Star Battle: Generation, Difficulty Analysis, and Heuristic learning

Individual Project

Sophia Williams
ID: 246549
smjw21@sussex.ac.uk

April 2025

School of Informatics
University of Sussex

Submitted in partial fulfilment of the requirements for the Degree of BSc Computer Science and Artificial Intelligence

Supervisor Dr. Vincent van Oostrom

Word Count: 9672

Statement of Originality

This report is submitted as part requirement for the BSc in Computer Science and Artificial Intelligence at the University of Sussex. It is the product of my own labour except where indicated in the text. The report may be freely copied and distributed provided the source is acknowledged.

- Sophia Williams

Professional Considerations

Since project involves generation, assessment and machine learning based on a logic puzzle, there are no major ethical issues to be considered. The researcher has ensured they were knowledgeable on the contents of the Chartered Institute of IT's Code of Conduct and will abide by all sections relevant to any IT project.

Abstract

This project explores the generation, difficulty assessment, and solution of Star Battle puzzles using both algorithmic and machine learning approaches. By formalising the puzzle as a constraint satisfaction problem (CSP), we investigate rule-based techniques for classifying puzzle difficulty and generating valid, uniquely solvable puzzles. A convolutional neural network (CNN) is trained to predict star placements, and interpretability methods such as Grad-CAM are applied to understand learned heuristics. This project aims to combine human-intuitive strategies with data-driven learning, contributing to puzzle research and adaptive game design.

Keywords— Puzzles, generation, difficulty, heuristics, constraint satisfaction, combinatorics

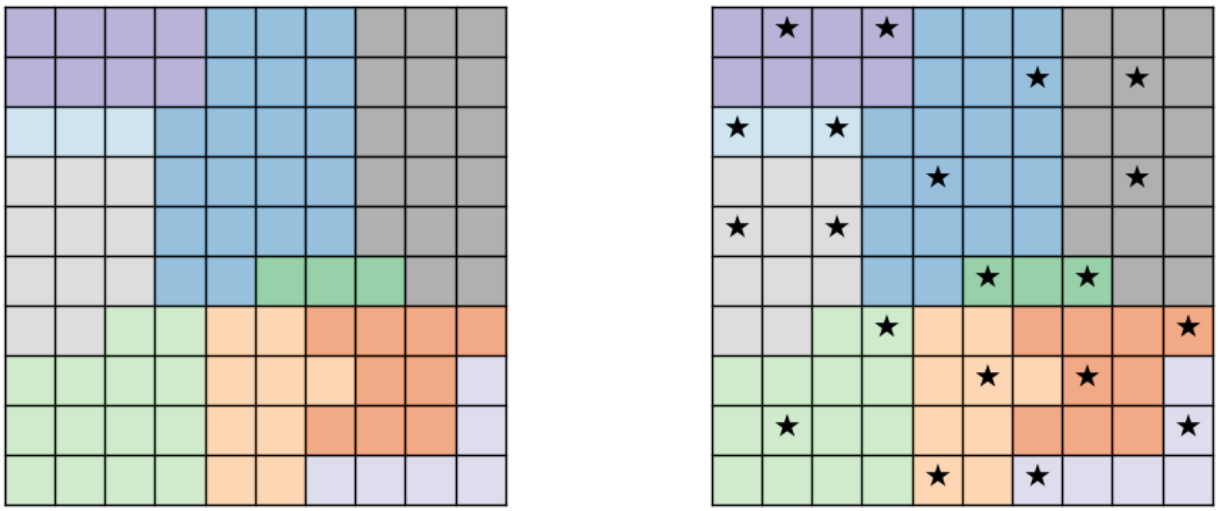
Acknowledgements

I would like to thank Dr. Vincent van Oostrom, my supervisor, for supporting my dissertation. I would also like to thank Jim Bumgardner, as his `math.stackexchange.com` post saved my life when I was losing my mind over valid puzzle generating :).

Contents

1 Introduction

Star Battle is a logic puzzle where the objective is to place a specified number of stars in each row, column, and defined region, while ensuring no two stars are adjacent, diagonally or orthogonally (Figure 1). While the rule set is simple, the number of valid configurations grows combinatorially with grid size, making larger puzzles significantly harder to solve both for humans and algorithms (Laszlo and Mukherjee, 2023). The puzzle has become popular due to its accessibility to beginners and complex solutions, appealing to advanced players, making it an ideal puzzle for computational and algorithmic exploration. The rules of Star Battle make it an interesting constraint satisfaction problem (CSP), a type of mathematical problem that involves finding values for a set of variables, such that any constraints on the variables are satisfied.



(a) A 10x10 Star Battle Puzzle

(b) The same puzzle with a valid solution

Figure 1: A valid 10x10 Star Battle puzzle and its solution

Similar logic puzzles like Sudoku, Nonogram and Kakuro share a foundation with Star Battle as they share a Latin square; an NxN grid. Star Battle's rule constraints introduce a unique combinatorial challenge, making it an intriguing area of study into constraint satisfaction and optimisation. This project uses the complexity of the puzzle to explore generation, difficulty assessment, and solving techniques.

Additionally, it aims to examine whether solving heuristics can be learned automatically using machine learning techniques. Machine learning is the use and development of computer systems that can learn and adapt without following explicit instructions. It draws on algorithms and statistical models to identify patterns in data and make decisions or predictions based on them (Mirshekarian and Sormaz, 2018). In this project, we apply machine learning to Star Battle puzzles with two main goals: first, to gain deeper insight into the effectiveness and intuition behind the heuristic techniques already widely used in solving these puzzles; and second, to explore whether machine learning can find more advanced or subtle solving strategies that may not have been captured through manual rule design. By doing so, we aim to learn from the data-driven approaches to puzzle solving.

1.1 Problem Area

The challenge in Star Battle lies in its constraints. For a Star Battle instance to be considered valid, each row, column and region must contain a specified number of stars that cannot be adjacent. This rule set turns Star Battle into a puzzle requiring consideration of combinatorial decision and spatial reasoning. Before placing a star, one must consider the effect on surrounding regions, rows and columns. Generating, solving and analysing these puzzles at various difficulties is complex. Small changes in the puzzle structure can lead to significant changes in complexity. The puzzle's design, with its overlapping constraints on rows, columns and irregular regions, contribute to the difficulty of automation. Unlike simpler logic puzzles, valid placements in Star Battle must simultaneously satisfy multiple constraints across the entire grid, and small changes in any given area can have cascading effects on the rest of the puzzle. This interdependency makes automating the process of generating, difficulty assessment and solving particularly difficult.

1.2 Formal Definition

A more formal description of Star Battle puzzles is as follows. Star Battle is a constraint satisfaction puzzle played on a 2D grid, where each cell belongs to exactly one region (Laszlo and Mukherjee, 2023). The goal is to place a fixed number of stars per row, per column and per region such that:

Figure 1.2: Star Battle Puzzle Formal Specification

Rules:

- No two stars are adjacent, diagonally or orthogonally.
- Each row, column, and region must contain exactly k stars, where $k \in \{1, 2, 3\}$.

Puzzle Definition:

- The puzzle is defined on an $N \times N$ rectangular grid.
- The grid is partitioned into regions $R = \{r_1, r_2, \ldots, r_N\}$, where each $r_i \subseteq C$ and $C = \{(i, j) \mid 0 \leq i, j < N\}$.
- Each region is non-empty, disjoint, and all regions together cover the entire grid.
- A fixed positive integer k specifies the number of stars per row, column, and region.

Solution Representation:

- Each cell (i, j) contains a Boolean variable $S(i, j) \in \{0, 1\}$, where $S(i, j) = 1$ indicates a star.

Constraints:

- *Row constraint:* For every row $i \in [1, N]$,
$$\sum_{j=1}^{N} S(i, j) = k$$
- *Column constraint:* For every column $j \in [1, N]$,
$$\sum_{i=1}^{N} S(i, j) = k$$
- *Region constraint:* For each region $r \in R$,
$$\sum_{(i,j) \in r} S(i, j) = k$$
- *Adjacency constraint:* No two stars may be adjacent. Define the 8-neighborhood:
$$N(i, j) = \{(x, y) \in C \mid \max(|x - i|, |y - j|) = 1\}$$
The neighbourhood around a cell (i, j) is defined as an element in the set of all Cells that has a distance from the cell $(i, j) = 1$ (adjacent).

The puzzles generated will need a single solution to be considered valid. Understanding the properties that contribute to a puzzle's difficulty and exploring ways to generate and solve these puzzles algorithmically are non-trivial tasks. Using machine learning to explore human heuristics may prove challenging as they need to be presented in a way understandable outside of a computational perspective, allowing the new techniques to be applied by human solvers.

This project looks to address what makes a Star Battle puzzle challenging, how puzzles can be efficiently generated with corresponding difficulty levels, and whether solving strategies can be learned automatically. By analysing Star Battle puzzles through combinatorial and machine learning techniques, the project aims to contribute to a clearer understanding of the structure and solving methods for Star Battle puzzles and CSPs in general. By reviewing the results of the project, we can hopefully gain insights into puzzling, with potential applications in fields like game design, AI problem-solving and algorithm optimisation.

This report assumes a basic familiarity with machine learning techniques, particularly supervised learning and convolutional neural networks (CNNs). However, to support accessibility a brief overview of relevant concepts is included throughout. Readers unfamiliar with these methods may benefit from introductory resources such as Alpayadin's Introduction to Machine Learning (Alpaydin, 2020), which provides foundational knowledge on neural networks, training frameworks, and evaluation strategies. Where more advanced techniques are referenced, appropriate context or citations are provided to ensure clarity.

2 Project Objectives

The project aims to explore generating and solving Star Battle puzzles using techniques and insights from common solving strategies. The aim is to automate the puzzle development and solving process. We then look to solve the puzzles using both encoded solving techniques (observed when solving the puzzles by hand) and to use machine learning techniques observe if common techniques can be learnt, or if new techniques are found.

2.1 Primary Objectives

2.1.1 Generate puzzles in a complete process

We aim to develop a full process for generation of valid Star Battle puzzles, ending with uniquely solvable puzzles. This involves ensuring region contiguity, adhering to puzzle constraints, and verifying each puzzle has a single solution using constraint-based solvers (e.g., SAT or SMT solvers). The objective is to build a reliable and reproducible system capable of creating puzzles of varying difficulty.

2.1.2 Determine the difficulty of solving a given puzzle

Create a framework for assessing puzzle difficulty based on human-like heuristics. This includes implementing a set of production-rule solvers that encode the logical deduction techniques used by experienced puzzle solvers. The system should classify puzzles into categories such as "Normal", "Hard" and "Extreme" by simulating the reasoning process and observing which rules were required to reach a solution.

2.1.3 Assess automatic technique learning approaches

Explore the application of machine learning models such as CNNs to learn solving heuristics directly from puzzle data. This includes training models to predict star placements or puzzles states, interpreting what strategies the models have learned (e.g. through Grad-CAM heatmaps), comparing outputs to human-like strategies. The goal is to evaluate how well these models can learn or extend solving logic and whether they can highlight deduction patterns.

2.2 Extensions

The following are extensions building on the core objectives of the project. These components aim to enhance the practical utility and accessibility of the system, particularly for researchers and advanced users.

2.2.1 Develop an interface of puzzling and evaluation

Create a lightweight, user-friendly interface to allow researcher and testers to interact with the puzzles directly. This tool will support exploration of the data such as evaluation of difficulty classifications, or evaluation of machine learning techniques. The goal is to consolidate insights from the puzzle generation and analysis in one environment, supporting real time experimentation and visualisation.

2.2.2 Create an automated hinting system

Design a system capable of helping solvers based on deduction techniques used. Hints should reflect human reasoning rather than brute force approaches, with varying levels of assistance.

2.3 Expected outcome

The expected outcome of this project is a complete system for the generation, analysis, and solving of Star Battle puzzles. The outcomes aim to advance both practical and theoretical understanding of constraint-based puzzle solving.

3 Motivations

The motivations for this project come from a personal interest in logic-based puzzling and algorithm design. The draw of a new puzzle is enticing to problem solvers, and the challenge posed by harder puzzles incites a sense of accomplishment when solved. Puzzling provides interesting ground for academic research in areas like studying CSPs, machine learning and combinatorics due to their mathematical properties. Star Battle stands out as a modern puzzle; a Latin grid, cell values containing only one of two states, irregular region shapes. These features make the puzzle unique, making it an insightful exploration project. As well as this Star Battle puzzles are less explored academically than classic logic puzzles like Sudoku or Kakuro, meaning there is likely untapped potential for insights in combinatorics and optimisation techniques.

The structure makes it ideal for studying constraint propagation and combinatorial reasoning in a given space. The rules seem simple, but by formalising the problem we can observe a deep complexity, forming a compelling angle for research. Its rising popularity online and in puzzler communities is a testament to its allure, but lacks systematic academic attention compared to puzzles like Sudoku. This could be for several reasons. For example, the may be considered hard to generalise mathematically as they are highly dependent on their region layout. This irregularity across the puzzles make it seemingly more complicated for generalised mathematical analysis. The lack of formal literature on the puzzle also makes it a challenge as frameworks, benchmarks and academic repositories are not widely available, leading researcher to need to develop these from scratch.

There is opportunity to create a novel system. A complete repository for the generation, assessment and solving of Star Battle puzzles. Current generation techniques are fairly random, lacking insight and adaptability. There is room for improvement involving optimisation and data driven approaches to generation and solving of these puzzles.

There is a particular excitement to potentially finding new solving techniques, improving general knowledge of solving methodologies for new and advanced players alike.

The project allows practical application of knowledge in python, algorithms, and ML in a meaningful, interdisciplinary context. The determining of features about a given puzzle, understanding constraint satisfaction elements, development of machine learning models all involve the culmination of knowledge from multiple years of academic pursuit. The project also provides opportunity for hands-on experience with common tools such as NumPy, TensorFlow, and scikit-learn in a project that is both of personal enjoyment and technically rich.

4 Hypothesis

The project focuses on several key research areas. We hypothesise that the generation of valid Star Battle puzzles can be done in a lightweight algorithm, providing a set of unique puzzles at varying difficulties. We expect that our difficulty estimation algorithm can successfully classify Star Battle puzzles into predefined categories, and that this will support the human intuition methods used by human solvers. We hypothesise that applying machine learning to Star Battle puzzles will uncover new patterns and heuristics that may not be explicitly defined in existing algorithms, leading to more efficient puzzle generation and solving strategies.

5 Results

This project successfully generated a dataset of 2,500 Star Battle puzzles and classified their difficulty levels. Using rule-based heuristics, the puzzles were categorized into levels with a focus on pattern recognition techniques at varying intuition levels. A machine learning model was then developed to further classify these puzzles, achieving an overall accuracy of 81%. Additionally, Grad-CAM visualizations revealed that the model successfully identified human-like solving patterns, highlighting its potential for pattern-based decision-making. Future work will focus on improving the puzzle classification efficiency and refining the model's overall performance.

6 Review of previous work

6.1 Constraint Satisfaction Problems

Constraint Satisfaction problems are mathematical problems that involve assigning values to variables within a domain with some kind of constraint; a set of conditions that must be satisfied by variables. Logic puzzles like Star Battle, Sudoku, Kakuro, and more are examples of CSPs as they have single constraints that lead to a single valid solution. The significance of CSPs in puzzles is in their flexibility in defining both the structure of the puzzle and potential solving strategies. Approaches such as backtracking, constraint propagation, and local search each offer different trade-offs in terms of completeness and efficiency, especially as puzzles grow in complexity. These foundational concepts are well-established in constraint-solving literature (Brailsford et al., 1999).

Recognised techniques for solving CSPs involve backtracking, constraint propagation or local search, or potentially combinations of these techniques. Backtracking, in the context of CSPs, is a depth first search method used to explore possible solutions to a given problem. It assigns values to variable and backtracks if an assignment violates a constraint. It is a recursive algorithm that assigns values to variable until there is a violation, continuing until either a solution is found, or all possibilities yield a violation. Due to its simplicity, backtracking is widely used, but can become inefficient on larger, more complex problems.

Constraint propagation techniques use subsets of the problem in order to simplify the task. It looks at a form of local consistency. The algorithm may apply the constraints to individual cells and then update surrounding possibilities. As it takes each subset and applies the constraints of the problem it further reduces the options for variables in the problem as a whole.

Local Search starts with an initial guess and then iteratively improves on the guess by making small adjustments. It improves upon this initial solution (even if it is invalid) by reducing the number of violating constraints, depending on how exactly the puzzle is formulated. The main concern with Local Search as a CSP technique is it can get stuck in a local optimum, failing to completely find a valid solution.

An important concept in CSPs is Arc Consistency. This looks at simplifying problems by eliminating values that violate the constraints and so cannot be in the final solution. This is key in Star Battle as when finding a solution, a solver will naturally eliminate spaces that cannot contain a star, narrowing down possible options.

This project will use Satisfiability (SAT) Solving in several aspects, from validating generated Star Battle puzzles to determining difficulty of our test set of puzzles. The Satisfiability problem is looking at whether a given formula is satisfiable, whether there exists a variable assignment that satisfies the constraints of the problem (Barrett and Tinelli, 2018). This applies quite well to our formal definition of the Star Battle puzzle, as the puzzle can be formalised using propositional logic as well, although this approach requires a transformation to define only in Boolean states.

6.2 Combinatorics in puzzle generation and analysis

Combinatorics is a branch of mathematics that looks at counting, arranging and combining objects. It helps to understand how to determine the number of possible arrangements, combinations, permutations within a given problem.

n	Plane: $N(n)$
8	2
9	664
10	146,510
11	31,197,434
12	6,798,881,226
13	1,567,979,086,356
14	390,673,279,156,006
15	106,280,659,533,411,296

Figure 2: Number of valid puzzles for a grid size n

Combinatorics is incredibly helpful in the Star Battle generation due to its foundations in design theory. Combinatorics supports designing structures that meet specific criteria, such as the constraints in Star Battle. Combinatorial optimisation may also be helpful in this project as understanding the combinatorial structure of a given puzzle can avoid redundant or impossible configurations, giving our algorithms the ability to make a much more informed decision, reducing the search space (Wallis and George, 2016).

Some more practical aspects of combinatorics can involve improvements in pattern recognition that can guide solving strategies for human solvers. Using combinatorial methods can also aid to estimate difficulty of a given puzzle. Assisting classification of a puzzle.

6.3 Star Battle Configurations

In a paper by Michael Laszlo and Sumitra Mukherjee, an algorithm was developed to find a total number of Star Battle puzzles on a given grid size N. By calculating the total number of valid puzzles, the paper finds insights into the combinatorial explosion; how quickly complexity increases with size. Their method highlights how the number of valid configurations grows rapidly with grid size and offers a robust model for exploring the space of valid puzzles while maintaining constraints such as adjacency and column balance. This aligns closely with the challenges faced in this project when evaluating grid scalability and search space reduction.

The paper outlines a number of core representations that define the constraints of a puzzle including the row, column, adjacency and binary constraints (Figure 2). It also defines a column count vector (CCV). This is an array that tracks the number of stars in each column, ensuring there are no more than 2. The paper also introduces the concepts of a path, a profile and a bundle. A path looks at a sequence of rows, a profile is a pair of a path's CCV and the last row in the path, and a bundle is a collection of all paths with the same profile. The algorithm then looks at building these bundles and matching them iteratively to calculate the total number of configurations by summing the size of the matched bundles.

The technique used in the algorithm is a dynamic programming approach combined with state-space search. The algorithm explores all the states and expands them step-by-step, ensuring the puzzle's constraints remain satisfied. The algorithm was run on a number of different grid sizes and the results recorded (Figure 2). The paper highlights a number of intriguing findings, namely the number of Star Battle puzzles growing rapidly with the increased grid size. Furthermore, splitting the problem up into the profiles an then bundles improves the efficiency, breaking down the larger task into manageable subtasks.

The findings of the paper are crucial to this report, as they reiterate the importance of configurations of the Star Battle puzzle having unique and solvable solutions (Laszlo and Mukherjee, 2023). As the paper only looks at finding a total for the Star Battle puzzle instances, it does not go further into the constraints posed by the regions present in Star Battle puzzles, which would be interesting to look into as further research.

6.4 Difficulty Estimation in Puzzles

In a paper by Radek Pelanek, a system to understand and evaluate the difficulty of sudoku puzzles was built. It evaluates factors influencing the difficulty of these puzzles and outlines methods of quantifying the difficulty. As Star Battle is a similar logic puzzle, the paper is valuable for understanding difficulty assessment. The paper identifies two main factors affecting the difficulty of a given puzzle: the complexity of individual steps and the dependency between steps. The former refers to the difficulty of any one logical operation required to solve a puzzle, while the latter looks at how individual steps in the solution process are dependent on one another. Puzzles that are highly dependent need to be solved in a specific sequence and so are harder to solve than those with independent steps, offering multiple paths to a solution.

Pelanek's work identifies the importance of modelling both the individual logical complexity of each solving step and the way solutions are chained together. This is applicable to the Star Battle search space, where early placements of stars can dramatically limit or enable future moves. The use of refutation scoring and constraint relaxation offers a framework for algorithmic difficulty classification that can inform future Star Battle heuristic design, especially when aligning solver performance with human intuition (Pelánek, 2014).

The paper describes several metrics for evaluating the difficulty of sudoku, focusing on those that most closely align with human problem-solving techniques. Algorithmic-based techniques were tested such as backtracking search and harmony search. While these are solid computational methods to solving sudoku, they do not strongly correspond with human solving strategies. Humans are more likely to avoid systematic searching in favour of deduction and logical reasoning. Another approach tested is constraint relaxation. This is where constraints are removed from a puzzle and the number of solutions is then analysed again. This is an interesting approach as while it also is not similar to human behaviour, it does offer meaningful predictions on difficulty.

The method closest to replicating human Sudoku solving methods involved building computational models that simulate the logical deductions and strategies that humans use. The models focus on constraint propagation, which we have previously discussed, looking at subsections of the problem. The models also use a set of logic techniques, that all sudoku players know and use by default. One key metric that arose from these models is the refutation score. This metric measures the difficulty of assigning the correct value to a cell when basic logic techniques cannot provide a simple solution. It uses a process of elimination where incorrect values are refuted assuming it would violate the sudoku constraints. These computational models were able to accurately predict the perceived difficulty of Sudoku puzzles, as well as the likely order a human would fill in the cells. The models achieved a very close alignment with human solving strategies, and so provided a detailed insight into heuristics of solving.

One limitation of the computational models used is the level of sudoku knowledge needed to design these models, as well as the reliance on detailed logic that can make this approach difficult to apply to other CSPs. Despite these limitations, the paper has provided a lot to think about in terms of the ability to use computational models in predicting puzzle difficulty and human problem-solving behaviours. The approach provides an interesting potential framework as to how assessing difficulty and learning about heuristics can be achieved for Star Battle as a CSP.

6.5 Heuristic and Machine Learning in Puzzle Solving

One interesting paper, by Benjamin Estermann, introduces a system for evaluating the reasoning capabilities of reinforcement learning (RL) algorithms. It uses logic puzzles from Simon Tatham's Portable Puzzle Collection. The paper explains the developed PUZZLES environment used to evaluate the strengths and limitations of their algorithms. It also looks to generalise the use of the models on future research. Puzzles are incredibly powerful in assessing and improving neural networks as they provide a number of key elements that promote algorithmic, logical solutions. While the heuristics used in puzzling may seem abstract, the skill learnt by these models can be applied to a wider range of important real-world tasks.

The paper covers a number of different puzzle types that present unique challenges to the RL agents. The puzzles often have their own special parameters that differentiate them from the other puzzles, giving the RL agents new constraints based on the task.

The paper describes several strategies used to train their neural models within the PUZZLES framework. It looks at different reward structures. Agents can receive a "sparse" reward or "custom" rewards. Sparse rewards describe agents receiving reward only once a puzzle is complete, encouraging agents to learn the most efficient path possible. Custom rewards are tailored for specific puzzles that guide the learning process and improve the efficiency of training. The paper also looks at masking and different observation types. These help the agents to find solutions faster as it prevents the agents from exploring redundant options. The paper also then points to the potential of future research in the PUZZLES framework. There is potential for studying graph neural networks, state-space exploration, among other interesting concepts.

The potential of logic puzzles as testbeds for machine learning is increasingly being explored, especially in reinforcement learning. Estermann's PUZZLES framework demonstrates how structured environments, diverse constraints, and incremental learning can guide agent reasoning. The use of sparse and custom rewards, masking, and tailored observations showcases how puzzles like Star Battle can be integrated into broader ML research for learning logical deduction strategies. This approach highlights how puzzle-solving not only aids neural network training but also contributes to understanding and skill transfer in AI models (Estermann et al., 2024).

The paper has examined key concepts that are applicable to Star Battle and can hopefully lead to some insight into human solving strategies. The work of this project will hopefully be able explore human solving methods through machine learning in a way that contributes to a deeper understanding of Star Battle and puzzling as a whole.

7 Puzzle Generation

7.1 Puzzle Representation

The Star Battle puzzle is played on an NxN, Latin grid with N, number of regions. The regions are random and are not required to be equal in size. Common Star Battle puzzles come in sizes of 5, 6, 8, 10, and 14, with 10x10 puzzles being the most common. To work with the puzzles in the project, we have chosen to represent them in a matrix format as a puzzle grid, showing labelled regions, and a solution grid of the same shape, showing where the stars are on the puzzle,

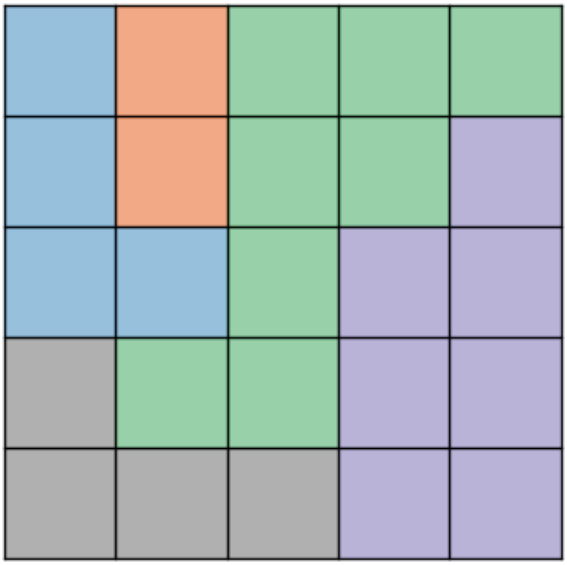


Figure 4: Output of the *puzzle* matrix from our visualiser function

represented by "1" (Figure 1). The puzzles and solutions are stored as NumPy arrays, for their simplified handling, and comprehensive library (Bressert, 2012).

Listing 1: Puzzle and solution matrices

```
puzzle = [
 ['0', '1', '2', '2', '2'],
 ['0', '1', '2', '2', '3'],
 ['0', '0', '2', '3', '3'],
 ['4', '2', '2', '3', '3'],
 ['4', '4', '4', '3', '3']
]

solution = [
 ['0', '1', '0', '0', '0'],
 ['0', '0', '0', '1', '0'],
 ['1', '0', '0', '0', '0'],
 ['0', '0', '0', '0', '1'],
 ['0', '0', '1', '0', '0']
]
```

Figure 3: 5x5 Puzzle grid and corresponding star solution represented in matrix form

To start, we implemented a simple visualiser function, using Matplotlib (Tosi, 2009). This drew the puzzle to a grid separating the regions by colour, improving readability of the results. This meant that throughout the project, we can quickly see the regions generated and roughly check the progress of the generator. (Figure).

7.2 Initial Generation Approach

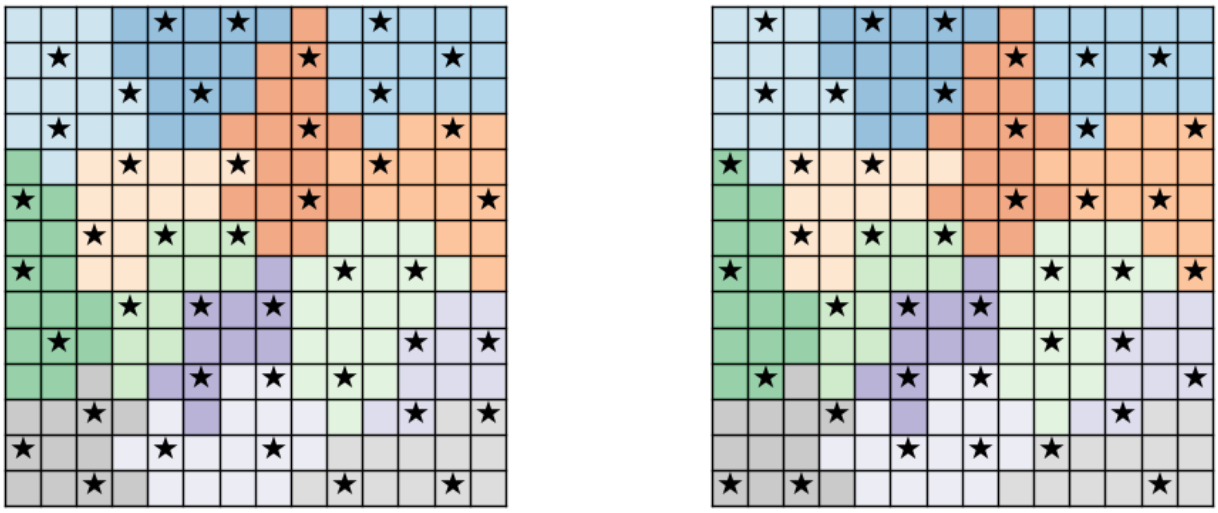
To begin, we created a function to validate a Star Battle puzzle by ensuring it follows standard rules. This includes limiting the number of stars per row, column, and region, verifying that the number of regions matches the grid size, and ensuring stars do not touch orthogonally or diagonally. We can then use this function to validate any puzzles we create.

The initial generation approach involved generating a random placement of stars for a given grid size, then building the regions around the given stars. We created two helper functions to check the star positions were not incorrect and the adjacency rules were not broken. The solution was then passed to a generate regions function which initialised a grid, assigned each star a region, added the regions to an expansion queue, then expanded each region until all cells had a region.

This method worked well for the grid size 5x5, although several challenges presented themselves as the grid size increased. On grids sized 10x10, the number of stars per column, row and region increases to 2, and on grids sized 14x14

it increases to 3. We changed our approach to first group stars together based on their Manhattan distance (Chiu et al., 2016). We created a function to enable this, queuing the stars and grouping them, ensuring all stars have a group, before generating the regions around the grouped stars. It worked in a similar fashion to the initial region generation but did incur some problems. Grouped stars were usually placed in the same region, but the regions would often be incontiguous, disconnected across the grid. Presumably issues with the distance calculation, or the region expansion. Many iterations of the functions were tested, using BFS (Apostolico and Drovandi, 2009) as an alternative star grouping mechanic, and flood fill algorithms, attempting to ensure the regions were contiguous. The final iteration resulted in seemingly valid looking puzzles.

Upon testing the puzzles with the validator function, and attempts at solving by hand, it was revealed that the puzzles generated were invalid. This was due to puzzles having more than one solution, for example, picture in Figure 5. Arguably, this was due to the programmatic approach to puzzle generation. The initial algorithms lacked the randomness that made the puzzles unique. This was becoming difficult to fix based on the original approach, so the research was pivoted in another direction.



(a) An invalid 14x14 puzzle with star placements

(b) The same puzzle with a different valid solution

Figure 5: Another solution for the same puzzle

7.3 Final Generation Method

While researching, we came across the publisher Jim Bumgardner of Krazydad (Bumgardner, 2005). His comments in a maths stack exchange post stimulated the method we use in our final generation algorithm. This looked at generating the random regions first, then solving the puzzle using a solver function, if a second solution is found the puzzle is discarded. The randomness of the method does mean the majority of the puzzles generated here will be invalid, but generated puzzles are guaranteed to be solvable and have only one solution.

To implement this idea, we started generating a random layout for the puzzle, by placing an initial cell and expanding the region, again using a flood fill algorithm. While the article suggested a heuristic-based solver, we decided against this to first generate an initial test set of puzzles. The solver involved using Microsoft Z3, a satisfiability modulo theory solver. The constraints are formalised, and the solver either returns a solution to the puzzle or a None, indicating either no solution was found or there were multiple solutions. A challenge with this method on a 10x10 grid was that I would mostly generate with at least one region smaller than 3 cells. This was an issue because it meant that most of the puzzles were unsolvable, regions in 10x10 puzzles need a minimum size of 3 cells to have 2 stars. We made simple changes to the layout function like this to decrease the likelihood of an invalid region layout.

We excluded 14x14 puzzles due to the significant increase in generation time and computational inefficiency caused by the larger grid size. The expanded search space and more complex constraints led to longer convergence times, making it impractical to train with these puzzles. However, optimization techniques such as more advanced constraint propagation, or more efficient state-space search could potentially make training on larger grids feasible in the future, allowing us to handle larger puzzles without sacrificing efficiency.

7.4 Generation and Storage of dataset

Finally, for the later stages of the project we need a dataset of puzzles and solutions to test on, so we generate this data set of puzzles using out working method. We start by initialising two lists labelled puzzles and solutions, to store the puzzle and solution as two np arrays. It builds the dataset by running our generation functions while the data set is less than the number of puzzles. One challenge was the long runtime required to reach the 1,000 puzzles, having issues such

crashes. To avoid big losses in the dataset, we implemented more frequent saving. The new implementation checks if there is already a file of the same name to start from any previously saved puzzles, and continues building the dataset, saving every 10 puzzles. This means in the event of a crash we can continue from the last saved hundred. The puzzles are saved in an NPZ file (Bressert, 2012) for ease of use, preserving the data structures and ensuring efficient storage.

We also implemented saving the dataset in a JSON format. In the event in future, we need a more readable format of the puzzles. It is also more widely compatible across different programming languages if we do visualise or publish the puzzles elsewhere (Zunke and D'Souza, 2014).

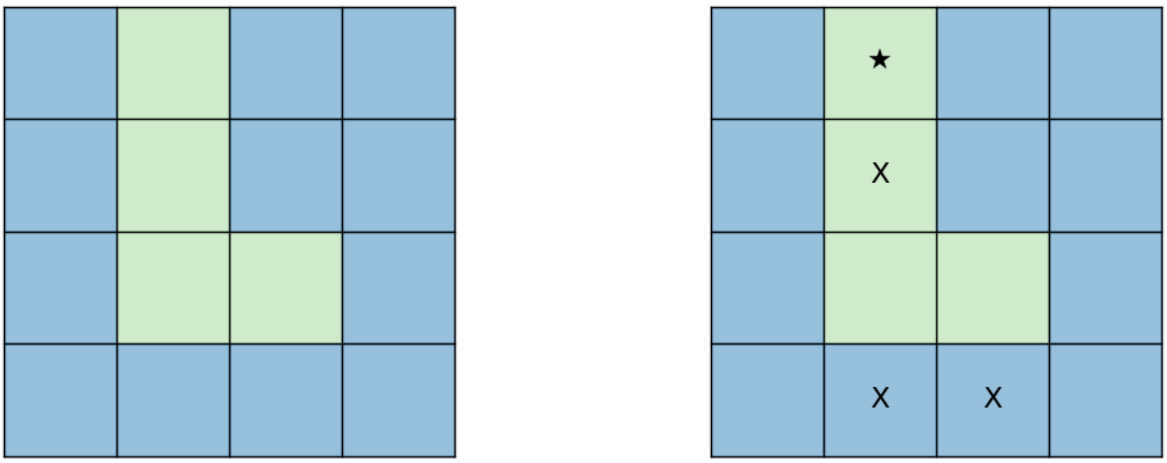
8 Difficulty Estimation

To estimate the difficulty of the puzzles, we considered human like solving techniques, implementing a production-rule solver that mimics human deduction. We manually attempted to solve several star battle puzzles, noting the heuristics that occur naturally, or with little deduction, as well as any that may require a more advanced deduction. The heuristics were split into "normal" and "hard" solving techniques. The method to determine the difficulty using the solver involves multiple stages. Initially, the function attempts to solve the puzzle using only the "normal" level heuristics, labelling the puzzle as "normal" if it succeeds. If it cannot be solved using only the normal heuristics, it moves on to work with both the "normal" and "hard" heuristics, labelling the puzzle as "hard" if it is solved. Any puzzle that cannot be solved with the heuristics we have encoded is then labelled as "extreme".

Two functions we were able to write before going into the specific techniques were based on the essential rules of Star Battle; any row, column and region must contain exactly two stars, no stars can be touching orthogonally or diagonally. One function found any stars placed in the answer and placed "0"s around the star to denote no star can be in that cell. The other function simply ensured once there was 2 stars in a row column or region, the rest of the area is then marked as "0". These are the base functions applied between iterations of our difficulty estimation function, to ensure we do not evaluate cells we can confirm do not have a star.

8.1 "Normal" Level Heuristics

We implemented 8 "normal" level heuristics. These essentially looked at recognisable shapes in the regions, revealing areas where we can immediately place stars or deduct the cells. For example, a common region shape is the "L" shaped region, this region contains 4 cells, 3 cells in a line, with an extra cell perpendicular to the line, pictured in figure 6. We can make the deductions shown based on the constraints, if a star was placed in any of the cells containing an X, the region would not be able to reach the minimum of two stars needed without breaking the star adjacency constraint. This allows human solvers to quickly recognise the pattern at any stage of the puzzle and make progress towards finding the final solution. The other normal level techniques follow a similar pattern, easily identifiable shapes, forcing star placements or deductions. We then use the base functions to ensure all possible deductions are made from the iterations of the techniques.



(a) An "L" shaped region we can apply a rule to

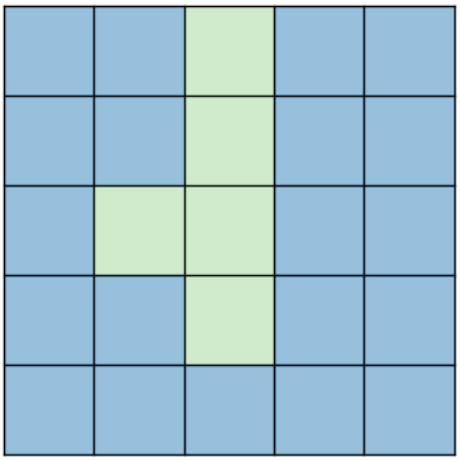
(b) The application of the "L" shape rule

Figure 6: Another solution for the same puzzle

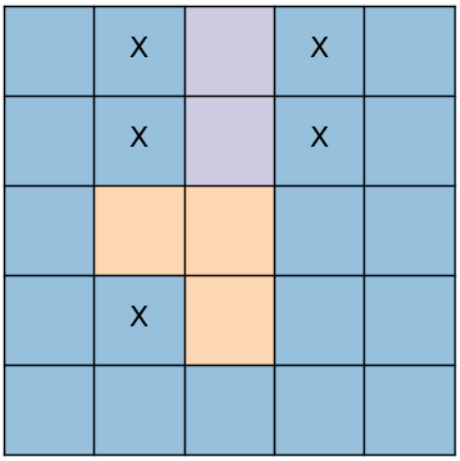
8.2 "Hard" Level Heuristics

The "hard" level heuristics were more complicated, in that they involved multiple regions or more complicated deduction chains. For example, the "Rule of Clumps", involving splitting up a region into two smaller regions that can fit in a 2x2 cell area, then applying adjacency rules to deduct surrounding cells. Combining techniques in this way might not be

natural to a new player, and so they have been regarded as more advanced techniques. As we have not hard encoded the rules for the “extreme” difficulty level, we have determined them as anything beyond the hard rules, as they are solvable by our z3 solver. This implies a human solver may require even more advanced techniques, or more simply brute force. This section of the project involved many iterations of testing due to the complicated nature and interactions of the solving algorithm.



(a) A region that can be separated into two 2x2 ”clumps”



(b) Adjacency rules can be applied to each clump

Figure 7: Another solution for the same puzzle

The difficulty assessment was then conducted on the full dataset of 2,500 puzzles, resulting in 2,497 “Extreme” puzzles, 0 “Hard” puzzles and 3 “Normal” puzzles. This implies our generation algorithm generated a much greater number of extreme levelled puzzles. Through our research and hand attempting these puzzles, the puzzles generated are greatly difficult, but are solvable as proved by our generation algorithm, and cross compared with an online solver, by Rob Owen King (King, 2019), to ensure it was not an issue with our generation algorithm.

9 Machine Learning for heuristic identification

The aim for the final section of this project is use machine learning to assess our difficulty classifications and identify any patterns or solving techniques that arise. We have chosen to use a convolutional neural network to use its strengths in pattern recognition and grid-based processing. We also will later be able to use GradCam to examine areas of the puzzle influencing the decisions of the model.

9.1 Initial CNN Model

The generated dataset is first loaded and checked to ensure its integrity is intact. Before designing any model, we first look to augment the data. We aim to artificially expand the dataset to improve its ability to generalise to new data. We start with the initial set of 1,200 unique puzzles and apply a several methods of augmentation. To begin, the dataset is flipped both horizontally and vertical, and then rotated by 90, 180 and 270 degrees. This method maintains label correctness as all puzzles and solutions were transformed identically. Each original puzzle was transformed into 6 total variations, resulting in a 6 times increase in the size of the dataset. The increase in data should allow for our models to learn more complex patterns, reduce overfitting, and improve the model’s ability to work on unseen data, ultimately leading to more accurate and robust predictions.

Next, we normalise the data to the range [0,1]. The puzzles region labels are divided by 9.0 so the values are between 0 and 1. The solutions already are arrays with only the values 0 or 1 so normalisation is not needed there. The values are normalised to ensure a consistent input scale for the CNN, improving training stability. It also prevents large values from unintentionally dominating gradients, which could lead to slower convergence or poor learning. CNNs in particular perform better with a small, consistent range of input values so normalisation was helpful to support the model’s ability to learn. We also at this stage have performed a floating-point conversion on both the puzzles and solutions to ensure compatibility with TensorFlow, preventing unintended type casting issues while training the data. The data is then split into a training and testing set, so we can train the model and then test on the unseen data. This ensures we have enough training data for the model to learn meaningful patterns, and a set of unbiased data to assess the model’s ability to generalise onto new data.

The CNN architecture is then defined. We used a sequential CNN structure optimised for grid-like data. There are first two convolutional blocks with 64 filters and 3x3 kernels in order to detect local patterns, batch normalisation layers that support stabilising the data, and MaxPooling layers to reduce the spatial dimensions. The following block has a

1x1 convolution with 128 filters to model feature combinations across channels without increasing the spatial complexity. The output is flattened for the final prediction on a dense layer with 100 sigmoid outputs, representing the 100 cells on a 10x10 grid. We tested with different kernel sizes, depth and other variables to improve the identifying of spatial patterns.

9.2 Initial Results

The initial training on 1,200 samples, augmented to train with 7,200 samples, resulted in an accuracy of 72.50%. The result demonstrated the model was able to learn meaningful patterns even from a relatively small dataset. As an initial test the result was welcome as a baseline, proving proof of concept that CNNs could work well at predicting solutions to a given set of star battle puzzles. However, the performance seemed to plateau early, indicating the model may have extracted all the learnable features.

9.3 Results with increased dataset

To further experiment with the model, we generated a further 1,300 puzzle and solution samples, bringing our dataset to a total of 2,500 unique puzzles. Augmenting this larger dataset brought the total samples to 15,000. Running a duplicated model with the larger set of data resulted in an accuracy of 73.77%, a 1.27% increase in accuracy. This is only a marginal improvement. The results here indicate that increasing the size of the training dataset might not be enough to significantly improve the accuracy of the model. There could be several reasons the model saw only slight improvement. Despite the promising results of our initial CNN model, it may be capacity limited, resulting in the underfitting of the prediction. Our current CNN model may be struggling to capture the long-range dependencies of the puzzle as it considers more localised patterns in the puzzle samples. Given these challenges, it became clear that a more sophisticated approach was needed.

One promising architecture that addresses these issues is U-Net, which is commonly used in image segmentation tasks. U-Net has been proven effective in capturing grained spatial relationships and learning contextual information, which are crucial for tasks that require detailed pattern recognition and the understanding of long-range dependencies. Motivated by its success in various segmentation and pixel-level classification tasks, we decided to explore U-Net for our puzzle-solving model. The architecture's encoder-decoder structure with skip connections enables it to learn both local and global features more effectively, making it a suitable candidate for improving puzzle classification (Ronneberger et al., 2015).

9.4 U-Net Architecture Approach

To try a new approach, we adapted U-Net Architecture to fit our dataset. The U-Net architecture is most used in image segmentation tasks due to its ability to efficiently capture spatial hierarchies and local features in images. U-net is designed specifically for segmentation tasks, which can be applied to Star Battle if we assume each cell to be a "pixel" that either contains a star or does not. U-Net can segment the grid into regions or cells, which makes it a reasonable approach for exploration. The architecture's encoder-decoder structure allows U-Net to capture low- and high-level features, potentially capturing a crucial constraint of Star Battle, the positioning of stars relative to neighbouring cells. U-Net can model spatial relationships, hopefully improving the accuracy of predictions.

The architecture is designed as follows. An input layer defines the shape of the input image, in this case a 10x10 grid with a single channel to represent a greyscale image. The following convolutional block encodes the puzzle, extracting features from the input image and reducing its spatial dimensions while increasing the number of feature channels, essentially understanding the content of the grid. The following block is a decoder aiming to reconstruct the image back to its original size, refining the features determined in the encoding phase. It progressively reconstructs the grid, providing a prediction matching the original dimensions as the final output. The final convolution uses a sigmoid activation function. While developing the model, an issue we encountered involved mismatched shapes when concatenating features from the encoder and decoder. This occurred due to the stride in convolutional layers not resulting in the exact shape required, leading to testing with Upsampling and Zero-padding. Ultimately, due to time constraints, the model's complexity was reduced, limiting the number of layers and feature maps used in both the encoder and decoder. By simplifying the model, the training time decreased, and more complex shape mismatches that may occur when working with deeper architectures were avoided.

9.4.1 Hyperparameter Tuning

To make up for a simplified model, we implemented manual hyperparameter tuning. First the data was split into training, testing and validation data. Introducing the validation set prevents overfitting by removing the testing data from the process. It provides a method of model evaluation that is unbiased.

The hyperparameters explored were the learning rate, number of filters, batch size, epochs, the dropout rate, and the choice of optimiser. By adjusting the following values, we aim to optimise the validation accuracy of our model while minimising the overall loss. This is means our model can make the best possible predictions (Iqbal et al., 2022).

- The learning rate determines the step size at each iteration while moving towards a minimum of the loss function. A smaller learning rate may result in more precise convergence, while a larger learning rate may lead to faster convergence but may overshoot the minimised loss.

- The number of filters in the convolutional layers effects how well the specific features in the input image are detected. More filters allows for a model to capture more complex features but increases computational cost and risk of overfitting.
- The batch size defines the number of training samples used in one forward and backward pass of the model. Smaller batch sizes improve a models generalisation while larger batch sizes speed up training at a cost to generalisation.
- Epochs are the number of time a model iterates over an entire training dataset. A larger number of epochs risks overfitting to the data, while too small a number of epochs can result in underfitting.
- Dropout is a regularisation technique where a couple of neurons are randomly turned off during training in an attempt to prevent overfitting. Higher dropout rates may prevent overfitting but could also slow learning if too many neurons are ignored.
- The optimisation algorithm used updates the model's weights during training. The optimisers evaluated were the Adam optimiser and SGD. Adam is generally more efficient as it adapts the learning rate per parameter, while SGD updates weights based on the gradient of the loss function.

9.4.2 Hyperparameter Tuning Results

We tuned the initial model's hyperparameters using a combination of grid search and manual adjustment. The key parameters were optimized based on validation accuracy and constraint satisfaction. For the final model, we selected a shallow U-net architecture with the tuned hyperparameters. The results of the hyperparameter tuning are listed in the figure 8.

The initial model had a high recall for the class "0", indicating the model worked well in identifying empty cells. The struggle for the model was in making predictions for the "1" class of star placements. The model results for class 1 show a low recall score of 18%, and a precision score of 61%. These results indicate the model struggled more when detecting cells containing stars, when predicting stars, it missed many. The weighted f1 score of 77% implies the model was more biased towards predicting the empty cells, likely due to the class imbalance of empty cells to starred cells.

The final model achieved similar results. With an accuracy of 81% we can infer that the performance was fairly stable. There was a slight improvement in the precision of the "1" class, suggesting improved predictions of star placement overall. The lower dropout rate and number of filters may have reduced the over-regularisation, supporting the classes in maintaining useful patterns.

Both models performed better than our previous CNN models in accuracy going from 77% to 81%, although accuracy may not be the best metric due to the class imbalance. The final model was more efficient with a training time of 3 minutes, compared to the initial U-net model, training in 9 minutes. The reduction in training time did not incur any major performance loss, which would be helpful if we were using the same model for a larger dataset.

Initial Model

	Precision	Recall	F1-score	Support
0.0	0.83	0.97	0.89	120000
1.0	0.61	0.18	0.27	30000
Accuracy		0.81		150000
Macro avg	0.72	0.57	0.58	150000
Weighted avg	0.78	0.81	0.77	150000

filters: 64, learning rate: 0.001, dropout rate: 0.5, batch size: 32, epochs: 50

Final Model

	Precision	Recall	F1-score	Support
0.0	0.82	0.98	0.89	120000
1.0	0.63	0.16	0.25	30000
Accuracy		0.81		150000
Macro avg	0.72	0.57	0.57	150000
Weighted avg	0.78	0.81	0.76	150000

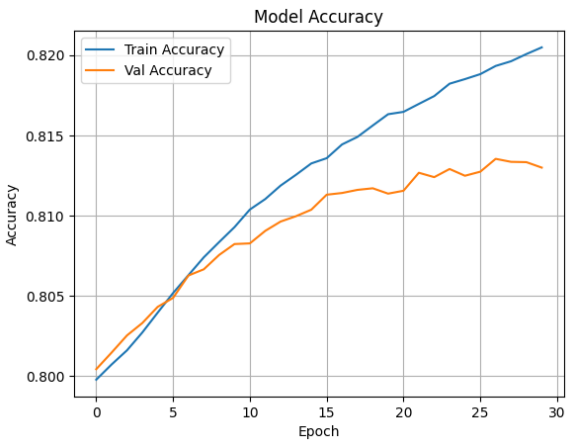
filters: 32, learning rate: 0.001, dropout rate: 0.3, batch size: 16, epochs: 30

Figure 8: Comparison of model performance metrics for initial and final CNN architectures

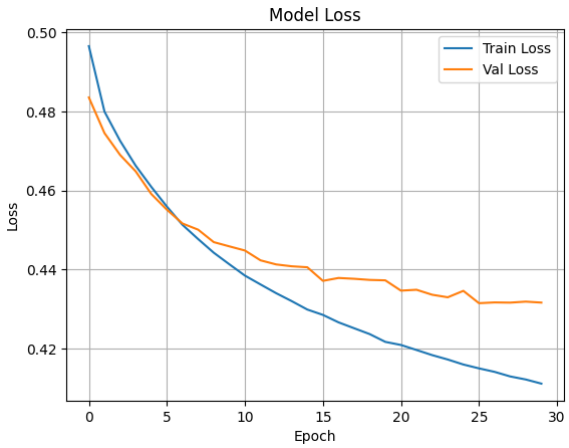
To further analyse our final model, we observe the training curve, highlighting the accuracy and loss of both the training dataset and the validation dataset (as shown in Figure 9). The training curve for our final model shows that the validation accuracy and loss began to plateau earlier than that of the training data. This implies the model may have been able to learn most of the patterns in earlier epochs of training. The graphs show a bit of a gap emerging between the training and validation results. By the end of the training, the training data accuracy exceeded validation accuracy by roughly 0.016, and validation loss was around 0.02 higher than training loss. This slight divergence suggests the beginning of overfitting in the later stages of training, affirming the decision to minimise the epochs to 30 for more efficient training.

9.5 Visualisation using Grad-CAM

We finally made predictions on the training data and were able to gain insights on the model's learning by observing the results in a Grad-CAM heatmap format. We generated several heatmaps, including a heatmap of the prediction with the highest confidence. In this visual the resulting heatmap revealed some areas of high activation. In figure 10, we can see a 3-cell region in the top left, the corresponding cells in the heat map are highly activated, indicating it has been able to identify this pattern with great confidence. Another region we can identify is the 6-cell region in the bottom right of the sample. The bottom right cell of this region is brightest in the heatmap, indicating the highest activation on this sample. This further supports the idea of pattern-based decision making, or potentially cell level reasoning. These two shapes are both commonly used in early human solving through Normal heuristics. The model's focused attention on these regions suggests it may be learning similar deductive steps, supporting the link between learned spatial reasoning and human-like strategies.



(a) Training curve of the model accuracy on training and validation datasets



(b) Training curve of the model loss on training and validation datasets

Figure 9: Graph of training curves from the final model training

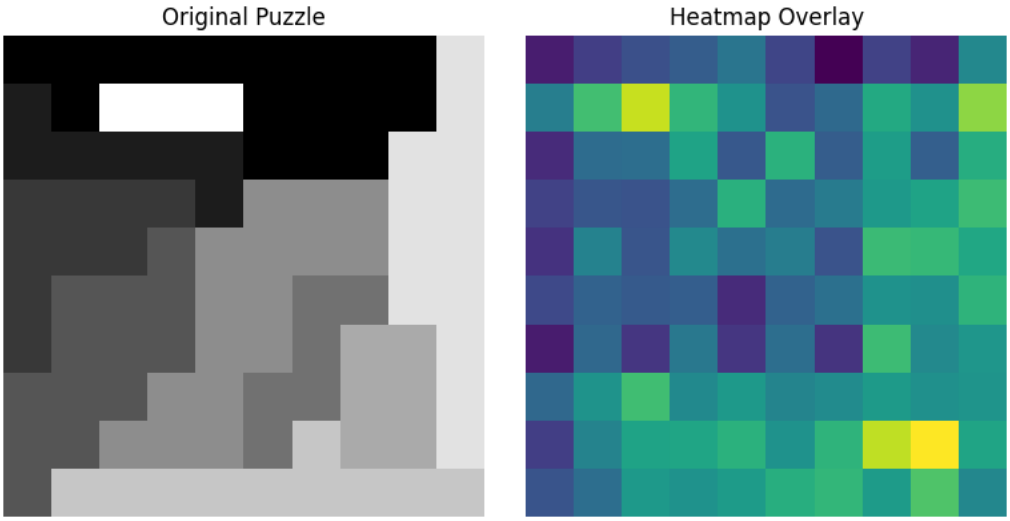


Figure 10: Grad-CAM visualisation of most confident prediction on the test dataset

10 Results and Discussion

10.1 Puzzle Generation

Over the course of the project we were able to successfully generate 2,500 unique Star Battle puzzles. The randomness of the generation technique along with the base conditions met for region sizes meant that there is a great variety of puzzles. All with the fundamental shapes that allow user deductions to make progress on the finding the solution. The region generation algorithm works well providing a diverse set of playable puzzles, with little bias towards shape due to the irregularity of randomised regions.

There were some limitations to our final limitation algorithm however as when tested on grid sizes above 14, the runtime grew exponentially, making it unsuitable for exploration in this project.

After assessing the difficulty of the puzzles, it was found that a majority of the puzzles fell into an “Extreme” level category. By hand-solving several of the puzzles, the difficulty is justified by puzzles explicitly requiring certain intuitions beyond a beginner or intermediate level. This implies there may be improvements we can make to our layout algorithm. Potentially a combination of randomised regions and modifications to manually create some easier puzzles.

10.2 Difficulty Assessment

We have implemented a production-rule based solver to assess and label the difficulty of the puzzles. This has given 3 normal level puzzles, 0 hard level puzzles, and 2497 extreme level puzzles. Of the few normal level puzzles, we were able to manually solve them using mostly beginner and techniques, consolidating the assessment of these puzzles. The dataset having no puzzles at a hard level may be partly due to issues with the difficulty classification algorithm, such as improper implementation of the Hard-level heuristics. However, it is likely that the nature of the random generation

process was producing puzzles with structurally complex regions, naturally leading to more "Extreme" level difficulty.

While the implementation was challenging, we can say the algorithm overall was effective at determining the difficulty of the puzzles. Encoding naturally occurring intuition in humans was a more difficulty task than expected, leading to inefficient implementation in certain sections, that could be improved in future works.

10.3 CNN Model and Evaluation

Our final CNN model was trained with a learning rate of 0.01, 32 filters, a dropout rate of 0.3, a batch size of 16 and 30 epochs. It achieved an overall accuracy of 81% on the test data. While it made predictions with high precision and recall for non-star cells (class 0), its performance on star cells (class 1) was weaker, with an F1-score of just 0.25. This reflects the class imbalance, suggesting the model is unconfident in predicting stars, trying to minimise false positives. Training curves also revealed that validation accuracy and loss plateaued early, and by the end of training, a small but noticeable generalisation gap appeared, highlighting potential underfitting.

To interpret what the model had learned, Grad-CAM heatmaps were generated for several predictions, including the prediction with the highest confidence. Notably, the most activated regions in the heatmap aligned with known human deduction strategies, such as identifying 3-cell linear regions and 6-cell rectangular shapes, associated with typical beginner solving heuristics. Furthermore, some heatmaps showed focused attention on specific cells within a region, suggesting that the model may be learning patterns or deductions at a cell-level. These results provide early evidence that the model has begun to capture human-like visual heuristics in its decision-making, despite its overall limitations.

10.4 Conclusions

This project explored the computational and machine learning approaches to understanding and solving the logic-based puzzle Star Battle. Through the implementation of puzzle generation, a rule-based difficulty classifier, and a machine learning solver, we investigated the potential for algorithmic and machine learning methods to assist in logic tasks usually requiring human intuition.

We successfully developed a generation algorithm capable of producing valid Star Battle puzzles. Alongside this, a set of handcrafted heuristics were designed to assess puzzle difficulty, classifying of puzzles into difficulty levels based on rule complexity and solving stages. This approach provided insight into how constraint-based logic could be used not just to solve puzzles, but also to evaluate their challenge.

To complement this, we trained convolutional neural networks (CNNs), and explored with a simplified U-Net architecture, to learn from puzzle solutions directly. Despite limited time and resource constraints, the machine learning models demonstrated the ability to recognise potential patterns and cell level intuitions within puzzles. Grad-CAM visualizations gave further insight into the model's areas of interest, suggesting the beginnings of learned heuristics similar to that of human solving methods.

Overall, this project has shown that reasoning systems hold significant promise in the domain of combinatorial puzzles. The integration of interpretability tools like Grad-CAM heatmaps allowed for analysis of both model decisions and their alignment with known logical strategies.

10.5 Future Work

This project opens up several interesting directions for further development, both in improving technical capability of our algorithms and improving insight into Star Battle and related constraint satisfaction problems.

Our puzzle Generation can be enhanced beyond simple valid puzzle creation. By incorporating new features, looking into region irregularity, complexity, or symmetry, we can aim for controlled generation of puzzles at target difficulty levels. Additionally, generative models such as autoencoders could be trained to produce Star Battle layouts. This could significantly increase dataset diversity and allow for more controlled puzzle production.

Difficulty classification could benefit from integration with machine learning. Instead of relying purely on human-like solving heuristics, models could be trained to classify puzzles based on region layout features or learn difficulty representations directly. Simulation-based approaches, such as tracking solver runtime, depth, or branching complexity, could help create more standardised level definitions.

As for our CNN models, these can be greatly improved. CNNs may be extended with deeper architectures and connections to improve feature learning. A new attempt of the U-Net architecture with more depth could help capture global puzzle structures. Furthermore, reinforcement learning could be explored to train solvers by its techniques of positive or negative feedback, mirroring human trial-and-error strategies.

Scaling the dataset is another practical goal. Increasing the dataset and obtaining difficulty votes from advance puzzlers could provide a larger, more accurately labelled dataset. Additionally, frameworks could be developed to further evaluate solver performance under varying constraints, contributing back to the broader CSP and puzzle AI communities.

At present, all widely recognised solving strategies for Star Battle puzzles are hand-crafted, based on human intuition and logical reasoning. There may be effective strategies that are not yet formalised into widely accepted puzzle techniques. There is opportunity through further work to interpret deep learning models using tools like heatmaps or saliency maps to provide further insights into solving patterns learning by a given model. We might look to puzzles predicted with high confidence score, or specifically training on only difficult puzzles to look for areas of high activation. This can help to improve widespread understanding of the puzzle space, helping to understand the "black box" learning completed by deeper learning models.

11 Bibliography

Alpaydin, E. (2020). *Introduction to machine learning*. MIT press.

Apostolico, A., & Drovandi, G. (2009). Graph compression by bfs. *Algorithms*, *2*(3), 1031–1044.

Barrett, C., & Tinelli, C. (2018). Satisfiability modulo theories. *Handbook of model checking*, 305–343.

Brailsford, S. C., Potts, C. N., & Smith, B. M. (1999). Constraint satisfaction problems: Algorithms and applications. *European journal of operational research*, *119*(3), 557–581.

Bressert, E. (2012). Scipy and numpy: An overview for developers.

Bumgardner, J. ". (2005, January). Krazydad free puzzles. https://krazydad.com/twonottouch/

Chiu, W.-Y., Yen, G. G., & Juan, T.-K. (2016). Minimum manhattan distance approach to multiple criteria decision making in multiobjective optimization problems. *IEEE Transactions on Evolutionary Computation*, *20*(6), 972–985.

Estermann, B., Lanzendörfer, L. A., Niedermayr, Y., & Wattenhofer, R. (2024). Puzzles: A benchmark for neural algorithmic reasoning. *arXiv preprint arXiv:2407.00401*.

Iqbal, S., Qureshi, A. N., Ullah, A., Li, J., & Mahmood, T. (2022). Improving the robustness and quality of biomedical cnn models through adaptive hyperparameter tuning. *Applied Sciences*, *12*(22), 11870.

King, R. O. (2019, January). Star battle solver. https://www.robowenking.com/star_battle?class=card-project

Laszlo, M., & Mukherjee, S. (2023). Counting star-battle configurations. *Mathematics in Computer Science*, *17*(2), 8.

Mirshekarian, S., & Sormaz, D. (2018). Machine learning approaches to learning heuristics for combinatorial optimization problems. *Procedia Manufacturing*, *17*, 102–109.

Pelánek, R. (2014). Difficulty rating of sudoku puzzles: An overview and evaluation. *arXiv preprint arXiv:1403.7373*.

Ronneberger, O., Fischer, P., & Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*, 234–241.

Tosi, S. (2009). *Matplotlib for python developers*. Packt Publishing Ltd.

Wallis, W. D., & George, J. C. (2016). *Introduction to combinatorics*. Chapman; Hall/CRC.

Zunke, S., & D'Souza, V. (2014). Json vs xml: A comparative performance analysis of data exchange formats. *IJCSN International Journal of Computer Science and Network*, *3*(4), 257–261.

12 Appendix

12.1 Progress Log

- Week 1: Researching on topics to chose from
- Week 2: Researching organising 3 topics I had interest in.
- Week 3: Reached out to prospective supervisor to discuss ideas for project
- Week 4: Begin project proposal and continue to research around the project topic
- Week 5: Understanding SAT and SMT Solvers
- Week 6: Reading on combinatorial problems and initiating interim report.
- Week 7: Finalise and submit interim report.

12.2 Project Proposal

Sophia Williams 246549

Dr Vincent Van Oostrom

Star Battle: Generation, Complexity Analysis, and Heuristic learning

Star Battle is a logic-based puzzle game played on a grid, where the objective is to place a set number of stars in each row, column, and designated region. The challenge lies in ensuring that no two stars touch, even diagonally. This constraint makes Star Battle an intriguing problem in the field of constraint satisfaction and combinatorial optimization. Originally designed as a recreational puzzle, it has gained popularity due to its simple rules yet complex solution strategies. The puzzle's structure makes it a suitable candidate for computational analysis, allowing for exploration of puzzle generation, solving techniques, and difficulty scaling. This project leverages these characteristics to study the complexities of the puzzle and explore potential for algorithmic and human-solving heuristics.

Aims:

The aim of this project is to explore puzzle generation, complexity analysis and solving strategies in the puzzle Star Battle. By focusing on these aspects, the project aims to provide insights into both the theoretical and practical challenges of puzzle creation and solving, contributing to a broader understanding of constraint satisfaction problems.

My motivation for selecting this topic stems from a deep interest in puzzles as a form of exploration.

Primary Objectives:

- Generate puzzles in a complete process
- Examine difficult of solving a given puzzle
- Assess whether heuristics for solving can be learned automatically

Extension:

- Develop an interface for puzzling
- Automated hinting system

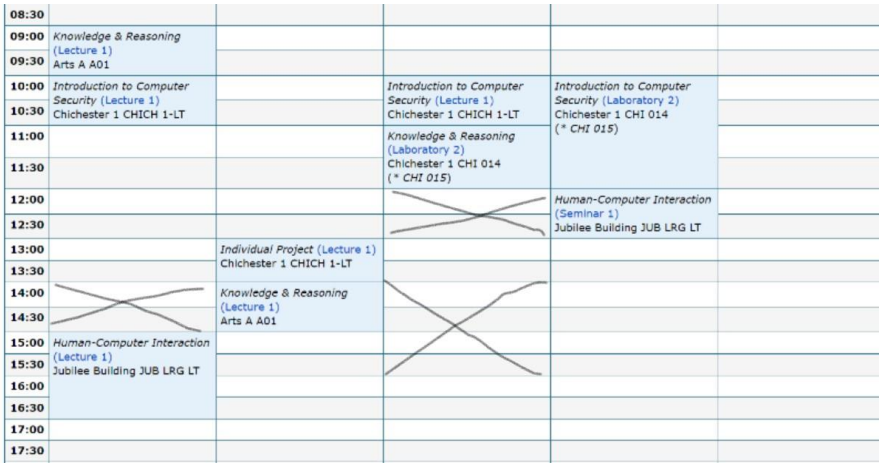
Relevance:

The project is relevant to my degree in Computer Science and Artificial Intelligence, as it draws on key concepts. The generation and solving of puzzles require a deep understanding of algorithms, constraint satisfaction problems and computational complexity, fundamental topics in computer science. Additionally, the exploration of heuristics and potential to automate solving strategies closely aligns with my interest in machine learning. This aspect offers an opportunity to apply my learnt knowledge in modules like fundamentals of learning and acquired intelligence and adaptive behaviour.

Resources:

- Python programming language
- JavaScript if I extend to interface development
- VS Code for an IDE
- NumPy and SciPy for matrix-based puzzle representation
- TensorFlow/Scikit-learn for ml heuristic strategies
- GitHub for version control and backup

Timetable:



I intend to dedicate 1 hour each weekday to my dissertation.

Bibliography:

Barrett, C., & Tinelli, C. (2018). Satisfiability modulo theories. Handbook of model checking, 305-343.

UEHARA, R. (2023). Computational complexity of puzzles and related topics. Interdisciplinary Information Sciences, 29(2), 119-140.

Felgenhauer, B., & Jarvis, F. (2006). Mathematics of sudoku I. *Mathematical Spectrum*, *39*(1), 15-22.

Russell, E., & Jarvis, F. (2006). Mathematics of sudoku II. Mathematical Spectrum, 39(2), 54-58.

Wallis, W. D., & George, J. C. (2016). Introduction to combinatorics. Chapman and Hall/CRC.

Laszlo, M., & Mukherjee, S. (2023). Counting Star-Battle Configurations. *Mathematics in Computer Science*, *17*(2), 8.

12.3 Project Timeline Gantt

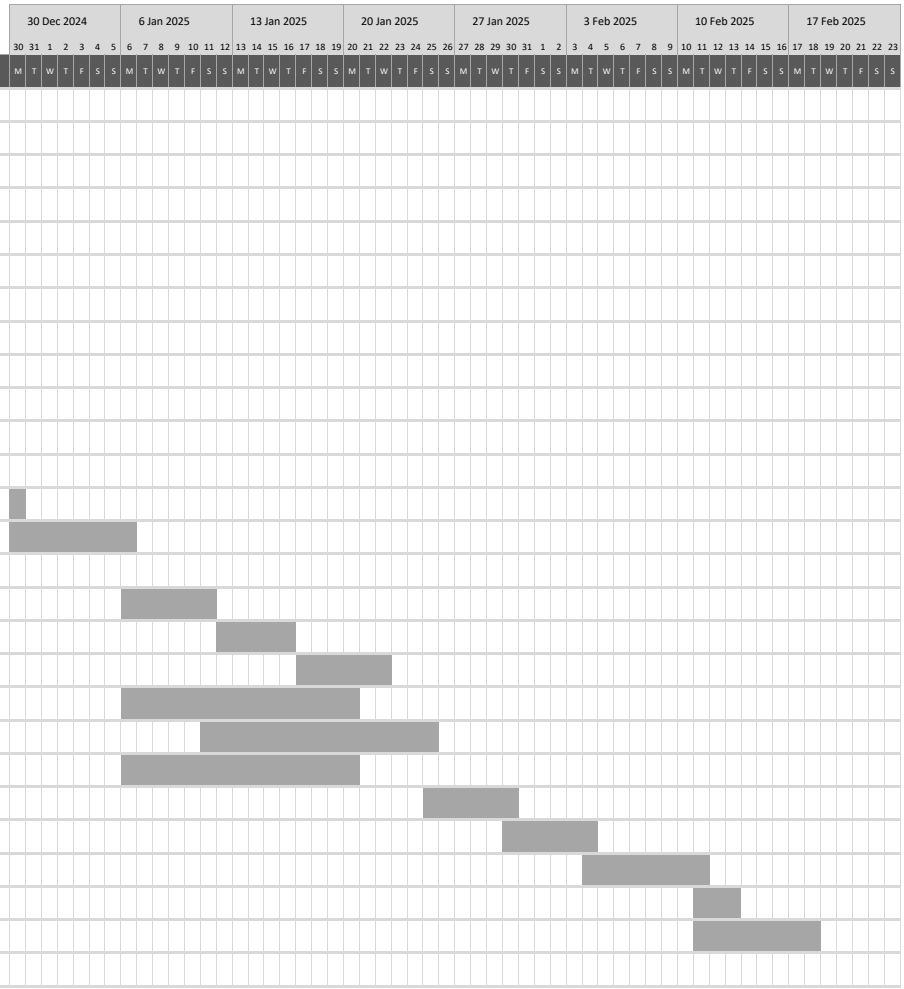
Star Battle: Generation, Analyis and Heuristics

University of Sussex
Sophia Williams

Project Start: Tue, 01/10/2024

14

TASK	PROGRESS	START	END
Phase 1 Research and Interim Report			
Research to confirm project	100%	30/9/24	1/10/24
Background research project	100%	1/10/24	23/10/24
Draft Interim Report	100%	23/10/24	5/11/24
Edit Interim Report	100%	5/11/24	6/11/24
Submit Interim Report	100%	6/11/24	6/11/24
Practice new libraries	100%	7/11/24	12/11/24
Phase 2 Generate Puzzles			
Define puzzle representation	100%	12/11/24	16/11/24
Implement basic generation algorithm	100%	16/11/24	23/11/24
Define constraints for solutions	100%	23/11/24	30/11/24
SAT solve to validate uniqueness	100%	30/11/24	7/12/24
Define puzzle collection format	100%	23/11/24	30/12/24
Generate initial set of test puzzles	100%	30/11/24	6/1/25
Phase 3 Determine puzzle difficulty			
Method discussion	100%	6/1/25	11/1/25
Set sat solver constraints	100%	12/1/25	16/1/25
Measure solution complexity	100%	17/1/25	22/1/25
Human puzzle solving and rating	100%	6/1/25	20/1/25
Apply results to difficulty decision	100%	11/1/25	25/1/25
Identify solving heuristics	100%	6/1/25	20/1/25
Simulate solving steps programmatically	100%	25/1/25	30/1/25
Establish criteria for difficulty levels	100%	30/1/25	4/2/25
Evaluate and adjust	100%	4/2/25	11/2/25
Generate and label test set	100%	11/2/25	13/2/25
Testing to validate difficulty measures	100%	11/2/25	18/2/25
Phase 4 Heuristic s			



14

TASK	PROGRESS	START	END
Define problem space	100%	19/2/25	22/2/25
Generate and preprocess training data	100%	22/2/25	24/2/25
Train initial heuristic model	100%	24/2/25	1/3/25
Evaluate initial model	100%	1/3/25	8/3/25
Experiment with alternative models	100%	8/3/25	15/3/25
Evaluate further and optimise	100%	15/3/25	29/3/25
Insert new rows ABOVE this one