Y`).

- Прыжки (перемещение на две клетки в любом направлении).
- Проверка допустимости ячейки.
Функция `valid_cell/1` проверяет, находится ли ячейка в пределах границ лабиринта и не является ли она стеной. Это позволяет избежать попыток перемещения в недоступные ячейки.
- Поиск пути
Функция `find_path/4` реализует рекурсивный поиск пути от начальной ячейки до конечной. Она принимает следующие параметры:
 - `Start` — начальная ячейка.
 - `End` — конечная ячейка.
 - `Visited` — список посещённых ячеек.
 - `K` — значение для счётной ловушки.

Алгоритм работает следующим образом:

- 1) Если текущая ячейка совпадает с конечной, путь найден, и функция возвращает список посещённых ячеек.
- 2) В противном случае, для каждой возможной ячейки, в которую можно переместиться, проверяется её допустимость.
- 3) Если ячейка является телепортом, игрок перемещается в указанную ячейку.
- 4) Если ячейка является счётной ловушкой, увеличивается счётчик шагов, и при достижении значения `K` выводится сообщение о срабатывании ловушки.
- 5) Рекурсивно вызывается функция для следующей ячейки.

Вывод

В конструкторской части мы подробно описали архитектуру программы, структуру данных и алгоритмы, используемые для реализации программы. Это позволяет понять, как программа организована и как она функционирует для решения поставленной задачи по нахождению выхода из лабиринта.

3 Технологическая часть

Программа была разработана с использованием языка программирования Prolog, который является языком логического программирования. Prolog идеально подходит для задач, связанных с искусственным интеллектом, обработкой естественного языка и решением логических задач, таких как поиск путей в лабиринтах.

Для разработки программы использовалась среда SWI-Prolog. Эта среда предоставляет мощные инструменты для написания, отладки и тестирования Prolog-кода.

Код программы:

```
% Определяем структуру лабиринта
% labyrinth(M, N, Cells).
% M: количество строк, N: количество столбцов,
% Cells: список определений ячеек
labyrinth(5, 5, [
cell(1, 1, empty),
cell(1, 2, wall),
cell(1, 3, empty),
cell(1, 4, empty),
cell(1, 5, empty),
cell(2, 1, teleport(3, 3)),
cell(2, 2, teleport(3, 3)),
cell(2, 3, empty),
cell(2, 4, wall),
cell(2, 5, empty),
cell(3, 1, teleport(3, 3)),
cell(3, 2, empty),
cell(3, 3, empty),
cell(3, 4, counting_trap(3)),
cell(3, 5, empty),
cell(4, 1, wall),
cell(4, 2, wall),
cell(4, 3, empty),
cell(4, 4, wall),
cell(4, 5, empty),
cell(5, 1, empty),
cell(5, 2, empty),
cell(5, 3, empty),
cell(5, 4, empty),
cell(5, 5, empty)
```

])).

% Определяем возможные ходы

```
move((X, Y), (X1, Y)) :- X1 is X + 1. % down
move((X, Y), (X1, Y)) :- X1 is X - 1. % up
move((X, Y), (X, Y1)) :- Y1 is Y + 1. % right
move((X, Y), (X, Y1)) :- Y1 is Y - 1. % left
move((X, Y), (X1, Y)) :- X1 is X + 2, Y = Y. % jump down
move((X, Y), (X1, Y)) :- X1 is X - 2, Y = Y. % jump up
move((X, Y), (X, Y1)) :- X = X, Y1 is Y + 2. % jump right
move((X, Y), (X, Y1)) :- X = X, Y1 is Y - 2. % jump left
```

% Проверьте, является ли ячейка допустимой

(находится ли она в пределах границ и не является стеной)

```
valid_cell((X, Y)) :-
    labyrinth(M, N, _),
    X > 0, X =< M,
    Y > 0, Y =< N,
    \+ cell((X, Y), wall).
```

% Получаем тип ячейки

```
cell((X, Y), Type) :-
    labyrinth(_, _, Cells),
    member(cell(X, Y, Type), Cells).
```

% Находим путь от Start до End

```
find_path(Start, End, Path, K) :-
    find_path(Start, End, [Start], Path, K, 0).
```

```
find_path(End, End, Visited, Path, _, _) :-
    reverse(Visited, Path).
```

```
find_path(Current, End, Visited, Path, K, Steps) :-
    move(Current, Next),
    valid_cell(Next),
    \+ member(Next, Visited),
    (    cell(Current, counting_trap(K)),
    NewSteps is Steps + 1,
    (    NewSteps >= K ->
```



```

write('Trap triggered!'), nl,
NewSteps1 is 0
;   NewSteps1 = NewSteps
)
;   NewSteps1 = Steps
),
(   cell(Current, teleport(X, Y)) ->
Next = (X, Y)
;   Next = Next
),
find_path(Next, End, [Next | Visited], Path, K, NewSteps1).

```

Пример теста программы:

```
?- find_path((1,1), (5,5), Path, 3).
```

```
Path = [(1,1), (3,1), (3,3), (4,3), (5,3), (5,4), (5,5)]
```

Все тесты были пройдены успешно, что подтверждает корректность реализации программы.

Вывод

В технологической части мы подробно описали реализацию программы, включая используемые технологии, среду разработки, структуру кода и процесс тестирования. Программа была успешно протестирована на различных сценариях, что подтверждает её работоспособность и эффективность в решении задачи нахождения выхода из лабиринта.

ЗАКЛЮЧЕНИЕ

Цель работы заключалась в разработке программы для формирования карты лабиринта и нахождения выхода из него. В ходе работы были выполнены задачи по созданию структуры лабиринта, реализации механики перемещения и поиску пути с учётом особенностей ячеек. Программа была реализована на языке Prolog и успешно протестирована. Результатом является работоспособная программа, способная эффективно находить выход из лабиринта.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Сергиевский Г. М., Волченков Н. Г. Функциональное и логическое программирование: учебник. – Москва: Издательский центр Академия, 2010. – 320 с.
2. Labyrinth Prolog — [<https://stackoverflow.com/>]
3. Язык Prolog — [<https://labs-org.ru/programmirovaniye-prolog/>]
4. Labyrinth — [<https://www.geeksforgeeks.org/>]

Приложение А

Код программы:

```
% Определяем структуру лабиринта
% labyrinth(M, N, Cells).
% M: количество строк, N: количество столбцов,
Cells: список определений ячеек
labyrinth(5, 5, [
cell(1, 1, empty),
cell(1, 2, wall),
cell(1, 3, empty),
cell(1, 4, empty),
cell(1, 5, empty),
cell(2, 1, teleport(3, 3)),
cell(2, 2, teleport(3, 3)),
cell(2, 3, empty),
cell(2, 4, wall),
cell(2, 5, empty),
cell(3, 1, teleport(3, 3)),
cell(3, 2, empty),
cell(3, 3, empty),
cell(3, 4, counting_trap(3)),
cell(3, 5, empty),
cell(4, 1, wall),
cell(4, 2, wall),
cell(4, 3, empty),
cell(4, 4, wall),
cell(4, 5, empty),
cell(5, 1, empty),
cell(5, 2, empty),
cell(5, 3, empty),
cell(5, 4, empty),
cell(5, 5, empty)
]).

% Определяем возможные ходы
move((X, Y), (X1, Y)) :- X1 is X + 1. % down
move((X, Y), (X1, Y)) :- X1 is X - 1. % up
move((X, Y), (X, Y1)) :- Y1 is Y + 1. % right
```

```

move((X, Y), (X, Y1)) :- Y1 is Y - 1. % left
move((X, Y), (X1, Y)) :- X1 is X + 2, Y = Y. % jump down
move((X, Y), (X1, Y)) :- X1 is X - 2, Y = Y. % jump up
move((X, Y), (X, Y1)) :- X = X, Y1 is Y + 2. % jump right
move((X, Y), (X, Y1)) :- X = X, Y1 is Y - 2. % jump left

```

```

% Проверьте, является ли ячейка допустимой
(находится ли она в пределах границ и не является стеной)
valid_cell((X, Y)) :-
    labyrinth(M, N, _),
    X > 0, X =< M,
    Y > 0, Y =< N,
    \+ cell((X, Y), wall).

```

```

% Получаем тип ячейки
cell((X, Y), Type) :-
    labyrinth(_, _, Cells),
    member(cell(X, Y, Type), Cells).

```

```

% Находим путь от Start до End
find_path(Start, End, Path, K) :-
    find_path(Start, End, [Start], Path, K, 0).

```

```

find_path(End, End, Visited, Path, _, _) :-
    reverse(Visited, Path).

```

```

find_path(Current, End, Visited, Path, K, Steps) :-
    move(Current, Next),
    valid_cell(Next),
    \+ member(Next, Visited),
    (    cell(Current, counting_trap(K)),
    NewSteps is Steps + 1,
    (    NewSteps >= K ->
    write('Trap triggered!'), nl,
    NewSteps1 is 0
    ;    NewSteps1 = NewSteps
    )
    ;    NewSteps1 = Steps
    ),

```

```
(    cell(Current, teleport(X, Y)) ->
Next = (X, Y)
;    Next = Next
),
find_path(Next, End, [Next | Visited], Path, K, NewSteps1).

% ?- find_path((1,1), (5,5), Path, 3).
```