Coursework

Project Title: Performance of A* Algorithm in a 2D platformer

Student ID: 20113642

Module: Designing Intelligent Agents (COMP3004)

Table of Contents

1. Introduction

A platformer is a game genre where the objective of the player is to move the in-game character to a position on a screen to a point of interest in order to fulfil a goal, be it to obtain a higher score by preventing the death of the in-game character or to arrive at a final objective. In a platformer game the in-game character is usually controlled by the player inside the in-game environment. The in-game characters in these platformers usually have the ability to move accordingly to the dimensions of how the game is rendered, given a 2D platformer environment, the in-game character can usually move left and right, jump and in some cases use an ability, for example, expelling a projectile from the in-game character's body.

The A* search algorithm is a search algorithm that was proposed by Hart, P. E., Nilsson, N. J & Raphael, B. (1968) **[1]**, this search algorithm can be viewed as a compliment to Djikstra's algorithm **[2]**. The purpose of Djikstra'a algorithm is to find the closest path given two nodes **[2]**, this algorithm can be used to find the shortest route between a "source" node and every other node in the graph **[3]**. This is usually done between a given starting node and an ending node in a graph. In addition to this, the A* search algorithm only finds the shortest path between a specified start node and a specified end node by using heuristics to better improve its' searching capabilities.

In this coursework, the A* algorithm is utilised in finding the shortest route in a two-dimensional platforming environment, this is done to analyse its performance in this environment.

2. Related Works

Path finding for the most optimal paths to a goal in a virtual environment has been done, in one case, in a two-dimensional platformer video game environment called Super Mario, a YouTuber with the channel name of SethBling, uses a genetic algorithm that generates evolving artificial neural networks called Neuroevolution of Augmenting Topoligies (NEAT) that was suggested by Ken Stanley in 2002, to complete levels in Super Mario **[4]**.

In the case of A* search algorithm has been utilised in many fields of research and even in real-world applications due to its effectiveness and robustness in path finding. However, it was originally designed to be used for general graph traversing applications **[5]**. In a paper, Wildan Budiawan *et al.* compares the effectiveness of the A* search algorithm against the Steepest Ascent Hill Climbing (SHAC) in games of Tic-Tac-Toe. They test these two heuristic algorithms for their speed and likeliness to win against a human player in the game. The results for this research is that SHAC performs faster while the A* algorithm is less likely to lose to the player.

In another case, Iskandar *et al.* identifies several pathfinding algorithms including Djikstra's, A* search, Depth-First Search, Breadth-First Search, Best First Search, and finally D* algorithms, then describes each of their strengths and weaknesses in the domain of pathfinding in platformer games.

3. Methodology

There are several steps that can be followed to qualitatively analyse the A* search algorithm in a two-dimensional platforming environment.

1. Generate a two-dimensional environment and an agent at the bottom which is the starting point of the environment.
2. Generate a fixed floor platform at the bottom of the environment then generate randomly placed platforms leading up to the goal platform.
3. Use an agent that utilises the A* search algorithm and a greedy search algorithm to traverse through the generated environments, then record the findings.
4. Count the number of platforms used by the agent in each generated environment, then visualise a graph based on that.
5. Compare the findings from each of the experiments.

3.1 Environment Generation

Before performing any experiments, an environment should be generated initially. After that, platforms within the environment are randomly generated along with a fixed floor platform to support the agent when generating it, a fixed goal platform initially and an agent that can traverse through the environment. Then can the search algorithms.

ALGORITHM 1: GENERATING THE ENVIRONMENT

```
Input: Seed, $i_{seed}$, integer value representing the random seed
Initialisation: Window dimension, $W_{(x,y)}$, integer tuple representing the size of the
                    environment, which is the game window
                Platform dimension, $P_{(x,y)}$, integer tuple representing the size of a
                    platform
                Agent dimension, $A_{(x,y)}$, integer tuple representing the size of the
                    agent

display_window ← set display dimensions to $W_{(x,y)}$
agent_object ← set sprite dimensions to $A_{(x,y)}$ at the bottom of display_window
floor_object ← set sprite dimensions to cover the entire bottom of display_window
goal_object ← set sprite dimensions to $P_{(x,y)}$ at the top of display_window
current_y_position ← integer set to $(P_y \times 2 + A_y \times 2)$
random.seed( $i_{seed}$ )
WHILE $current\_y\_position > goal\_object_y$ DO
    FOR a random range between 1 and 4 DO
        create sprite object of size $P_{(x,y)}$ at position…
                ... $(random(W_x), current\_y\_position)$
    ENDFOR
    current_y_position ← current_y_position + $P_y \times 2 + A_y \times 2$
ENDWHILE
```

Pseudocode 3.1.1: Pseudocode for platform generation

The above algorithm is used to generate all the platforms including the floor platform where the agent is positioned initially and the goal platform where the agent is supposed to reach during an experiment. Both goal and floor platforms are positioned at the top and bottom of the map respectively. Random platforms will be generated between the goal and floor platforms, from the top to the bottom of the display window.

3.2 Search Algorithms

ALGORITHM 2: A* SEARCH ALGORITHM

```
Input: All platforms, all_platforms, a pygame group object containing all platform
        objects that exist in the environment
Initialisation: Priority queue, priority_queue, an array containing the positions of
                all platforms that have not been traversed
                Completed queue, platforms_traversed, an array representing a
                queue that stores platforms that had been traversed
                Current platform, current_platform, an integer tuple that contains
                the current platform selected by the algorithm

current_platform ← floor platform position
WHILE $current\_platform \neq goal\_platform\_position$ DO
    platforms_in_range ← empty array
    FOR platform IN all_platforms.positions DO
        IF current_platform is within range of platform
            current_platform ← append platform
        ENDIF
    ENDFOR
    FOR platform IN platforms_in_range DO
        distance ← calculate distance between platform & current_platform
        heuristic ← calculate distance between platform & the goal platform
        total_distance ← distance + heuristic
        IF platform NOT IN priority_queue DO
            priority_queue ← append (platform, total_distance) tuple
        ELSE IF priority_queue[paltform] > total_distance DO
            priority_queue[paltform] ← total_distance
        ENDIF
        priority_queue ← sort priority_queue by total_distance
    ENDFOR
    platforms_traversed ← append current_platform
    current_platform ← priority_queue at index 0
    priority_queue ← delete priority_queue at index 0
ENDWHILE
platforms_traversed ← append current_platform
```

Pseudocode 3.2.1: Pseudocode for the implementation of A* search algorithm in this coursework

The pseudocode shown above represents the A* search algorithm used to find the shortest route between the floor platform where the agent is initially positioned and the goal platform which the agent should reach. The array *platforms_traversed* will then be given to the agent in order for it to traverse through the generated environment. The way that distance is calculated is by using the equation of Euclidean distance, in this case, the pixel coordinate values of both objects are used to calculate the distance between them.

ALGORITHM 3: GREEDY SEARCH ALGORITHM

```
Inputs: Agent, agent_obj, a pygame sprite object that represents an agent
        All platforms, all_platforms, a pygame group object containing all
        platform objects that exist in the environment

WHILE agent_obj has not reached goal_platform DO
    closest_platform_position ← (-1, -1)
    closest_platform_distance ← system maximum integer value
    FOR platforms IN all_platforms DO
        platform_distance ← distance between player_obj and platform
        IF platform_distance < closest_platform_distance DO
            closest_platform_distance ← platform distance
            closest_platform_position ← platform position
        ENDIF
    ENDFOR
    agent_obj ← set new target platform to closest_target_position
```

Pseudocode 3.2.2: Pseudocode for the implementation of a greedy search in this coursework

The pseudocode above illustrates the greedy search algorithm utilised by the agent, the algorithm, unlike the previous A* search algorithm is perform concurrently during the simulation of the agent traversing the environment. The agent will check for platforms that are within the range of the agent then perform movements toward that platform and the agent will repeat this process until it reaches the goal.

4. Results

As the platforms in the environment are generated based on seeded random generation, the environments used can be generated repeatedly, therefore the same environment can be used for both of the search algorithms.

4.1 Platform Generation

A total of 100 different environments are generated and used for both the greedy search algorithm and the A* algorithm. The following is an example of an environment that was generated,

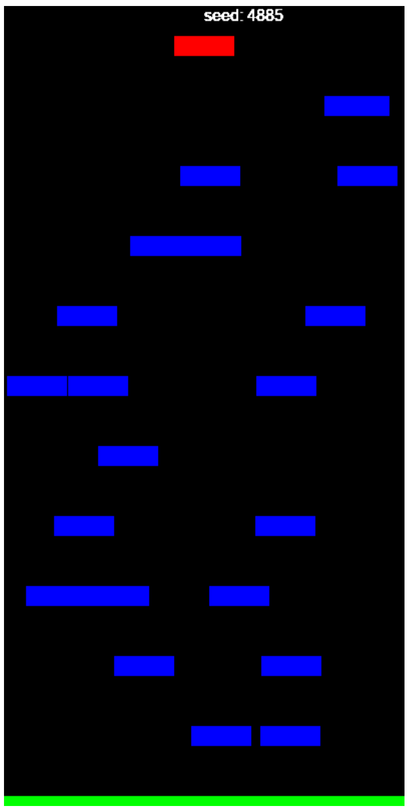


Figure 4.1.1: Randomly generated environment seeded at 4885

The above is an example of a seeded random generation out of the 100 seeded random generated environments. The green colour indicates the floor platform, the red colour implies the goal platform, and the blue colour represents platforms that were randomly generated. The complete 100 seeded random generated environments can be seen in **Figure 8.1** in the references section of this report.

4.2 Comparison Between Search Algorithms

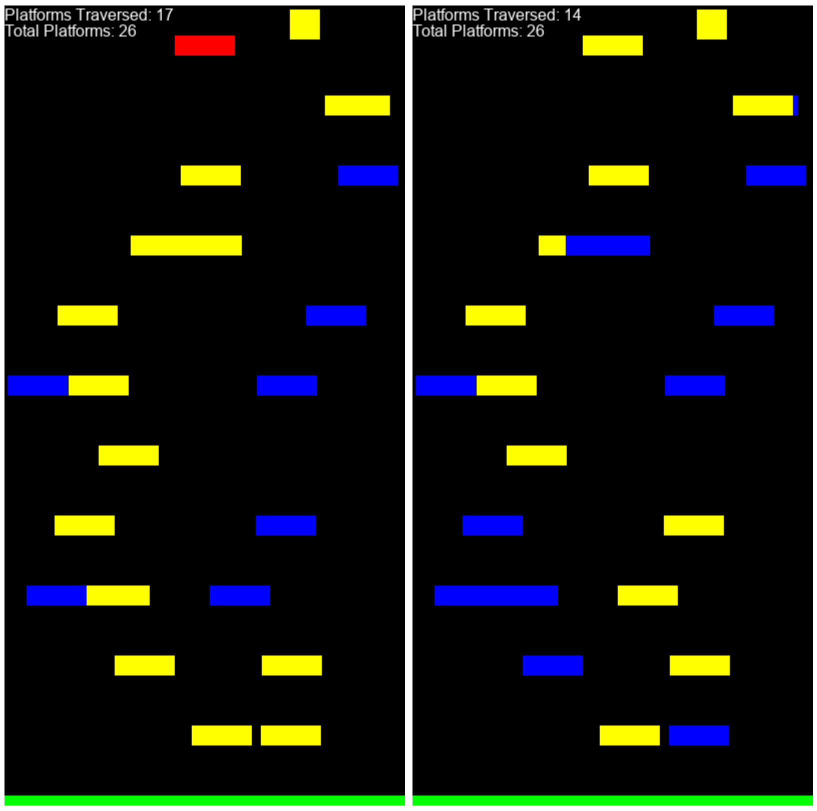


Figure 4.2.1: A comparison of platform traversal between the greedy search algorithm (left) and the A* search algorithm (right) on the 4885 seed

The figure above shows the difference in traversal routes between both search algorithms used by the agent, which is the yellow square, between the left, the greedy search algorithm route, and the right, the A* search algorithm route. Within the same environment, the agent managed to reduce the number of platforms traversed in order to reach the desired destination by using the A* search algorithm. As the greedy search algorithm chooses the nearest platforms to proceed, it costed the agent more in order to reach its destination using the greedy search algorithm. Another interesting comparison can be made with the following the figure,

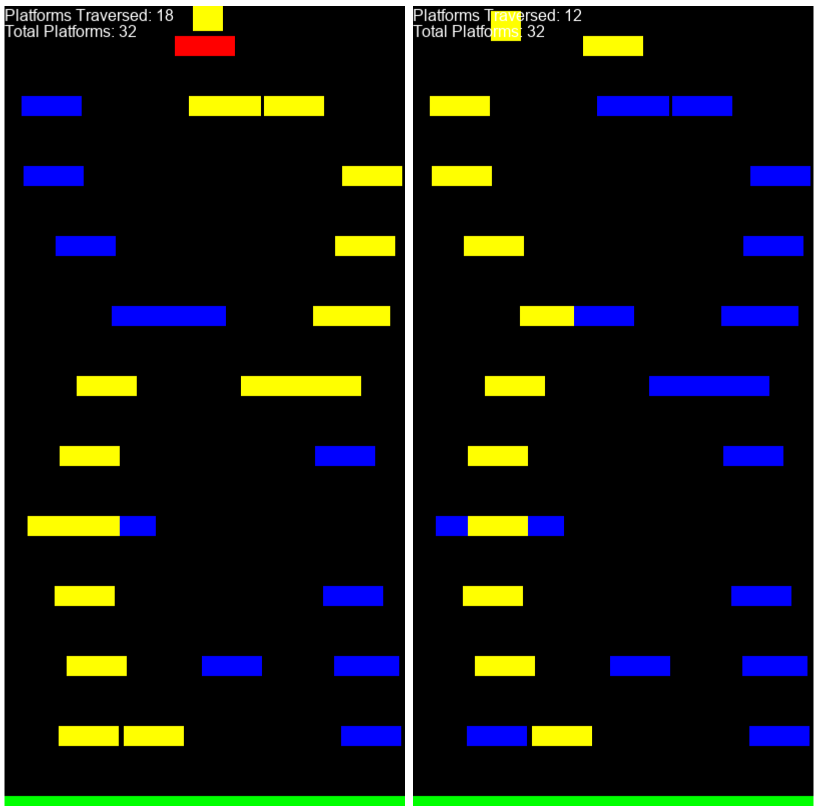


Figure 4.2.2: A comparison of platform traversal between the greedy search algorithm (left) and the A* search algorithm (right) on the 9775 seed

The above figure shows the comparison of routes used between the greedy search algorithm and the A* search algorithm. The A* search algorithm has yet again performed phenomenally in terms using the least platforms in order to reach the goal platform. While the agent on the left chooses to change direction in the middle of reaching its destination, causes it to traverse through more platforms, meanwhile the agent on the right sticks to the left side of the environment to get to its destination which causes it to take less steps to get to its destination. Several of these interesting differences can be seen in the 100 iterations for each algorithm, the results of all environment traversals can be seen in **Figure 8.2**, for the greedy search algorithm, and **Figure 8.3**, for the A* search algorithm, in the references section of this report.

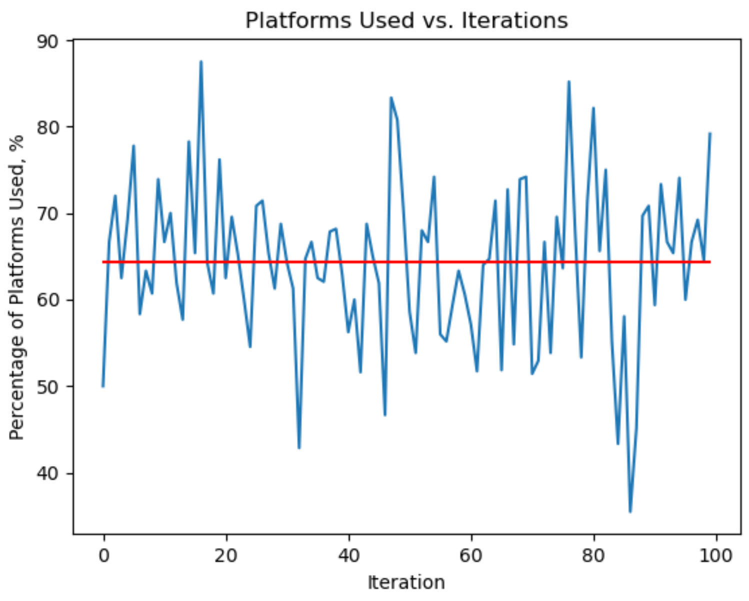


Figure 4.2.3: Graph representing the percentages of total platforms used by the agent using the greedy search algorithm

The above figure is a graph that illustrates the percentage of total platforms used over 100 iterations. The red line represents the mean percentage of total platforms used over these iterations, which displays the mean as 65% of the platforms will be used by the agent to reach the goal for the greedy search algorithm. With the highest possible platforms used, 87% and the lowest possible platforms used, 38%, the graph is mostly volatile as the total number of platforms that exist in any of the randomly generated environments are also random.

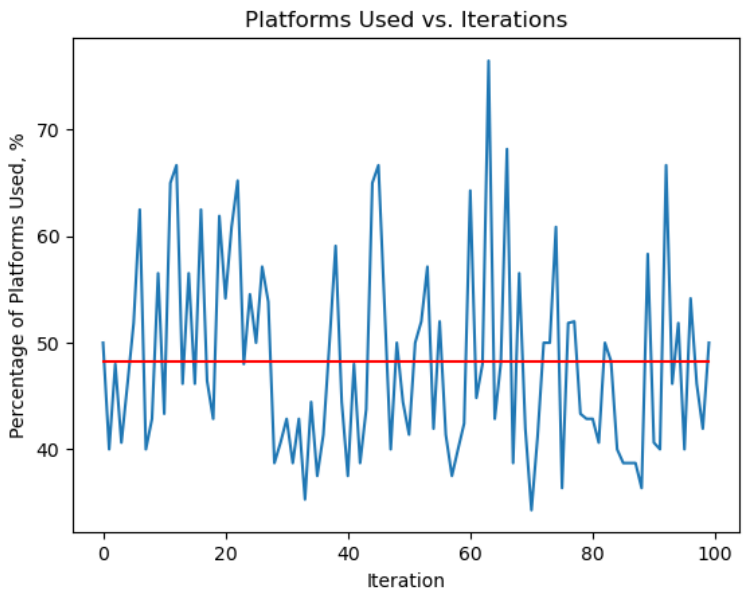


Figure 4.2.4: Graph representing the percentages of total platforms used by the agent using the A* search algorithm

Compared to **Figure 4.2.3**, the above figure illustrates lower percentages of platforms used by the agent when it utilises the A* search algorithm. While the mean for the previous figure, **Figure 4.2.3**, is 65%, the mean percentage of total platforms used by the agent for **Figure 4.2.4** is only around 48%. This shows a significant improvement in efficiency of the A* search algorithm compared to the greedy search algorithm. With the highest spike in percentage of total platforms used as 76% and having most of the percentages of total platforms used stay within the range of 34% and 50%, which demonstrates that the algorithm is able to consistently use fewer platforms for the agent to get to the top.

5. Discussion

Although both algorithms are implemented perfectly, some iterations of both the greedy search algorithm and the A* search algorithm present some invalid results. This is where the agents are not able to reach the final platform although they understand that they have to reach them. This could be either due to the physics mechanics or the collision mechanics used to implement the gravity and jumping of the agent in the virtual environment. However, these faulty iterations are still unnoticeable in the grand scheme of things as these faults are only a small subset of the large number of iterations performed.

6. Conclusion

It is clear that the A* search algorithm will outperform any other simplistic search algorithms that are used to compare against it. However, the implementation of this algorithm in a game-playing environment has rarely been performed much less in a two-dimensional platforming environment. However, without performing such an experiment of comparison between two algorithms, the exact differences in efficiency cannot be conclusively determined. In this case, the A* search algorithm manages to be much more efficient in traversing through a two-dimensional platforming environment compared to the greedy search algorithm.

7. Future Work

Several aspects of the program can be further meliorated, for instance, the movement mechanics of the game can be improved as the agent tends to stay still or become stuck in some generated map iterations and requires a slight nudge by an external force in order to force the agent to continue moving towards a platform. The physics logic in the game can also be further improved as the agent tends to fall under the map at times, while the current method to remedy this is to teleport the agent back onto the floor platform if it falls out of bounds.

Furthermore, a comparison of the algorithm against other existing algorithms can also be done in the future. Comparisons against population-based search algorithms, such as genetic algorithm, particle swarm algorithm and so on, can be used to compete against the A* search algorithm. Then additional obstacles such as disappearing platforms, trap platforms and so on can be used on these population-based search algorithms.

8. References

[1] Hart, P. E., Nilsson, N. J., Raphael, B. (1968). *"A Formal Basis for the Heuristic Determination of Minimum Cost Paths"*. IEEE Transactions on Systems Science and Cybernetics. 4 (2): 100–107. doi:10.1109/TSSC.1968.300136.

[2] Djikstra, E. W (1959). *“A note on two problems in connexion with graphs”(PDF). Numerische Mathematik.* 1: 269–271. doi:10.1007/BF01386390. S2CID 123284777.

[3] Mehlhorn. Kurt, Sanders. Peter(2008). *“Chapter 10. Shortest Paths”(PDF). Algorithms and Data Structures: The Basic Toolbox. Springer.* doi:10.1007/978-3-540-77978-0. ISBN 978-3-540-77977-3.

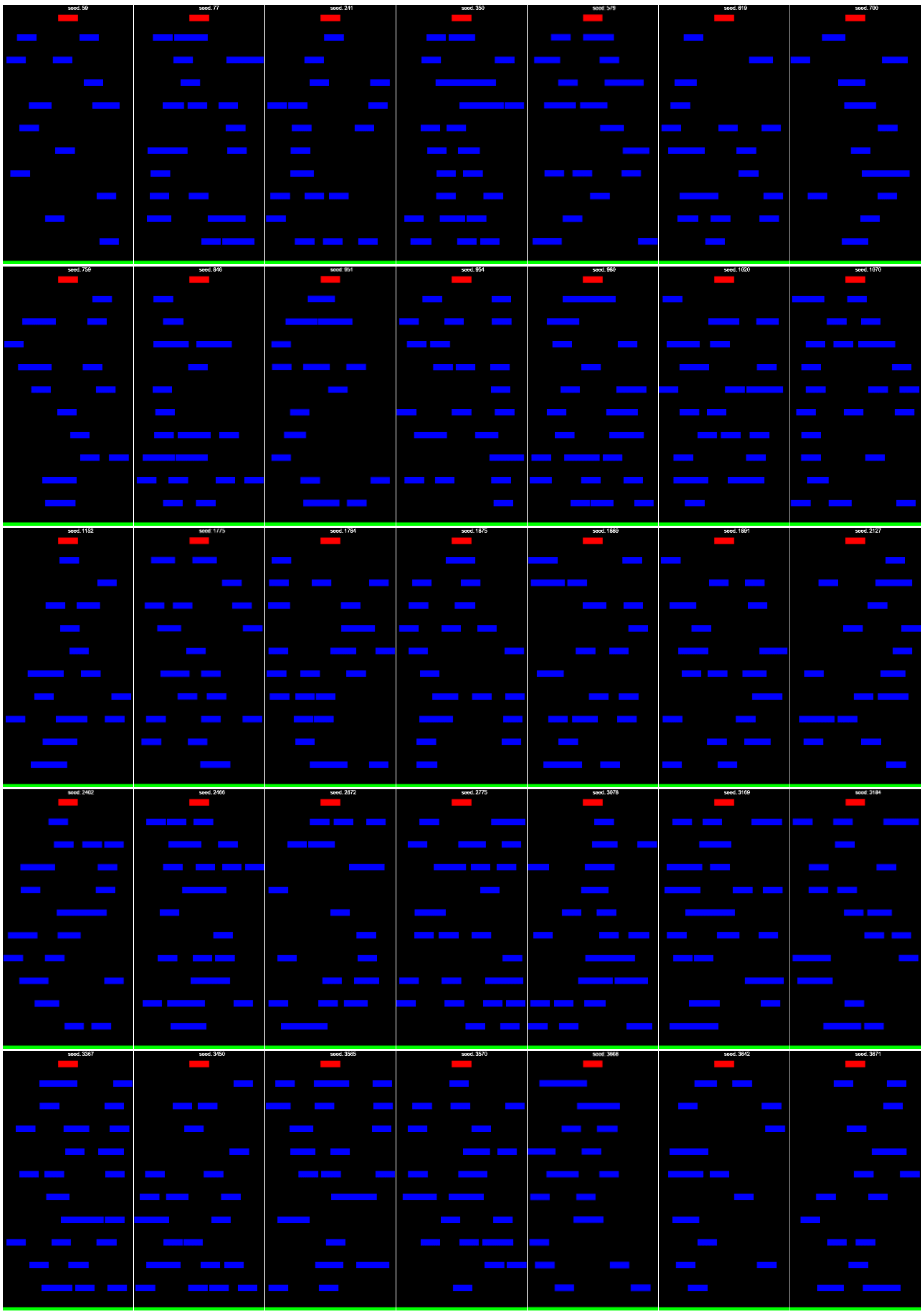
[4] SethBling, (2015) *MarI/O – Machine Learning for Video Games*, https://www.youtube.com/watch?v=qv6UVOQ0F44

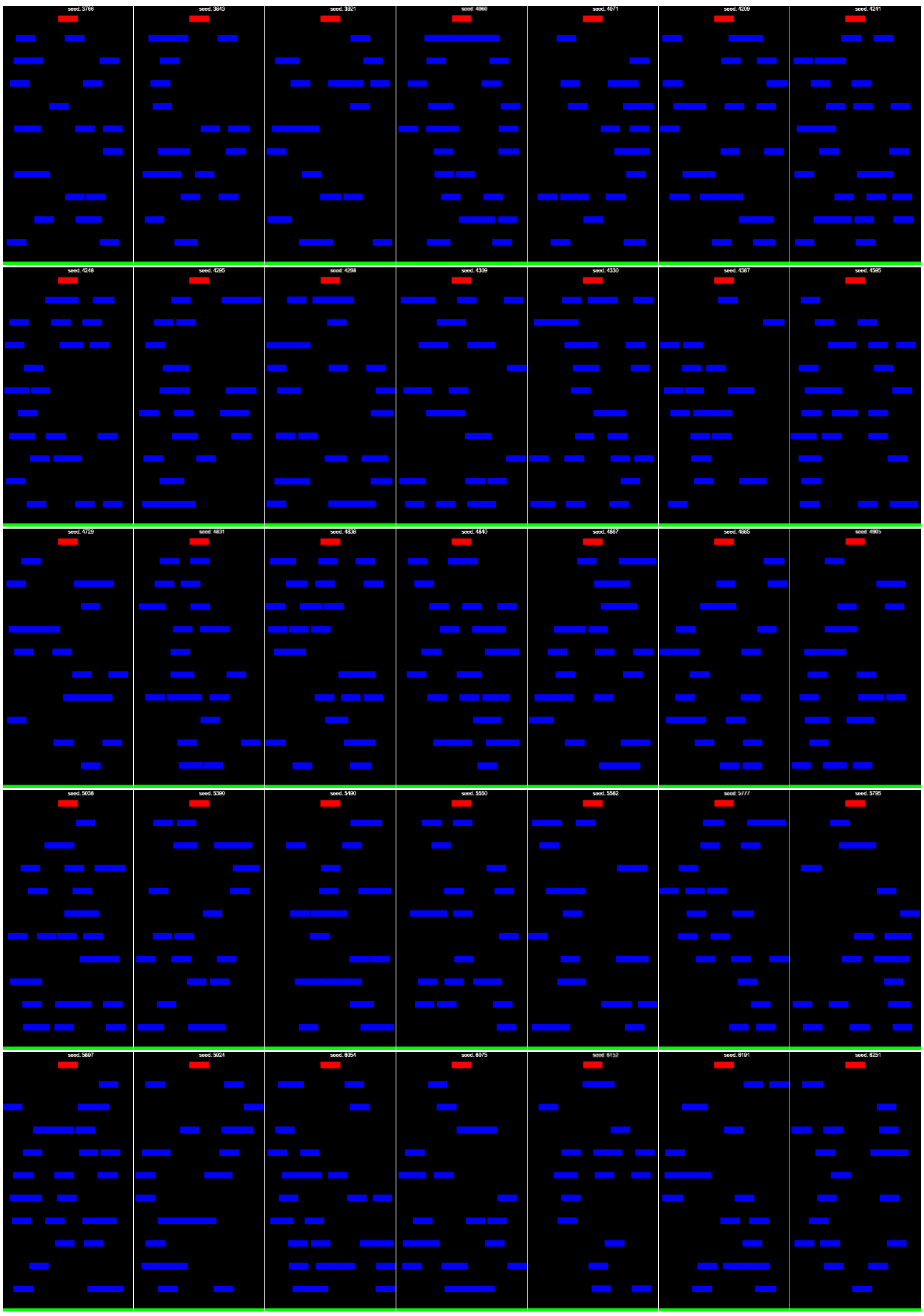
[5] Hart, P. E.; Nilsson, N. J.; Raphael, B. (1968). *"A Formal Basis for the Heuristic Determination of Minimum Cost Paths"*. IEEE Transactions on Systems Science and Cybernetics. **4** (2): 100–107. doi:*10.1109/TSSC.1968.300136*.

[6] Wildan Budiawan, Z., Diena Rauda, R., Jumadi, Rijal Muharom, Y., & Irfan, M. (2021). *The comparison of steepest ascent hill climbing and A-star for Classic game*. IEEE Xplore. Retrieved May 10, 2022, from https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9678428

[7] Iskandar, U. A., Diah, N. M., & Ismail, M. (2020). Identifying artificial intelligence pathfinding algorithms for Platformer Games. *2020 IEEE International Conference on Automatic Control and Intelligent Systems (I2CACIS).* https://doi.org/10.1109/i2cacis49202.2020.9140177

[8]


seed: 59
seed: 77
seed: 241
seed: 350
seed: 579
seed: 619
seed: 700
seed: 759
seed: 846
seed: 951
seed: 954
seed: 960
seed: 1020
seed: 1070
seed: 1152
seed: 1775
seed: 1784
seed: 1875
seed: 1889
seed: 1891
seed: 2127
seed: 2462
seed: 2466
seed: 2572
seed: 2775
seed: 3076
seed: 3169
seed: 3184
seed: 3367
seed: 3450
seed: 3565
seed: 3570
seed: 3608
seed: 3642
seed: 3671


seed: 3766
seed: 3843
seed: 3921
seed: 4060
seed: 4071
seed: 4209
seed: 4241
seed: 4248
seed: 4295
seed: 4298
seed: 4309
seed: 4330
seed: 4367
seed: 4595
seed: 4729
seed: 4831
seed: 4836
seed: 4840
seed: 4867
seed: 4885
seed: 4905
seed: 5036
seed: 5390
seed: 5490
seed: 5550
seed: 5562
seed: 5777
seed: 5795
seed: 5897
seed: 5924
seed: 6054
seed: 6075
seed: 6152
seed: 6191
seed: 6251

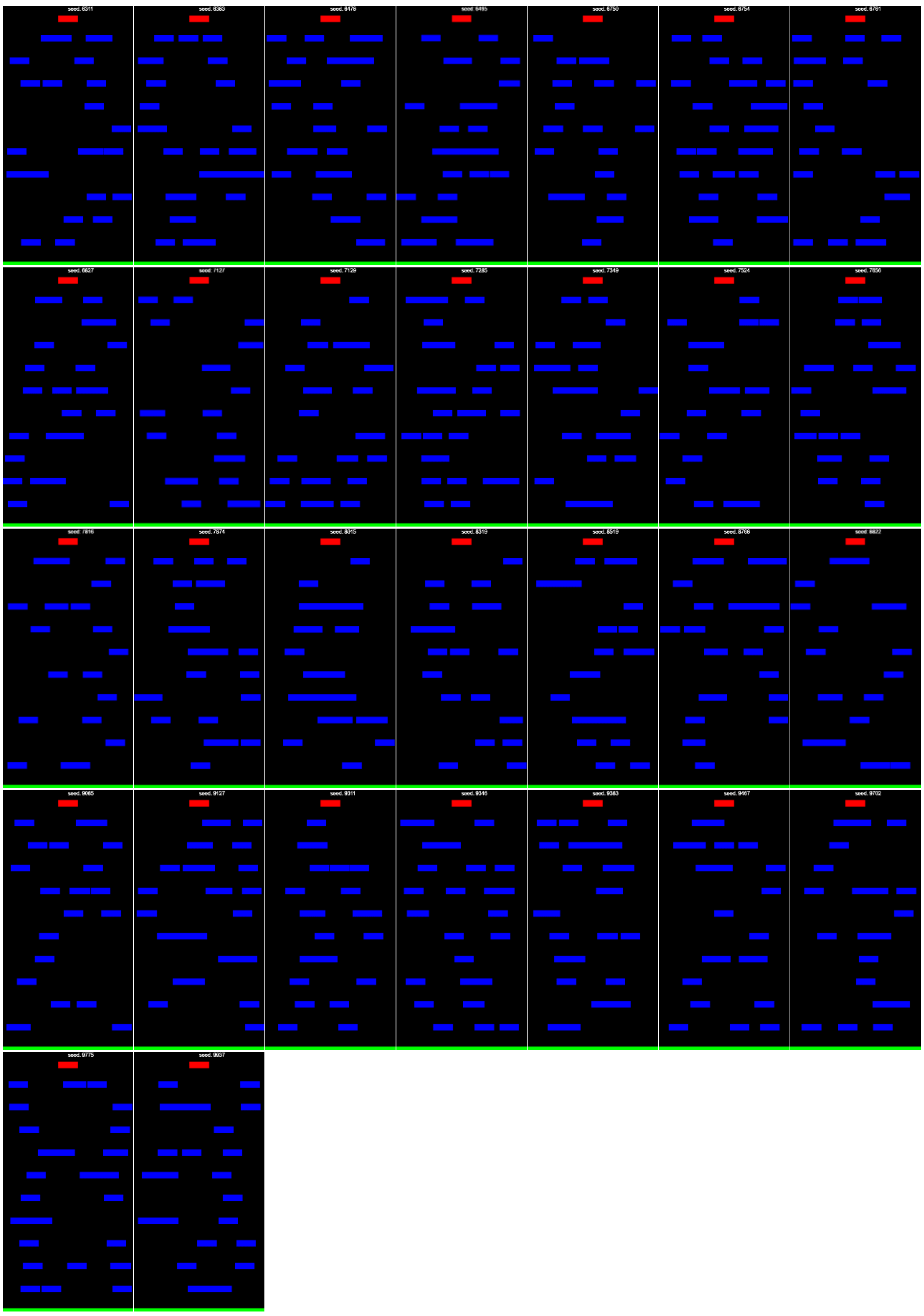
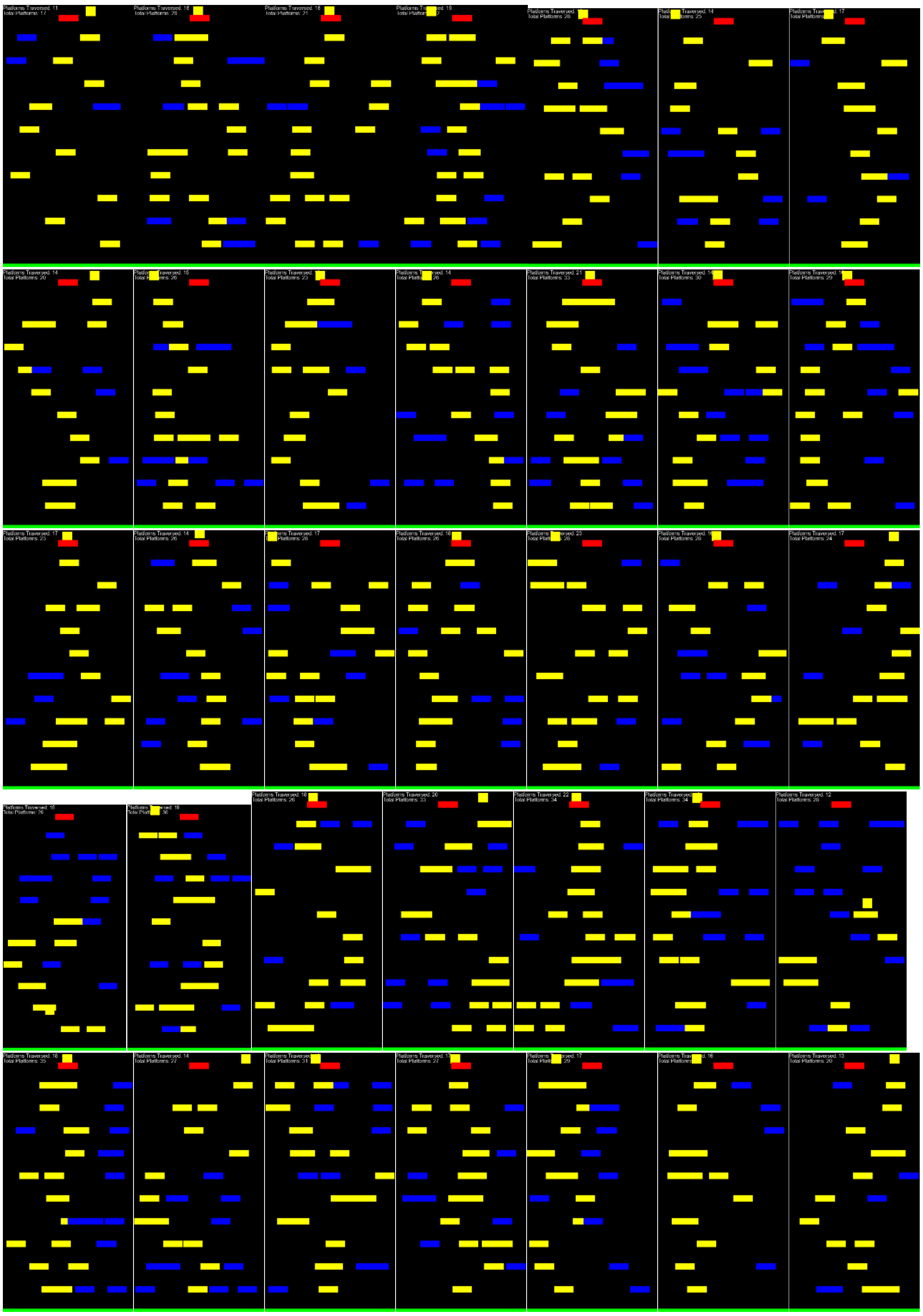
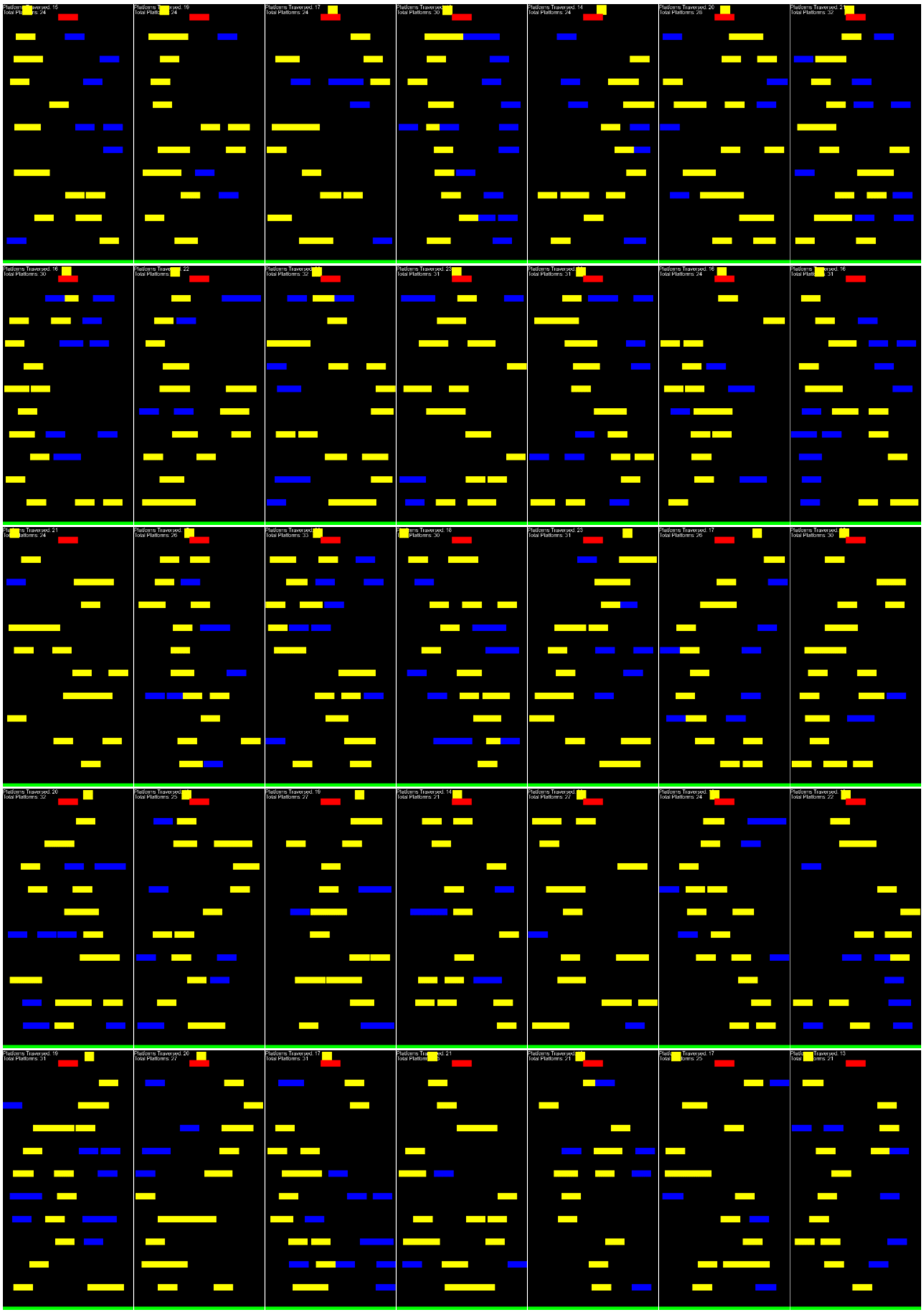


Figure 8.1: All 100 seeded random generated environments


Platforms Traversed: 11
Total Platforms: 17
Platforms Traversed: 16
Total Platforms: 28
Platforms Traversed: 16
Total Platforms: 21
Total Platforms: 28
Platforms Traversed: 14
Platforms Traversed: 14
Total Platforms: 20
Platforms Traversed: 15
Total Platforms: 23
Platforms Traversed: 14
Platforms Traversed: 21
Total Platforms: 33
Total Platforms: 30
Total Platforms: 29
Platforms Traversed: 17
Total Platforms: 23
Platforms Traversed: 14
Total Platforms: 26
Platforms Traversed: 17
Platforms Traversed: 18
Total Platforms: 26
Total Platforms: 28
Platforms Traversed: 17
Total Platforms: 24
Platforms Traversed: 15
Total Platforms: 29
Platforms Traversed: 18
Total Platforms: 26
Platforms Traversed: 20
Total Platforms: 33
Platforms Traversed: 22
Total Platforms: 34
Total Platforms: 34
Platforms Traversed: 12
Total Platforms: 28
Platforms Traversed: 18
Total Platforms: 35
Platforms Traversed: 14
Total Platforms: 27
Total Platforms: 31
Total Platforms: 27
Platforms Traversed: 13
Total Platforms: 20


Platforms Traversed: 17
Total Platforms: 24
Total Platforms: 30
Platforms Traversed: 14
Total Platforms: 24
Platforms Traversed: 20
Total Platforms: 28
Total Platforms: 32
Platforms Traversed: 16
Total Platforms: 30
Total Platforms: 32
Platforms Traversed: 23
Total Platforms: 31
Total Platforms: 31
Platforms Traversed: 16
Total Platforms: 24
Total Platforms: 26
Total Platforms: 33
Platforms Traversed: 23
Total Platforms: 31
Platforms Traversed: 17
Total Platforms: 26
Total Platforms: 30
Platforms Traversed: 20
Total Platforms: 32
Total Platforms: 25
Platforms Traversed: 19
Total Platforms: 27
Platforms Traversed: 14
Total Platforms: 21
Total Platforms: 27
Total Platforms: 24
Total Platforms: 22
Platforms Traversed: 19
Total Platforms: 31
Platforms Traversed: 20
Total Platforms: 27
Platforms Traversed: 17
Total Platforms: 31
Total Platforms: 21

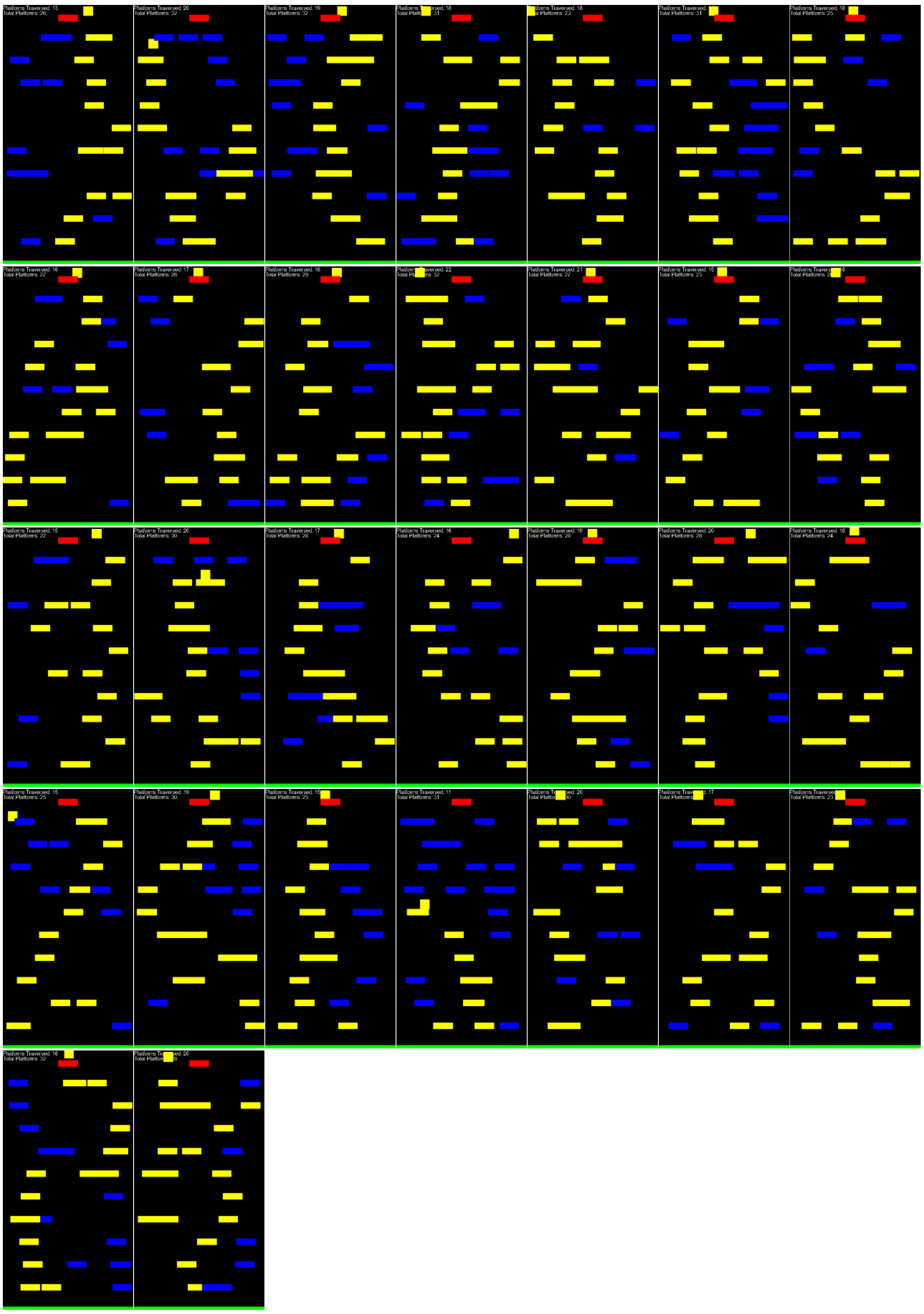
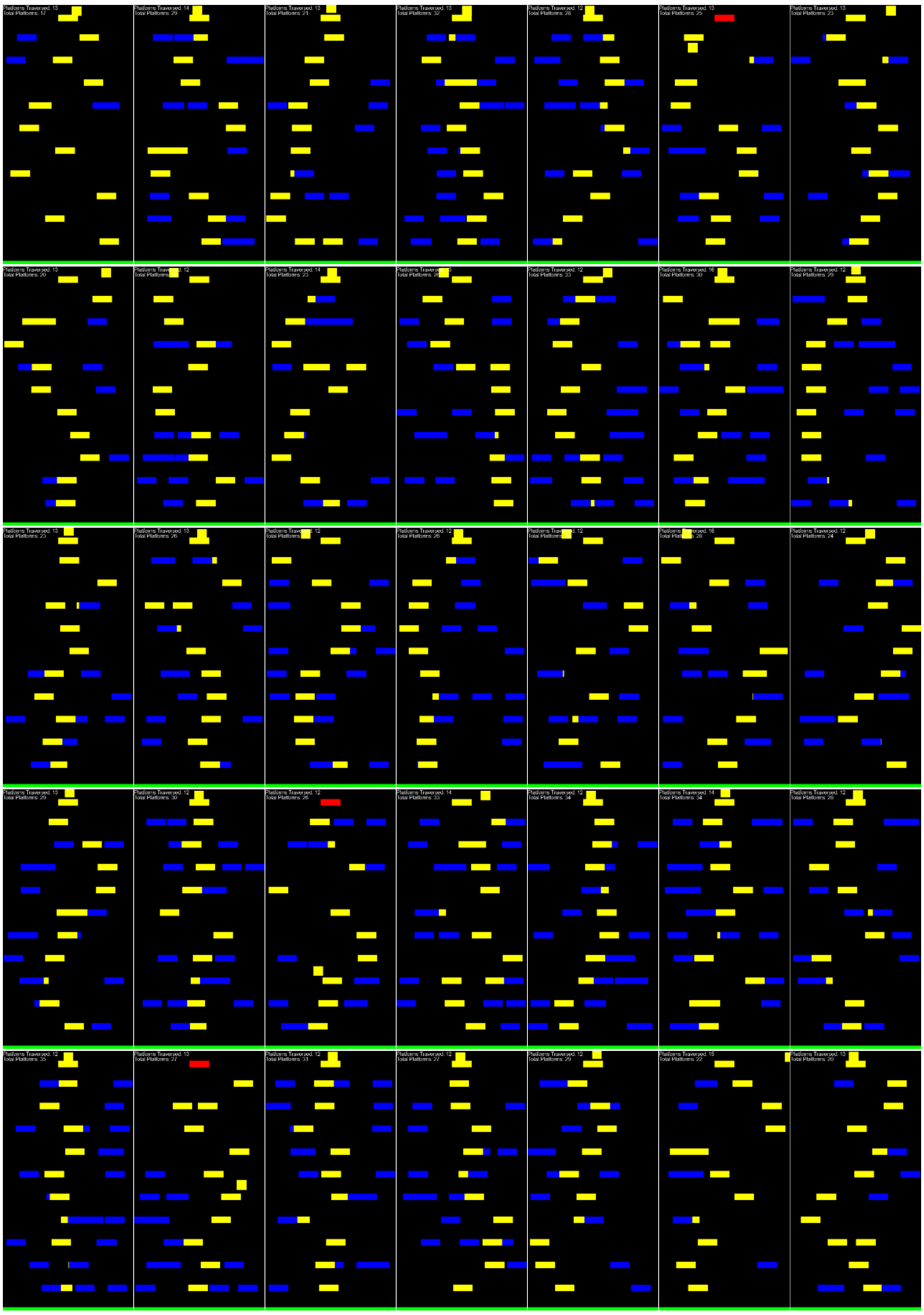
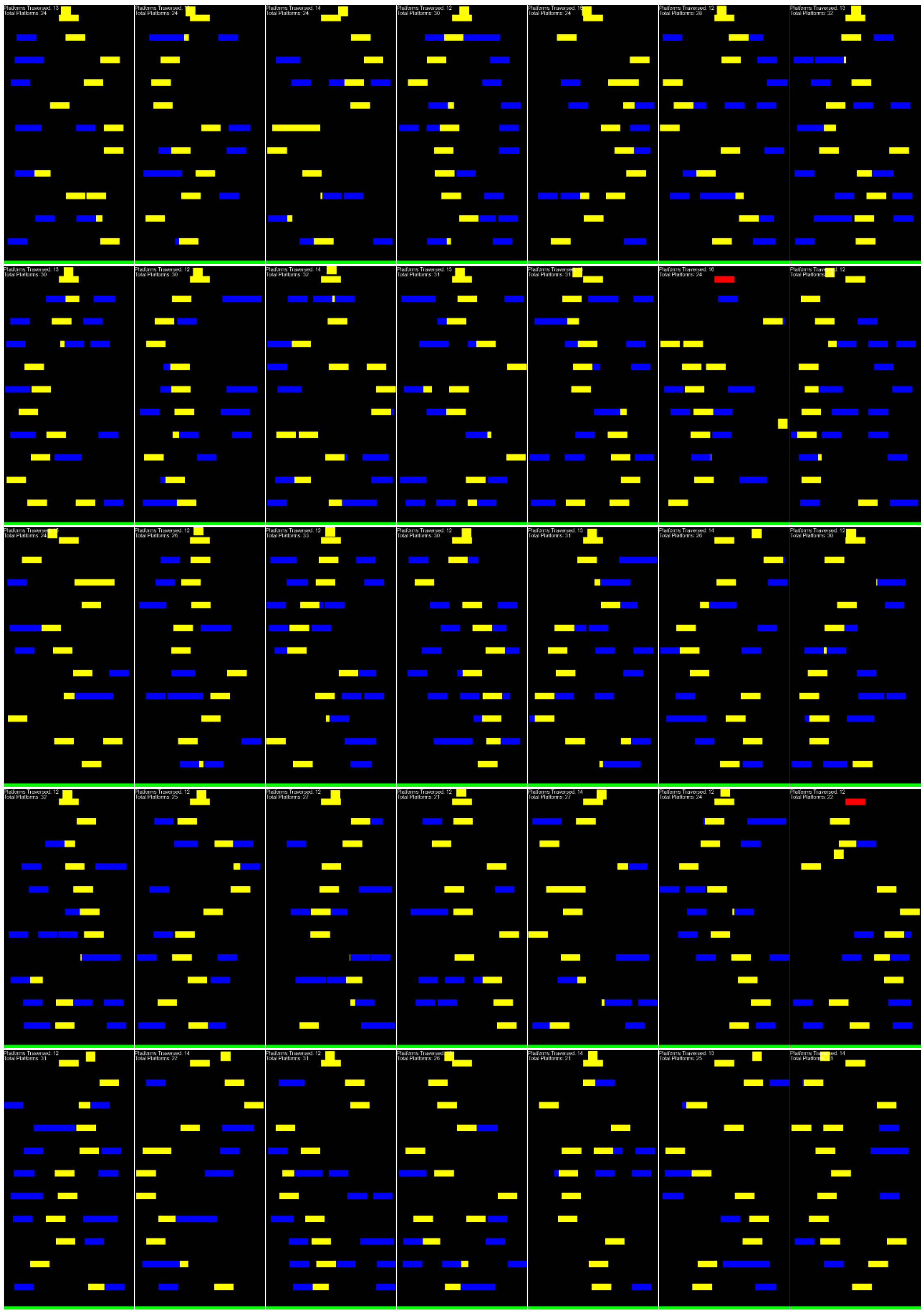


Figure 8.2: Agent traversing through 100 iterations of randomly generated environments using the greedy search algorithm


Platforms Traversed: 13 Total Platforms: 17
Platforms Traversed: 14 Total Platforms: 29
Platforms Traversed: 13 Total Platforms: 21
Platforms Traversed: 13 Total Platforms: 32
Platforms Traversed: 12 Total Platforms: 28
Platforms Traversed: 13 Total Platforms: 25
Platforms Traversed: 13 Total Platforms: 23
Platforms Traversed: 13 Total Platforms: 20
Platforms Traversed: 12 Total Platforms
Platforms Traversed: 14 Total Platforms: 23
Platforms Travers Total Platforms: 26
Platforms Traversed: 12 Total Platforms: 33
Platforms Traversed: 16 Total Platforms: 30
Platforms Traversed: 12 Total Platforms: 29
Platforms Traversed: 13 Total Platforms: 23
Platforms Traversed: 13 Total Platforms: 26
Platforms Traversed: 12 Total Platforms
Platforms Traversed: 12 Total Platforms: 26
Platforms Traversed: 12 Total Platforms
Platforms Traversed: 18 Total Platforms: 28
Platforms Traversed: 12 Total Platforms: 24
Platforms Traversed: 13 Total Platforms: 29
Platforms Traversed: 12 Total Platforms: 30
Platforms Traversed: 12 Total Platforms: 26
Platforms Traversed: 14 Total Platforms: 33
Platforms Traversed: 12 Total Platforms: 34
Platforms Traversed: 14 Total Platforms: 34
Platforms Traversed: 12 Total Platforms: 28
Platforms Traversed: 12 Total Platforms: 35
Platforms Traversed: 13 Total Platforms: 27
Platforms Traversed: 12 Total Platforms: 31
Platforms Traversed: 12 Total Platforms: 27
Platforms Traversed: 12 Total Platforms: 29
Platforms Traversed: 15 Total Platforms: 22
Platforms Traversed: 13 Total Platforms: 20



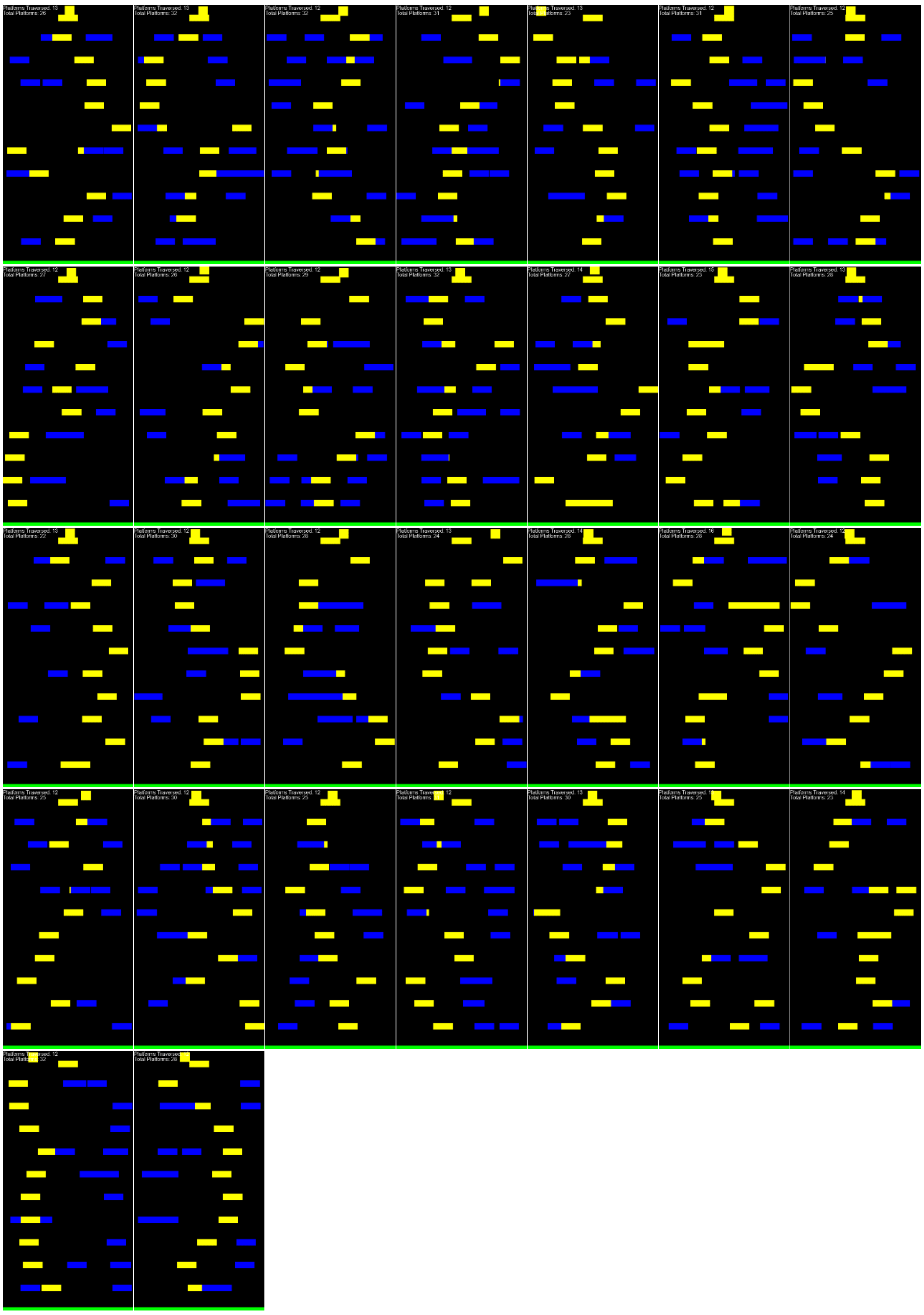


Figure 8.2: Agent traversing through 100 iterations of randomly generated environments using the A* search algorithm