Unlucky Explorer: A Complete non-Overlapping Map Exploration

Mohammad Sina Kiarostami
Center for Ubiquitous Computing, Faculty of ITEE, University of Oulu
Oulu, Finland
mohammad.kiarostami@oulu.fi

Saleh Khalaj Monfared
School of Computer Sciences, Institute for Research in Fundamental Sciences (IPM)
Tehran, Iran
monfared@ipm.ir

Mohammadreza Daneshvaramoli
School of Computer Sciences, Institute for Research in Fundamental Sciences (IPM)
Tehran, Iran
daneshvaramoli@ipm.ir

Negar Yousefian
School of Computer Sciences, Institute for Research in Fundamental Sciences (IPM)
Tehran, Iran
n.yousefian@ipm.ir

Mahsa Massoud
School of Computer Sciences, Institute for Research in Fundamental Sciences (IPM)
Tehran, Iran
m.massoud@ipm.ir

Aku Visuri
Center for Ubiquitous Computing, Faculty of ITEE, University of Oulu
Oulu, Finland
aku.visuri@oulu.fi

Simo Hosio
Center for Ubiquitous Computing, Faculty of ITEE, University of Oulu
Oulu, Finland
simo.hosio@oulu.fi

Dara Rahmati
Computer Science and Engineering Department, Shahid Beheshti University
Tehran, Iran
d_rahmati@sbu.ac.ir

Saeid Gorgin
Iranian Research Organization for Science and Technology (IROST)
Tehran, Iran
gorgin@irost.ir

ABSTRACT

In this work, we introduce the *Maze Dash* puzzle as an exploration problem where the agent must find a *Hamiltonian Path* visiting all the cells with a minimum number of turnings for most cases. We also discuss the real-world application of the problem, such as *8 ball* billiards and *Snooker* games. We investigate different methods by a focus on *Monte-Carlo Tree Search* (MCTS) and *SAT* to get an overview of which class of solutions solves the puzzle quickly and accurately. Also, we perform optimization to the proposed MCTS algorithm to prune the tree search. Also, since the prefabricated test cases of this puzzle are not large enough to assay the proposed method, we employ a technique to generate solvable test cases to evaluate the approaches. Eventually, our comparison indicates that the MCTS-based approach is an up-and-coming method that could cope with the test cases with small and medium sizes with faster run-time than SAT. However, for specific discussed reasons, including the features of the problem, tree search organization, and also the approach of MCTS in the *Simulation* step, MCTS takes more time to execute in large size scenarios. Our results can be employed to choose a proper approach to create an AI to solve the *Maze Dash*, *8 ball* billiards, and *Snooker* games.

CCS CONCEPTS

• **Theory of computation → Solution concepts in game theory**; **Representations of games and their complexity**; *Theory of randomized search heuristics.*

KEYWORDS

Maze Dash, Exploration, Hamiltonian Path, Monte-Carlo Tree Search (MCTS), SAT.

ACM Reference Format:
Mohammad Sina Kiarostami, Saleh Khalaj Monfared, Mohammadreza Daneshvaramoli, Negar Yousefian, Mahsa Massoud, Aku Visuri, Simo Hosio, Dara Rahmati, and Saeid Gorgin. 2021. Unlucky Explorer: A Complete non-Overlapping Map Exploration. In *2021 The 3rd World Symposium on Software Engineering (WSSE 2021), September 24–26, 2021, Xiamen, China.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3488838.3488864

1 INTRODUCTION AND BACKGROUND

Graph Traversal is an important and famous problem in computer science with many applications in memory and storage systems [1], network flow [4], as well as computer games [14]. As the conventional approaches to solving the *Graph Exploration* problem and other variations like *Tree Traversal*, Depth-first search (DFS), and Breadth-first search (BFS) are known to be effective in general. On the other hand, random-based approaches such as Monte-Carlo Tree Search (MCTS) are demonstrated to be efficient in many search-based problems and games as well [3, 6, 11]. The fundamental problem of many simple computer games lies in solving specific computer or mathematical puzzles. The solution methodology used in many of these games is very relevant to fundamental approaches. For instance, *Flow-Free* is a variant of a known mathematical puzzle named *Numberlink*, and interestingly, the problem could be addressed as a non-Overlapping Multi-Agent Pathfinding [12]. In this context, *Icosian Game* [13] as an old mathematical game invented by W.R Hamilton, could be considered as a modified version of a Graph Traversal problem. The objective in Icosian is finding a *Hamiltonian Cycle* along the edges of a dodecahedron, visiting all the vertexes of a graph by ending at the same point as the starting vertex. The Hamiltonian Path problem as an NP-Complete problem [10] has its own applications in various fields [5] with many solution methods [2].

In this article, we investigate the foundation of the *Maze Dash* game and demonstrate that the constraints involved in solving the game inevitably minimize the number of turning movements in the grid exploration procedure. Satisfying this particular condition applied in this game could be interesting in terms of real-world robot exploration since an extra cost is often associated with the turning in intelligent explorer vehicles [8]. Also, this problem could be modelled to other computer games or physical sports such as *8 ball* billiards, and *Snooker* games [16]. In this game, the ball would turn only if it hits the wall or other balls, which are considered obstacles. Hence, finding an efficient and effective solution to the focused *Maze Dash* game could lead to faster *Grid Traversal* approaches where realistic restrictions in the robots are considered. Furthermore, we mathematically define the underlying primary problem of the game as a particular case of a Hamiltonian Path problem. Then, by studying the problem's specifications, we tackle the problem with different possible approaches, including the MCTS, as one of the promising methods. We examine the unique characteristics of the involved tree search in detail and study the exclusive attributes of the Hamiltonian Path in this problem.

1.1 Maze Dash Game

Maze Dash is a puzzle game with a single agent and a 2-Dimensional grid map. The map might have some obstacles or blocking cells. The agent moves in the map and marks the cells after visiting them by changing their colour and can not return to the marked cells. So, each cell must be visited just once. Eventually, the puzzle aims to visit all the cells or to explore the whole of the map. The essential rule in this puzzle is that if the agent chooses to go to one of the quad directions, it will continue to move until it reaches an obstacle or wall. As shown in Figure 1, the agent starts to move from the initial cell (S) and decides to go down to reach the wall or the border of the grid. Then, it keeps moving to explore all the viable cells, finishing the traversal at the last cell (E).

1.2 Monte-Carlo Tree Search

MCTS is a best-first search algorithm with four main steps which are *Selection*, *Expansion*, *Simulation*, and *Backpropagation*. This algorithm uses Monte-Carlo methods to sample steps and create the search tree indeterminately to solve problems in their particular domain [3]. In the context of our problem, the initial state is the state

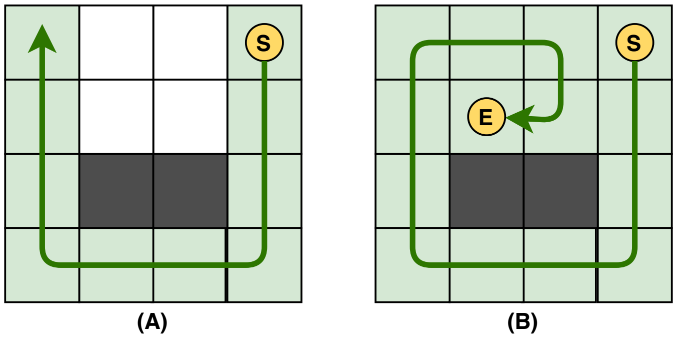

Figure 1: *Maze Dash* game solving process.

that the agent craves to move from its current cell. The first step of the MCTS is the selection that the algorithm chooses the best node, which is a leaf at the moment based on the *Tree Policy*. Then, at the expansion point, all non-terminal children of the selected node, if exist, will be expanded. In the next step, simulation, MCTS strides in the search tree aimlessly based on a policy until it reaches a leaf. The obtained result will be evaluated and measured how much is this result is analogous to the desired result and how many of the rules and conditions of the problem are satisfied. Finally, in the backpropagation, the results are propagated back through the tree, and all related node values are updated. After that, the next rounds will be iterated to find a suitable solution.

2 PROPOSED METHOD

This section defines the problem precisely, and then we provide several promising approaches to solve it.

2.1 Problem Definition

The exact definition of the problem as a modification of a *Hamiltonian Path* in a 2-D grid is described below. The set of $O = \{o_1, o_2, \ldots, o_m\}$ is demonstrated as the *obstacle* set which determines the coordination of the obstacle cells in the grid. By considering an $N \times N$ grid, the function π identifies the movement path in the grid:

$$\pi : \mathcal{N} \rightarrow \mathcal{N} \times \mathcal{N} \tag{1}$$

The input of π function rises incrementally to represents the movement path in the grid. The output represents the coordinates with the constrain of moving a single cell at each step to ensure the consistency of the solution path:

$$\begin{aligned}
&\pi(0) = S \\
&Direction = \{(1,0), (0,1), (-1,0), (0,-1)\} \\
&\forall (N^2 - |O|) > i > 0 : \\
&\pi(i+1) - \pi(i) \in Direction, \pi(i+1) \notin O \\
&\forall i, j : \pi(i) \neq \pi(j)
\end{aligned} \tag{2}$$

In Equation 2, S is the initial coordinate of the beginning cell. The Direction set D is the set of all possible movements that can be used in this context. As for the constraints regarding the reachable minimum turning movement restrictions in the game, the π function falls into either one of the two *Straight Movement*, *Turning*

Movement conditions as defined in Equation 3, respectively:

$$\begin{aligned}&\forall(N^2-|O|)>i>1: oneofThree:\\&Straight: \pi(i)+(\pi(i)-\pi(i-1))=\pi(i+1)\\&Turning: \pi(i)+(\pi(i)-\pi(i-1))\in O\\&Turning: \exists j<i: \pi(i)+(\pi(i)-\pi(i-1))=\pi(j)\end{aligned} \tag{3}$$

Note that the turning is occurred either by a blocking obstacle or a previously occupied cell by the earlier path. Hence, the i^{th} step in a cell must be as the same previous direction, or a turning movement happens.

Eventually, the solution of the game comprises of the adequate assignment for the π function satisfying all the constraints presented in Equations 2 and 3.

2.2 Promising Approaches

We apply and discuss three primary approaches to solve the problem.

2.2.1 SAT. The accurate mathematical definition discussed, as the foundation of the problem, could be fed to a Satisfiability solver (SAT Solver, e.g., *Z3*[9]) as a simple solution approach by converting the conditions into Boolean constraints. In this respect, the Boolean assignments of $t, s: \mathcal{N} \rightarrow \{0,1\}$, should be defined to determined whatever a cell falls into the *Turning Movement* or the *Straight Movement* sets. Then, by creating the same constraints in the function π for each cell of the grid, the Satisfiability check could be performed over the described assignments of $\{t, s, \pi\}$.

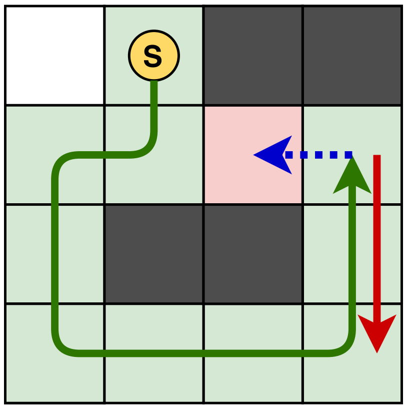

Figure 2: An example of the procedure of Backtracking in solving the puzzle.

2.2.2 Backtracking. As a naive approach to solve the problem, one could apply the *Backtracking* method to find the correct solution. The Backtracking process is very similar to a *DFS* method. Considering the conditions of the game, a single branch of the possible solution is pursued until all the cells are traversed, or a deadlock occurs. In the case of a failure, the movement path backtraces itself to the previous state and changes the branch by choosing another possible path. A simple backtrack demonstration is shown in Figure 2. The algorithm has to eliminate its current path to correct it since one of the cells is not visited.

2.2.3 MCTS. As mentioned in the background section, one of the particular and pronounced features of the game is that the agent must explore the grid with the reachable minimum number of turns. More precisely, the agent only could change its direction when it reaches the end of the current path. This constraint intensely affects the tree search of the puzzle. As shown in Figure 3, each non-terminal node of the tree search despite of the starting point, has only one or two children. Therefore, the *Branching Factor* of the tree is equal to or smaller than two making the tree significantly long in-depth. The starting point can be in the middle of a empty grid, then it has more than two expandable children. One of the approaches that could solve the problem is MCTS. As explained earlier, MCTS runs its four steps iteratively to construct the tree search and find the solution.

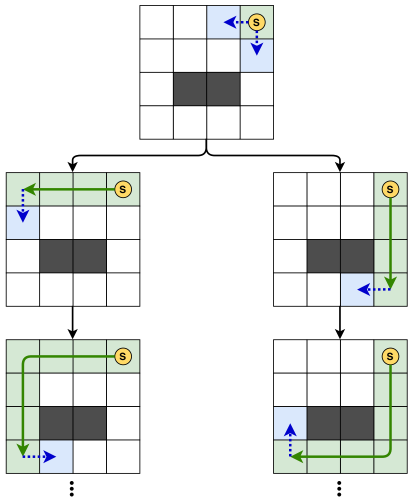

Figure 3: Construction of the tree search of the puzzle.

We disregard the nodes with a certain failure at their final stats to prune the tree to solve the problem more efficiently. More precisely, the Expansion function is amended to block a terminal state with an inadequate value of the desired final state. Then, the Selection function blocks a node if all of its children were blocked before by assigning 0 as the node's value. Therefore, gradually wrong paths could be blocked efficiently. Nevertheless, even by employing these modifications to the algorithm, the execution time increases immensely due to the computational overhead caused by the huge depth in the tree. Another optimization is employed by applying the *Fast Rollout Function*, where instead of constructing a new state for each *Child Node*, the state of the parent is updated each time. This optimization reduces the memory consumption of $O(N^2)$ to $O(1)$ for each node [7]. These details will be discussed in the result section.

Another approach to the problem could be a BFS method where all of the branches are searched at the time without traversing deep into the final state. Not surprisingly, this method burdens many memory usages to store the branch states. This memory overuse could be predicted due to the tremendous depth of the tree search. Consequently, a BFS approach would not be a suitable solution compared to other possible methods.

3 EVALUATION

As the test cases of the *Maze Dash* game are not large enough to assess the methods correctly, we utilize a method to generate significant random test cases. First, an empty grid with the desired size is assumed. Then an agent starts to move through the grid randomly based on the *uniform random* policy. Whenever the agent turns or changes its direction, an obstacle is placed at the next cell of the current cell. It means that there was a hypothetical obstacle in the path, so the agent decided to change the direction. For avoiding creating a unique path for solving the test case, we defined a variable as the number of obstacles. After generating the test cases, all of them would be tested by the Backtracking approach without any concerns regarding the execution time to ensure that they are solvable. All of the C-programmed compiled files with GCC compiler are executed on a machine running Ubuntu 16.04 equipped with two Intel XEON E5 2697V3 CPUs clocked at 2.6 GHz and 128 GB of DDR3 RAM.

As shown in Table 1, we have implemented and compared possible approaches to solve the defined problem. Each test case is executed by each method 50 times, and the average values are presented. The Backtracking algorithm is used to indicate our worst-case scenario, not to be a good rival. The MCTS approach as a promising candidate from random-based approaches performs well in small and medium-size test cases but could not cope with large ones under one minute, as shown in Figure 4. Also, based on Figure 4, our implemented SAT method indicates that it is a stable approach and solves the problem with any size, particularly in terms of memory consumption. However, MCTS is better than SAT in the test cases that it can solve. We should clarify that both Backtrack and MCTS methods can solve all the test cases in, let us say, an unlimited time, but we consider *Failed* as a result if they can not solve the problem in under one minute.

As highlighted in Table 1, we can see that MCTS handles typical test cases with a lesser number of obstacles due to the structure of the tree search, which causes the algorithm to search randomly better than the exact computations of the SAT method. The results of the MCTS method could be discussed more in detail. First, most of the execution run-time of the algorithm is spent in the simulation step. Assume that I defines the number of iterations of the simulation in each MCTS traversal. In each simulation, the algorithm would traverse the tree down to N^2 depth. After selecting each node and adding one depth to the MCTS tree, the algorithm would be repeated. Thus, the search would be performed for another N^2 times. Ultimately, the simulations process enlarges and will have an immense cost, calculated as follows:

$$SimulationsCost = I \times N^4 \tag{4}$$

This order is a massive cost for the problem since the agent could not determine or predict a complete solution until it tests all possible movements. For instance, the *Evaluation Function* could return 0.98 as a value of a final state, meaning that only two cells are not visited among 100. We know that this state is not the accurate answer, but the algorithm would recognize it as a promising state in the previous steps. So, the algorithm would never prune these kinds of states. Furthermore, the agent would face new obstacles after each movement due to the non-overlapping problem's feature.

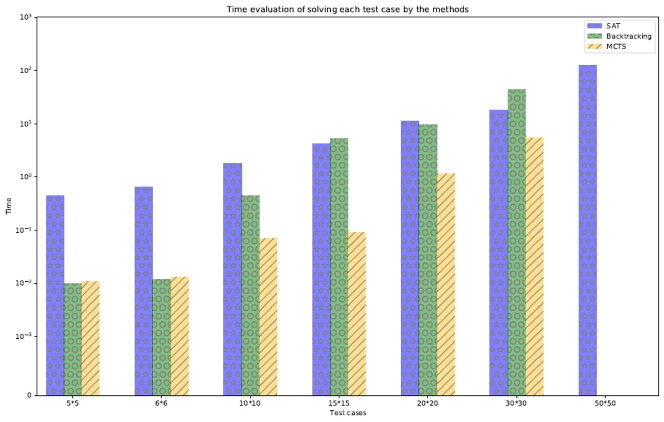

Figure 4: Comparison of the promising approaches in different test cases.

Moreover, previously visited cells are considered dynamic obstacles and walls, making the algorithm unable to be optimized by any pre-process methods. Furthermore, as explained, the branching factor of the problem is equal to or smaller than 2, constructing a tree with a considerable depth without many branches. This kind of tree produces a troublesome circumstance for MCTS in its simulation step.

4 DISCUSSION AND CONCLUSION

In this article, we first introduced the *Maze Dash* puzzle as a modification of the exploration problem or, more precisely, a non-overlapping exploration that the Hamiltonian Path could solve. It means that the primary purpose of this problem is to visit all of the cells in a 2-D grid. We investigated promising approaches to find a proper solution. Although three methods implemented to compare with each other, the main focus was on MCTS and SAT. However, our study is not finished yet, and we will investigate learning-based methods, such as Reinforcement Learning, as future work.

As expected, our implemented SAT could solve the auto-generated test cases accurately. Nevertheless, by reducing the number of obstacles in the grid, the execution time increased exponentially. In small and medium-size test cases, MCTS could outperform SAT. As explored in the significant test cases, the simulation time increased uncontrollably since the algorithm recognizes the failure early in the simulation. As our observation, due to the non-overlapping feature of the problem, the agent considers its previous path as dynamic walls or obstacles. Therefore, pre-processing methods could not optimize the algorithm, but the SAT performs more beneficial in these test cases.

Investigating the introduced intricacy, we came to realize two more practical and pronounced problems that should be considered. The first one is reasonably analogous to the current problem but differs in the non-overlapping constraint. It means that the aim of the problem is that an agent must explore all of the environment fast and accurately. However, the agent's previous path would not be defined as a new obstacle. Thus, the agent prefers not to use the

Table 1: Evaluation of different algorithms for solving *Maze Dash* game.

Grid Size(N^2)	No. Obstacles	SAT method		Backtrack method (DFS)		Randomized method (MCTS)	
		Run-Time(S)	Memory(MB)	Run-Time(S)	Memory(MB)	Run-Time(S)	Memory(MB)
5x5	4	0.45	6.01	**0.010**	7.7	0.011	**5.94**
6x6	10	0.65	8.12	**0.012**	**8.1**	0.013	10.61
10x10	32	1.79	12.83	0.45	**10.4**	**0.07**	11.5
15x15	66	4.18	**13.27**	5.24	13.5	**0.09**	14.2
20x20	133	11.28	**13.86**	9.62	19.2	**1.16**	16.9
30x30	378	17.94	**15.5**	43.7	26.1	**5.42**	20.1
50x50	776	**126.6**	**49.9**	Failed	Failed	Failed	Failed

visited cells but is not forced to do this. The second problem is the exploration of a grid to find a goal with minimum numbers of turns by assuming that turning movement has an additional cost since the agent must reduce its velocity, stop, and then start to move again [8, 15]. By this constraint, the agent prefers to choose a path with lesser turnings. In our future studies, we would concentrate on these two problems, which would be helpful in real-world applications, such as 2-D Robotic soccer.

ACKNOWLEDGMENTS

This research is connected to the GenZ strategic profiling project at the University of Oulu, supported by the Academy of Finland (project number 318930), and CRITICAL (Academy of Finland Strategic Research, 335729). Part of the work was also carried out with the support of Biocenter Oulu, spearhead project ICON.

REFERENCES

[1] Alok Aggarwal and S Vitter, Jeffrey. 1988. The input/output complexity of sorting and related problems. *Commun. ACM* 31, 9 (1988), 1116–1127.

[2] Andreas Bjorklund. 2014. Determinant sums for undirected hamiltonicity. *SIAM J. Comput.* 43, 1 (2014), 280–299.

[3] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. 2012. A Survey of Monte Carlo Tree Search Methods. *IEEE Trans on Compt Int and AI in Games* 4, 1 (March 2012), 1–43. https://doi.org/10.1109/TCIAIG.2012.2186810

[4] To-Yat Cheung. 1983. Graph traversal techniques and the maximum flow problem in distributed computation. *IEEE Transactions on Software Engineering* 4 (1983), 504–512.

[5] James Cooper and Radu Nicolescu. 2019. The Hamiltonian cycle and travelling salesman problems in cP systems. *Fundamenta Informaticae* 164, 2-3 (2019), 157–180.

[6] Juhriyansyah Dalle, Dwi Hastuti, and Muhammad Riko Anshori Prasetya. 2021. The Use of an Application Running on the Ant Colony Algorithm in Determining the Nearest Path between Two Points. *Journal of Advances in Information Technology Vol* 12, 3 (2021).

[7] Mohammadreza Daneshvaramoli, Mohammad Sina Kiarostami, Saleh Khalaj Monfared, Helia Karisani, Keivan Dehghannayeri, Dara Rahmati, and Saeid Gorgin. 2020. Decentralized Communication-less Multi-Agent Task Assignment with Cooperative Monte-Carlo Tree Search. In *2020 6th International Conference on Control, Automation and Robotics (ICCAR)*. 612–616. https://doi.org/10.1109/ICCAR49639.2020.9108073

[8] Mansoor Davoodi, Fatemeh Panahi, Ali Mohades, and Seyed Naser Hashemi. 2015. Clear and smooth path planning. *Applied Soft Compt* 32 (2015), 568–579.

[9] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An efficient SMT solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 337–340.

[10] Juris Hartmanis. 1982. Computers and intractability: a guide to the theory of NP-completeness (michael r. garey and david s. johnson). *Siam Review* 24, 1 (1982), 90.

[11] Khaoula Hassoune, Wafaa Dachry, Fouad Moutaouakkil, and Hicham Medromi. 2020. Dynamic Parking Guidance Architecture Using Ant Colony Optimization and Multi-agent Systems. *Journal of Advances in Information Technology Vol* 11, 2 (2020).

[12] M. S. Kiarostami, M. Reza Daneshvaramoli, S. K. Monfared, D. Rahmati, and S. Gorgin. 2019. Multi-Agent non-Overlapping Pathfinding with Monte-Carlo Tree Search. In *2019 IEEE CoG*. 1–4.

[13] Ed Pegg Jr. 2009. The icosian game, revisited. *The Mathematica Journal* 11, 3 (2009), 310–314.

[14] Aske Plaat, Jonathan Schaeffer, Wim Pijls, and Arie De Bruin. 1996. Exploiting graph properties of game trees. In *AAAI/IAAI, Vol. 1*. 234–239.

[15] HTWK Robots. [n.d.]. *GO 2015 SPL Finals: Nao-Team HTWK vs. B-Human 1st half.* Youtube. https://www.youtube.com/watch?v=NFNEOooEQX4

[16] David Silva and Rui Prada. 2018. MiniPool: Real-time artificial player for an 8-Ball video game. (2018).