УДК 004.4
DOI https://doi.org/10.32782/2663-5941/2024.3.1/20

Марчук Г.В.
Державний університет «Житомирська політехніка»

Любченко Д.В.
Державний університет «Житомирська політехніка»

ГЕНЕРАЦІЯ ЛАБІРИНТІВ ЗА ДОПОМОГОЮ АЛГОРИТМУ HUNT AND KILL

У статті досліджується алгоритм Hunt and Kill, що використовується для генерації випадкових лабіринтів. Тема генерації лабіринтів є актуальною для дослідників протягом багатьох років. Існує кілька популярних алгоритмів, таких як алгоритм Прима, Крускала, алгоритми глибинного пошуку. Усі ці методи мають свої переваги та недоліки, і вибір конкретного методу залежить від вимог до кінцевого продукту. Наукові роботи з цієї тематики зосереджені на оптимізації часу генерації, зменшенні використання пам'яті та покращенні якості генерованих лабіринтів. Дане дослідження ставить за мету провести комплексне вивчення алгоритму Hunt and Kill, висвітлюючи його сильні та слабкі сторони. У роботі з'ясовано, що алгоритм Hunt and Kill, на відміну від багатьох інших, дозволяє створювати лабіринти з менш передбачуваною структурою, що підвищує цікавість та складність його проходження.

Дослідження алгоритму Hunt and Kill, який генерує лабіринти шляхом прокладання випадкових шляхів через сітку та видалення стін, виявило, що його продуктивність залежить від двох ключових факторів – це розмір лабіринту і щільність стін. Чим більший лабіринт, тим довше буде проходити його генерація. Чим вища щільність стін, тим складніше алгоритму генерації знаходити вільні проходи та створювати лабіринт з багатьма тупиками. Це також збільшує час генерації. До факторів які можуть вплинути на продуктивність алгоритму може бути і його реалізація. Для генерації лабіринтів великого розміру доцільніше використовувати більш ефективні алгоритми. Подальші дослідження можуть бути спрямовані на оптимізацію алгоритму для покращення його продуктивності. Оптимізація алгоритму може зробити його ще більш корисним для широкого кола задач. Загалом, Hunt and Kill – це цінний інструмент для створення лабіринтів, який має як сильні, так і слабкі сторони.

Ключові слова: *лабіринт, генерація, алгоритм, Hunt and kill, складність, пошук шляху.*

Постановка проблеми. Генерація лабіринтів є важливою задачею в різних галузях, таких як ігровий дизайн, робототехніка, штучний інтелект та комп'ютерна графіка. Вона вимагає ефективних алгоритмів, здатних створювати складні, але проходжувані структури, які можуть бути використані в іграх, симуляціях або навчальних системах. Одним із таких алгоритмів є Hunt and Kill, який базується на поєднанні випадкового блукання та систематичного пошуку. Однак, попри його потенційні переваги, алгоритм Hunt and Kill залишається недостатньо дослідженим в аспекті його ефективності, складності та варіативності генерованих лабіринтів. Існує потреба в більш детальному аналізі цього алгоритму, зокрема, в контексті його порівняння з іншими популярними алгоритмами генерації лабіринтів, такими як Алгоритм Крускала, Алгоритм Прима та Алгоритм глибини-першого пошуку.

Таким чином, головною проблемою, яка потребує вирішення, є розробка та аналіз методів ефективної генерації лабіринтів за допомогою алгоритму Hunt and Kill, з метою виявлення його сильних і слабких сторін, а також можливостей для оптимізації та застосування в різних практичних задачах. Це включає дослідження аспектів, пов'язаних з якістю згенерованих лабіринтів, швидкістю виконання алгоритму та його адаптивністю до різних вимог користувачів.

Аналіз останніх досліджень і публікацій. У розробці відеоігор створення карт, ворогів і багатьох інших елементів ігрових рівнів є одним із важливих процесів. Щоб покращити ігровий досвід гравців, розробники ігор повинні розуміти поведінкові тенденції гравців і відповідно створювати рівні. У статті [1] проведено порівняння алгоритмів генерації лабіринтів. Проаналізовано їх природу та визначено найкращі з них. Досліджено алгоритми Прима, Крускала, DFS (Depth First Search), Еллерса, Вілсона, Hunt and Kill та алгоритм Альдуса-Бродера. Оцінка ефективності алгоритмів визначається двома параметрами:

часом, необхідним алгоритму для генерації лабіринту, і просторовою складністю лабіринту. Оцінки проводяться на основі кількості змінних, включаючи кількість перехресть і тупиків, разом із загальною кількістю кроків, зроблених агентами, щоб визначити, наскільки складним є лабіринт для навігації.

Лабіринти як проблемна область мають широке застосування, від мистецтва та ігор до тестування алгоритмів прийняття рішень. Хоча дослідження генерування лабіринтів традиційно зосереджувалося на топології, останнім часом з'явилося більше робіт, які досліджують генерацію лабіринтів з різним рівнем складності. У статті [2] досліджується метод генерації лабіринтів певної складності за допомогою нейронної мережі, параметризованого алгоритму генерації лабіринту та моделі складності.

У статті [3] проведено дослідження ранжування різних алгоритмів генерації лабіринтів за складністю згенерованих лабіринтів. В результаті було визначено, що найкращі алгоритми є похідними від алгоритмів пошуку рівномірних остовних дерев у графах.

Ідеальний лабіринт – це лабіринт, у якому будь-які дві клітини можна з'єднати унікальним шляхом. У літературі існує одинадцять алгоритмів генерації лабіринтів, зібраних Баком у 2015 році в його книзі « Лабіринти для програмістів». Кожен алгоритм створює лабіринти по-різному. Метою дослідження [4] було аналіз створення ідеального лабіринту. Дослідники запропонували два нових алгоритми генерації лабіринту, які називаються Prim and Kill і Twist end Merge. Ці два алгоритми генерують лабіринти інакше, ніж існуючі алгоритми.

У документі [5] описано реалізацію технології штучного інтелекту, відомої як генетичний алгоритм, яка використовується для вирішення випадково згенерованих лабіринтів різного розміру та складності. Щоб оцінити ефективність генетичного алгоритму, реалізовано декілька методів, не пов'язаних зі штучним інтелектом, наприклад пошук у глибину, пошук у ширину, алгоритм A-Star, алгоритм Дейкстри і Greedy.

У статті [6] наведено огляд трьох основних алгоритмів генерації двовимірних лабіринтів пошук у глибину (DFS), алгоритм Крускала та алгоритм Прима. Ці алгоритми описують три концептуально різні підходи до створення лабіринтів.

Алгоритми створення лабіринтів відіграють значну роль у відеоіграх з лабіринтами. Більшість існуючих методів спрямовані на підвищення продуктивності алгоритмів створення нерівномірного лабіринту. Однак створення різних лабіринтів з однаковою ймовірністю за допомогою єдиних алгоритмів має вирішальне значення для покращення ігрового досвіду користувача. У статті [7] досліджується розробка гібриду двох однотипних алгоритмів генерації лабіринтів, Олдоса-Бродера та Вілсона, чергуючи їх на деякій критичній фазі роботи, щоб зменшити час виконання.

У документі [8] представлено метод, який генерує бажаний лабіринт, коли користувачі надають потрібні властивості. У методі використовується підхід процедурного створення вмісту на основі пошуку (search-based procedural content generation, SBPCG), у якому процес неодноразово генерує та перевіряє лабіринти, щоб отримати задовільний лабіринт

Постановка завдання. Метою статті є дослідження алгоритму Hunt and Kill та його реалізація. Це включає детальне проектування алгоритму, реалізацію у вигляді програмного коду та тестування на предмет якості та ефективності згенерованих лабіринтів.

Виклад основного матеріалу. Алгоритм Hunt and Kill – це простий та ефективний алгоритм для створення лабіринтів. Він працює шляхом випадкового блукання по сітці та видалення стін, щоб створити проходи.

Алгоритм Hunt and Kill складається з двох основних етапів: режиму полювання (hunt) та режиму вбивства (kill). Для опису алгоритму використаємо наступні математичні позначення:

- M – матриця лабіринту розміром n m.
- V – множина відвіданих клітинок.
- C(i,j) – клітинка лабіринту на позиції (i,j).
- N(i,j) – множина сусідніх клітинок для C(i,j).

Етапи роботи алгоритму Hunt and Kill:

1. Ініціалізація. Обираємо випадкову клітинку C(i,j) та додаємо її до V.
2. Режим вбивства (Kill):

- Випадково обираємо сусідню клітинку C(i′,j′) з N(i,j), яка ще не відвідана.
- Додаємо C(i′,j′) до V та видаляємо стіну між C(i,j) та C(i′,j′).
- Повторюємо, поки всі сусідні клітинки не будуть відвідані.

3. Режим полювання (Hunt):

- Проходимо по всій матриці, поки не знайдемо не відвідану клітинку C(i′,j′), яка має відвіданого сусіда C(i″,j″).
- Додаємо C(i′,j′) до V та видаляємо стіну між C(i′,j′) та C(i/″,j″).

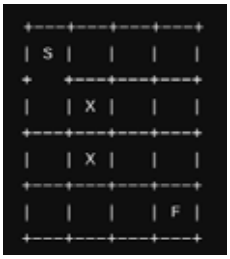

Рис. 1. Графічне представлення алгоритму Hunt and Kill

Повертаємося до режиму вбивства.

На рисунку 1 зображено матриця алгоритму Hunt and Kill, де S – початкова клітинка; F – Кінцева клітинка; X – Поточна позиція в режимі вбивства; +---+ – Стінки клітинок; - | – Відвідана клітинка без стіни.

На рисунку 2 наведено алгоритм побудови лабіринту за допомогою методу Hunt And Kill (HAK).

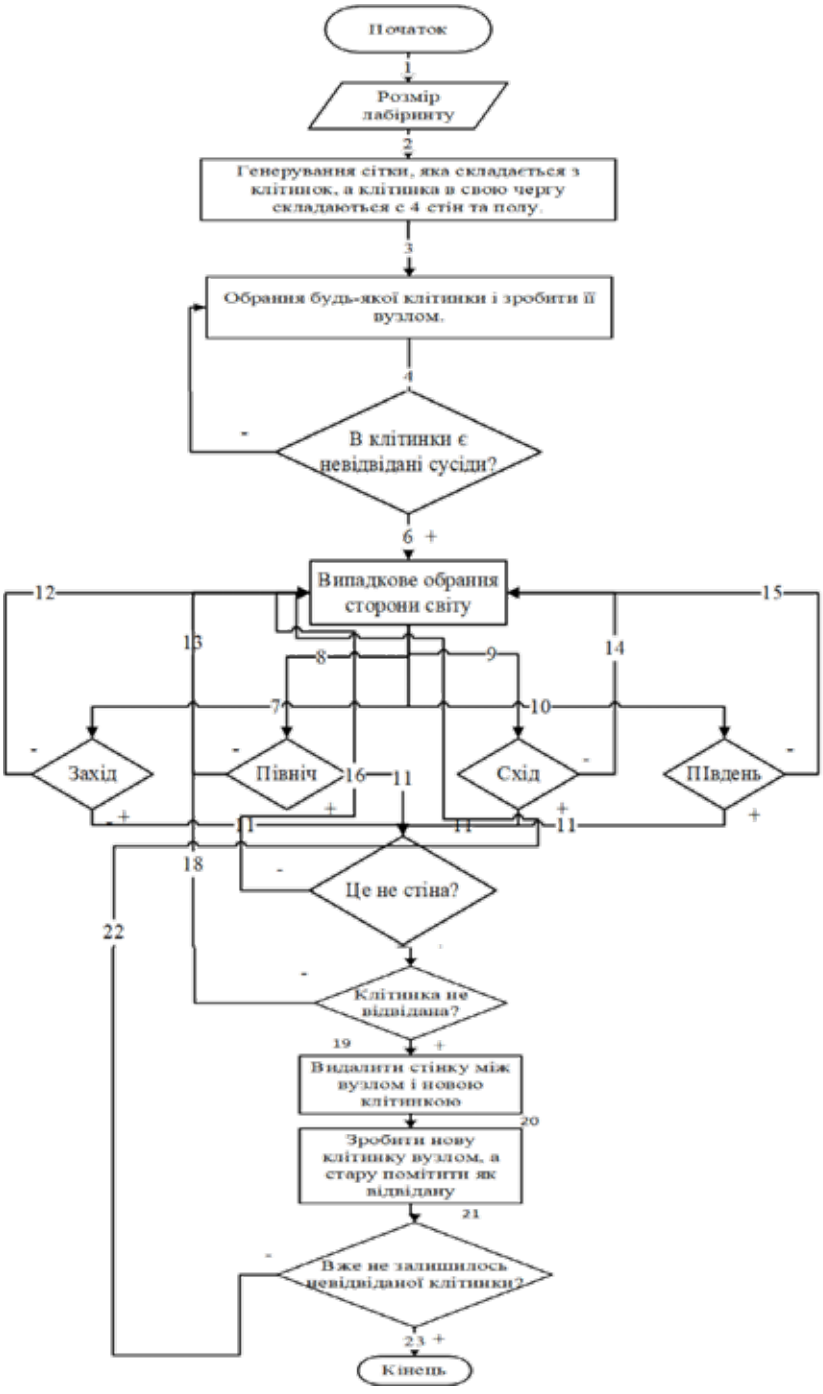

Рис. 2. Алгоритм побудови лабіринту Hunt And Kill

Початок роботи алгоритму починається одразу після передачі розміру лабіринту (1). Після відбувається генерація сітки (2). Після того як сітка буде сформована, алгоритм обирає будь-яку точку і робить її вузлом, тобто головною, а також помічає, що ця клітинка є відвіданою (3). Але для того, щоб йти далі, треба спочатку перевірити, чи сусіди клітинки є невідвідуваними (4). Якщо є, то треба випадково обрати одну із чотирьох сторін світу (6) і перевірити, чи обрана (7–10) сторона не є граничною стіною (11). Стіна – це границя лабіринту, її не можна рушити, так як це призведе до некоректної побудови лабіринту і до помилок. Після того як перевірили, треба зробити ще одну перевірку (17), а саме: чи обрана клітинка не відвідана; якщо ні то ця клітинка остаточно підходить (19), щоб стати продовженням майбутнього лабіринту. Обравши клітинку рушимо стінку (20) між вузлом (головна клітинка, звідки ми перейшли до нової) і обраної клітинки. Після того як стіна була знищена, ту клітинку, яку попередньо обрали робимо головним вузлом (21). Перероблюємо всі попередні дії до того моменту, поки вузол не «зажине себе в глухий кут» і йому не буде куди йти. При такій ситуації треба обрати новий вузол із тих, що залишились не відвіданими і пророблювати всі ці дії до того часу, поки не залишиться невідвідуваних клітинок (23). Після чого лабіринт буде повністю побудованим. Якщо якась клітинка не буде підходити то завжди буде обиратися нова клітинка (12–18).

Лістинг коду:

```
private void HuntAndKill() {
		mazeCells [currentRow, currentColumn].visited = true;
		while (!courseComplete) {
			Kill();
			Hunt();
		}
	}
	private void Kill() {
		while (RouteStillAvailable (currentRow, currentColumn)) {
			int direction = Random.Range (1, 5);
			if (direction == 1 && CellIsAvailable (currentRow - 1,
currentColumn)) {
				DestroyWallIfItExists (mazeCells [currentRow,
currentColumn].northWall);
				DestroyWallIfItExists (mazeCells [currentRow - 1,
currentColumn].southWall);
				currentRow--;
			} else if (direction == 2 && CellIsAvailable (currentRow + 1,
currentColumn)) {
								DestroyWallIfItExists (mazeCells
[currentRow, currentColumn].southWall);
				DestroyWallIfItExists (mazeCells [currentRow + 1,
currentColumn].northWall);
				currentRow++;
			} else if (direction == 3 && CellIsAvailable (currentRow,
currentColumn + 1)) {
				DestroyWallIfItExists (mazeCells [currentRow,
currentColumn].eastWall);
				DestroyWallIfItExists (mazeCells [currentRow, currentColumn +
1].westWall);
				currentColumn++;
			} else if (direction == 4 && CellIsAvailable (currentRow,
currentColumn - 1)) {
								DestroyWallIfItExists (mazeCells
[currentRow, currentColumn].westWall);
				DestroyWallIfItExists (mazeCells [currentRow, currentColumn -
1].eastWall);
				currentColumn--;
			}
			mazeCells [currentRow, currentColumn].visited = true;
		}
	}

	private void Hunt() {
		courseComplete = true;

		for (int r = 0; r < mazeRows; r++) {
			for (int c = 0; c < mazeColumns; c++) {
				if (!mazeCells [r, c].visited &&
CellHasAnAdjacentVisitedCell(r,c)) {
					courseComplete = false;
					currentRow = r;
					currentColumn = c;
					DestroyAdjacentWall (currentRow, currentColumn);
					mazeCells [currentRow, currentColumn].visited = true;
					return;
				}
			}
		}
	}
```

Таблиця 1

Результати генерації лабіринтів на Apple Macbook Air M1

Час виконання генерації лабіринту в залежності від розміру (в мілісекундах)			
10×10	55×55	100×100	500×500
0.25	1.5	7.0	75.0
0.30	2.0	8.5	80.0
0.28	1.8	7.5	78.0
0.27	1.7	6.8	76.5
0.26	1.9	7.2	77.5

Алгоритм Hunt and Kill – це алгоритм генерування лабіринтів, який належить до категорії алгоритмів випадкового вибору.

Часова складність алгоритму Hunt and Kill залежить від кількості ітерацій, необхідних для створення лабіринту. В середньому алгоритму потрібно пройти через кожну клітинку лабіринту один раз. В найгіршому випадку алгоритму може знадобитися пройти через кожну клітинку лабіринту кілька разів.

Просторова складність алгоритму Hunt and Kill залежить від кількості пам'яті, необхідної для зберігання інформації про лабіринт.

В таблиці 1 наведено результати дослідження роботи алгоритму Hunt and Kill

За результатами дослідження можна зробити висновки:

– Збільшення розміру лабіринту призводить до значного збільшення часу виконання.

– Час виконання не є лінійно-пропорційним розміру лабіринту.

– Найбільш значне збільшення часу виконання спостерігається при переході від 100×100 до 500×500 лабіринтів.

Висновки. Алгоритм Hunt and Kill є ефективним методом генерації лабіринтів, особливо у випадках, коли необхідно створити складні та непередбачувані структури. Основні переваги цього алгоритму полягають у його простоті реалізації та можливості створювати лабіринти з високим рівнем випадковості. Недоліками є потенційно довгий час виконання при великих розмірах лабіринту. Подальші дослідження можуть бути спрямовані на оптимізацію режиму роботи та зменшення витрат ресурсів.

Список літератури:

1. Mane, Deepak, et al. An Extensive Comparative Analysis on Different Maze Generation Algorithms. *International Journal of Intelligent Systems and Applications in Engineering* 12.2s (2024): 37–47.

2. Fujihira, Keita, Chu-Hsuan Hsueh, and Kokolo Ikeda. Procedural Maze Generation Considering Difficulty from Human Players' Perspectives. *Advances in Computer Games*. Cham: Springer International Publishing, 2021. 165–175

3. Gabrovsek, Peter. Analysis of maze generating algorithms. *IPSI Transactions on Internet Research* 15.1 (2019): 23–30.

4. Bellot, Victor, et al. How to generate perfect mazes? *Information Sciences* 572 (2021): 444–459. https://doi.org/10.1016/j.ins.2021.03.022

5. Sagming, M. N., Reolyn Heymann, and E. Hurwitz. Visualising and solving a maze using an artificial intelligence technique. *2019 IEEE AFRICON*. IEEE, 2019.

6. Shah, Ms Shivani H., et al. "Survey paper on maze generation algorithms for puzzle solving games." *International Journal of Scientific & Engineering Research* 8.2 (2017): 1064–1067.

7. Zhao, Guangxin, Hao Gu, and Emma Cai. A hybrid approach to maze generation algorithms. *International Conference on Algorithms, High Performance Computing, and Artificial Intelligence (AHPCAI 2023)*. Vol. 12941. SPIE, 2023.

8. Simonyan, Ernest, Olga Medvedeva, and Sergey Medvedev. The Reverse Approach for Generating Maze With Unique Characteristics. *2020 2nd International Conference on Control Systems, Mathematical Modeling, Automation and Energy Efficiency (SUMMA)*. IEEE, 2020.

Marchuk G.V., Liubchenko D.V. GENERATION OF MAZES USING THE HUNT-AND-KILL ALGORITHM

The article examines the Hunt and Kill algorithm used for generating random mazes. The topic of maze generation has been relevant to researchers for many years. There are several popular algorithms, such as Prim's, Kruskal, and depth-first search algorithms. Each method has advantages and disadvantages, and the choice of a specific method depends on the requirements for the final product. Scientific papers focus on optimizing generation time, reducing memory usage, and improving the quality of generated mazes. This study aims to comprehensively examine the Hunt and Kill algorithm, highlighting its strengths and weaknesses.

The study found that, unlike many others, the Hunt and Kill algorithm allows for creating mazes with less predictable structures, increasing the interest and difficulty of navigating through them. The investigation of the Hunt and Kill algorithm, which generates mazes by carving random paths through a grid and removing walls, revealed that its performance depends on two key factors – the size of the maze and the wall density. The more giant the maze, the longer the generation will take. The higher the wall density, the more challenging it is for the algorithm to find open passages and create a maze with many dead ends, increasing generation time. Factors that can influence the algorithm's performance include its implementation. For generating giant mazes, it is more appropriate to use more efficient algorithms.

Further research may focus on optimizing the algorithm to improve its performance. Optimization could make the algorithm even more helpful for many tasks. Overall, Hunt and Kill is a valuable tool for creating mazes with strengths and weaknesses.

Key words: *maze, generation, algorithm, Hunt and Kill, complexity, pathfinding.*