EXPRESSING HUMAN-ORIENTED STRATEGIES VIA PLANIMATION

RESEARCH THESIS (COMP90055)

STUDENT NAME:	ADITI BASU
STUDENT ID:	1178282
ADVISORY PANEL:	Prof Leon Sterling, Dr Nir Lipovetzky
DATE OF SUBMISSION:	18/11/2022
ORGANISATION:	UNIVERSITY OF MELBOURNE

1 Acknowledgements

I would like to sincerely acknowledge my supervisors, Professor Leon Sterling and Dr Nir Lipovetzky, for the engagement, enthusiasm and support they have offered me and this research project. They have truly made my first research experience very positive and worthwhile, by enabling me to execute and deliver this research.

Having had numerous insightful discussions about human strategies in FlowFree puzzles with Leon and been directed towards useful and beneficial technical resources by Nir, I felt strongly supported by the combined skills and knowledge they had to offer. Above all, this has been a highly enriching learning experience for me.

Special thanks to Planimation developer, Yi Ding, for his assistance in navigating errors in Planimation during the development of the animation profile.

2 Abstract

AI systems solve puzzles quite differently to human players and their solutions may not be entirely interpretable or intuitive to humans. This can promote mistrust towards AI systems, as the reasoning behind certain moves is unclear. The area of Explainable AI (XAI) aims to reduce mistrust by increasing the interpretability of AI systems. Planimation, an open-source framework, contributes to XAI by visualising planning problems and their sequential solutions. Another way to increase interpretability of AI systems is to shape their solutions in a way that resembles those that humans might produce. This research focuses on FlowFree, a prime example of an AI planning problem, by encoding one and visualising two human strategies on Planimation. The solutions produced by these strategies are compared with one another, and with that of a regular AI solver, through the visuals generated by Planimation.

3 Introduction

Understandably, humans tend to solve puzzles quite differently to Artificial Intelligence (AI) systems, considering they do not bear the computational power and memory that AI systems can have. Instead, humans use strategies, a problem-solving technique which involves either deconstructing the puzzle into smaller, human-solvable sub-problems or modelling the puzzle in a different light. The area of Explainable AI (XAI) aims to move away from 'black-box' algorithms by coupling the computational and memory capacities of traditional AI systems and the strategies used by humans to achieve AI systems that are more trustworthy and amenable to humans.

Planimation (Chen et al., 2020) is an open-source framework that visualises sequential solutions for planning problems specified in the Planning Domain Definition Language (PDDL). It has been an essential step towards attaining XAI by making AI-generated solutions more interpretable to humans. Planimation has successfully solved and visualised problems in an array of domains (e.g., Towers of Hanoi, Blocks, etc.) to date. As a result of this research, it can now also execute and visualise human strategies.

To narrow its area of focus, this research aims to incorporate human strategies used in FlowFree puzzles within Planimation, so it can be used to effectively communicate strategies between humans and assist in identifying drawbacks of strategies, if any. This is an essential step towards achieving XAI and integrating human strategies into planning solutions generated by AIs.

FlowFree is a prime example of an AI planning problem where, given an initial state, the task is to find a sequence of actions that will achieve the goal state. In FlowFree, the initial state is the configuration of the coloured dots on the square grid with no pipe connections. Actions include extending a pipe from any of the coloured dots or incomplete pipe connections. The goal is to achieve a state where all matching dots are connected and the pipes fill up the entire square grid without intersecting each

other, all with the minimum number of moves possible. Every switch to a different colour is considered a move.

4 Related Works

Increasing interpretability and trust in AI systems is a principle aim of AI research. One way AI systems can win the trust of humans is by providing human-understandable explanations for achieving a collaborative task [Akula et al.]. Such explanations can be achieved through visualisation, which can communicate strategies and plans both effectively and succinctly. Effective visualisation of human strategies is therefore a focus of this research.

Humans can describe strategies they employ in problem-solving scenarios, which is generally more effective than simply enumerating an algorithm or a plan [Kim et al. 2017], as learners can then generalise these strategies to other domains. This research successfully provides visual explanations of strategies and enables humans to compare and identify drawbacks of their strategies via these visualisations.

Bian [2021] specifies and implements a PDDL domain in her thesis to solve variations of FlowFree puzzles. Unfortunately, her FlowFree PDDL domain specification resulted in the PDDL solver running out of stack memory for certain 7x7 Flow Free puzzles. Furthermore, Bian's domain does not contain any human strategy encodings; it is a pre-strategy domain that an AI solver can use to solve FlowFree puzzles in a fashion that differs significantly from how human players would solve them. This research project extends Bian's research by producing a new FlowFree PDDL domain and encoding a human strategy onto it.

5 Research Design & Methodology

The first step in the research was to examine Bian's FlowFree PDDL domain, looking for unnecessary complications and odd moves. This analysis informed my construction of the pre-strategy domain described in Section 5.1.

Concurrently, I identified human strategies used in FlowFree by solving puzzles regularly and discussing approaches for solving FlowFree puzzles with Professor Leon Sterling. We each identified a strategy; I called mine the Edgy strategy and Professor Sterling called his the Outside-in strategy. These strategies are described in Section 5.3.

Once the pre-strategy domain was established and conformed to a reasonable standard, it was extended to incorporate the Edgy strategy. The Edgy strategy domain is described in Section 5.3.1. The Outside-in strategy has not been encoded onto the pre-strategy domain as it has not yet been sufficiently formalised. However, the two strategies can be partially compared, as discussed in Section 7.2.

Finally, the animation profile was developed to allow Planimation to visualise FlowFree problems and their solutions. The animation profile can be shared across the pre-strategy and Edgy strategy domains. It can also be used to visualise Outside-in strategy-based solutions using predefined plans (as opposed to using a solver).

5.1 Pre-Strategy Domain

The following definitions will be used throughout this report.

- Node: the building block of a FlowFree puzzle grid. Each node can either be empty or contain a pipe or a dot.
- dot: the coloured circles that are initially placed in a FlowFree puzzle grid
- pipe connection: the pipe connecting two dots of a certain colour. Each pipe may consist of two or more nodes.

Predicates and actions in the pre-strategy domain are described in the next two sections.

5.1.1 Predicates

- `(head ?n ?d)`: holds true if node ?n is the head of a pipe connection of a certain colour ?d (i.e. the furthest node ?n a pipe connection has reached from one dot ?d to the other ?d-coloured dot).

- `(empty ?n)`: holds true if node ?n does not contain a dot or a pipe.
- `(selected ?d)`: holds true if ?d-coloured dots are the currently selected dots that are being connected.
- `(finished ?d)`: holds true if two ?d-coloured dots have been connected.
- `(adjacent ?n1 ?n2)`: holds true if nodes ?n1 and ?n2 are next to each other.
- `(filled ?n ?d)`: holds true if node ?n is filled with a ?d-coloured dot.

5.1.2 Actions

- `(select-colour ?d1 ?d2)` has the following effects:
    - ?d2 becomes the selected colour `(selected ?d2)`
    - ?d1 is no longer the selected colour `(not (selected ?d1))`
    - ?d1-coloured dots are deemed connected to each other `(finished ?d1)`

  The above effects apply if and only if:
    - ?d1 is currently selected `(selected ?d1)`
    - ?d2 is not currently selected `(not (selected ?d2))`
    - ?d2-coloured dots have not yet been connected `(not (finished ?d2))`
- `(move ?n1 ?n2 ?d)` has the following effects:
    - ?n2 becomes the head for ?d colour `(head ?n2 ?d)`
    - ?n2 is no longer empty `(not (empty ?n2))`

  The above effects apply if and only if:
    - ?n2 is either empty `(empty ?n2)` or filled with ?d-coloured pipe `(filled ?n2 ?d)`
    - ?d is the currently selected colour `(selected ?d)`
    - ?n1 is the head for colour ?d `(head ?n1 ?d)`
    - ?n1 and ?n2 are adjacent `(adjacent ?n1 ?n2)`

A limitation of Bian's pre-strategy domain is the existence of partial connections. Figure 1 below shows an example of partial connections. While this is a supportive

strategy that human players might use in larger FlowFree puzzles, it is not as helpful in smaller puzzles as it costs extra moves to the player. Furthermore, there is no reason for AI solvers to use this method as they have sufficient computational capabilities and memory – both of which their human counterparts may lack – to avoid this unnecessary cost.

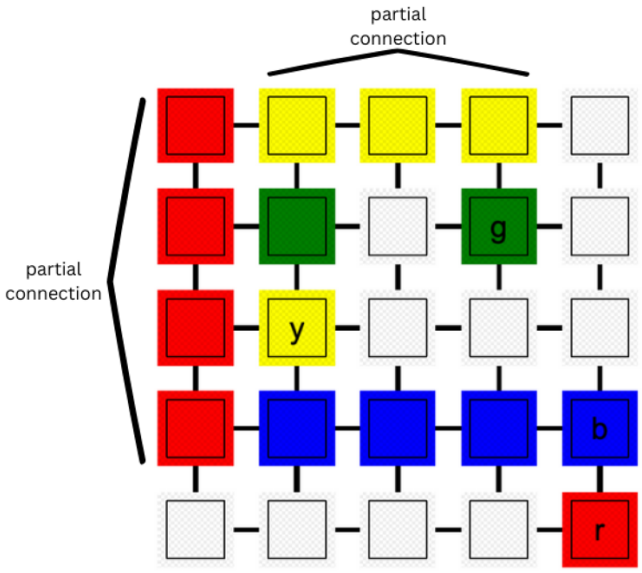

Figure 1: Partial connections in a FlowFree puzzle. The red and yellow dots are only partially connected, while the blue dots are fully connected.

For these reasons, the pre-strategy domain was designed in a way where no state with partial connections exists. This was achieved by defining the `select-colour` action such that it never selects a colour ?d that has already been selected once (i.e., `(finished ?d)` holds true). Refer to the pre-strategy domain code excerpt below.

```
(:action select-colour
        :parameters      (?d1 - dot
                          ?d2 - dot)

        :precondition    (and
                             (selected ?d1)
                             (not (selected ?d2))
                             (not (finished ?d2))
                         )

        :effect          (and
                             (selected ?d2)
```

```
                    (not (selected ?d1))
                    (finished ?d1)
                )
)
```

Figure 2: Code excerpt defining the select-colour action. The predicates highlighted in red show how this action never chooses a colour that has already been selected once, thus preventing the existence of partial connections.

5.2 Pre-Strategy Problem

The problem file definition involves specifying the objects existing in a FlowFree puzzle, the initial state, and the goal state. The puzzle shown in Figure 3 is used as a reference puzzle for this section.

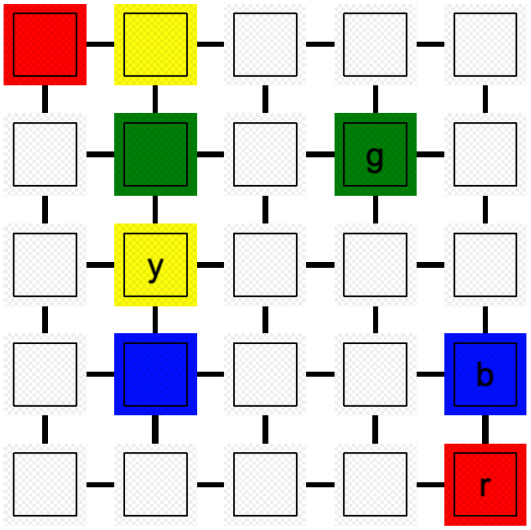

Figure 3: Initial state of sample 5x5 FlowFree puzzle

5.2.1 Objects

Objects in FlowFree puzzles constitute all nodes in the puzzle grid and the coloured dots. Nodes are numbered using the “node[x]-[y]” convention, where `node1-1` lies in the bottom-left corner of the puzzle grid, `node1-5` in the top-left corner, `node5-5` in the top-right corner, and `node5-1` in the bottom-right corner. Puzzles of other grid sizes follow a similar convention. For Planimation to create the correct visual, this convention needs to be followed strictly.

The coloured dots can be named using any convention that suits the person creating the problem file. In this case, the dots have been named by using the first letter of

their colour, which is arguably the most intuitive convention. Note that each dot object refers to the pair of like-coloured dots, and not each dot individually.

```
(:objects
    r y g b - dot
    node1-5 node2-5 node3-5 node4-5 node5-5 - node
    node1-4 node2-4 node3-4 node4-4 node5-4 - node
    node1-3 node2-3 node3-3 node4-3 node5-3 - node
    node1-2 node2-2 node3-2 node4-2 node5-2 - node
    node1-1 node2-1 node3-1 node4-1 node5-1 - node
)
```

Figure 4: Code excerpt specifying objects for the sample FlowFree problem file.

5.2.2 Static Predicates

All node adjacencies need to be specified using the `(adjacent ?n1 ?n2)` predicate as part of the initial state in the problem file. The `(position ?n)` predicate also needs to be specified for every node ?n in the puzzle. These relations remain unchanged throughout the course of the puzzle (i.e., no new relations are created using these predicates). All puzzles of the same grid size bear the same node adjacencies and position predicate specifications. For brevity, the code excerpt specifying these relations has not been included in this section – refer to the Appendix for a sample FlowFree problem file.

For every node ?n that contains a dot ?d in the initial state, the `(filled ?n ?d)` predicate needs to be specified, as shown in Figure 5 below. These predicate specifications are unique to each FlowFree puzzle. Like the adjacent and position predicates, the filled predicates remain unchanged throughout the course of the puzzle.

```
; filled relations
(filled node1-5 r) (filled node2-5 y)
(filled node2-4 g)
(filled node2-2 b)
(filled node4-4 g)
(filled node2-3 y)
(filled node5-2 b)
(filled node5-1 r)
```

Figure 5: Code excerpt showing the required `filled` relations for sample FlowFree puzzle

5.2.3 Non-Static Predicates

Since interpretability of the pre-strategy domain is not pertinent, the first colour is always selected arbitrarily and manually specified as part of the initial state by the person creating the FlowFree problem file. Refer to the code excerpt below.

```
; select first colour arbitrarily
(selected r)
```

Figure 6: Code excerpt showing the first colour selection.

The initial state also includes specification of all empty nodes, using the `(empty ?n)` predicate.

```
; empty relations - CHANGE PER PROBLEM
 (empty node3-5) (empty node4-5) (empty node5-5)
 (empty node1-4) (empty node3-4) (empty node5-4)
 (empty node1-3) (empty node3-3) (empty node4-3) (empty node5-3)
 (empty node1-2) (empty node3-2) (empty node4-2)
 (empty node1-1) (empty node2-1) (empty node3-1) (empty node4-1)
```

Figure 7: Code excerpt showing all required `empty` relations for sample FlowFree puzzle.

The pre-strategy domain allows for the user to manually specify which dot in each pair of coloured dots the pipe connection should start at. In the reference puzzle, there are two red dots – one at `node1-5`, and the other at `node5-1`. The initial state would involve specifying the `(head ?n ?d)` relation for only one of these dots – either `(head node1-5 r)` or `(head node5-1 r)`. This choice is made arbitrarily by the creator of the problem file.

```
; head relations - CHANGE PER PROBLEM
 (head node1-5 r) (head node2-5 y)
 (head node2-4 g)
 (head node2-2 b)
```

Figure 8: Code excerpt showing all required `head` relations for sample FlowFree puzzle.

5.3 Human Strategies

Two human strategies are proposed in this research – the Edgy strategy and the Outside-in strategy. The Edgy strategy has been fully formalised and defined as part of this research, while the Outside-in strategy still requires more formalisation.

Nevertheless, the rules specified in the Outside-in Strategy Specification section have been sufficient in solving several FlowFree puzzles.

5.3.1 The Edgy Strategy Specification

The Edgy strategy was inspired by the observation of pipe connections 'snaking' around each other as opposed to forming the shortest connections possible, especially in larger FlowFree grids. This 'snaking' attribute is an important attribute to realise in larger grids because there are far more empty nodes to fill compared with smaller grids. Simply forming the shortest connections possible between pairs of dots may result in all the dots being connected but not in all the nodes being filled with pipe connections.

The Edgy strategy centres around moving the head of a pipe connection along edges as much as possible. The order of selection of the pairs of dots to be connected is also dependent on how many nodes their connection can traverse that are next to an edge – the more such nodes a connection can traverse, the more its colour is prioritised.

Under this strategy, the initial state definition of any FlowFree puzzle will include the area outside the puzzle grid as being an edge, thereby rendering the outermost layer of nodes as being next to an edge. This holds in addition to the initial state definition of the pre-strategy FlowFree domain. As complete pipe connections are formed between like-coloured nodes, these pipe connections start functioning as edges for the rest of the puzzle.

While the conceptual goal state does not change from the pre-strategy FlowFree domain, this strategy tries to maximise the total number of nodes next to an edge (at the time of connection) that all pipe connections traverse.

5.3.1.1 The Edgy Strategy Domain

Under the Edgy strategy, an edge is defined as the wall around the FlowFree puzzle grid, and any node that contains a pipe that is part of a completed pipe connection between two like-coloured dots.

The Edgy strategy domain introduces two new predicates.

- `(nextToEdge ?n)`: holds true if the node ?n is next to an edge
- `(currentPath ?n)`: holds true if the node ?n is part of the current pipe connection being formed.

Due to the technical issue described in Section 6.3, the pipe predicate was added and the empty predicate removed.

- `(pipe ?n)`: holds true if node ?n contains a pipe or dot. This predicate is the exact opposite of the previously defined empty predicate (i.e., `(pipe ?n) = (not (empty ?n))`).

This domain also uses the `:action-costs` requirement to introduce a cost function that keeps track of the total cost in a FlowFree puzzle. Under this strategy, the cost increments by one every time a connection traverses a node ?n for which `(nextToEdge ?n)` does not hold true. Figure 9 shows the expression of the cost function. The total cost is modified through the actions specified in this domain.

```
; cost function
(:functions
      (total-cost) - number
)
```

Figure 9: Code excerpt showing the cost function for the Edgy strategy.

This domain modifies the `move` and `select-colour` actions specified in the pre-strategy domain and introduces two new actions.

- `(select-colour ?d1 ?d2)`: this action maintains the same definition as in the pre-strategy domain, except for the addition of a cost modification statement. This action increments the cost by one.

```
; move selection from d1 to d2 based on cost
```

```
    (:action select-colour
        ...
        :effect         ( ...

                            (increase (total-cost) 1)
                        )
    )
```

Figure 10: Code excerpt specifying `select-colour` cost modification in Edgy strategy domain.

- `(move ?n1 ?n2 ?d)`: this action has been modified from its original definition in the pre-strategy domain by only catering for moves where node ?n2 is not next to an edge and does not contain a dot ?d. ?n2 also gets added to the set of nodes that are part of the current pipe connection (i.e. `(currentPath ?n2)` holds true). This action increments the cost by one.

```
; move from n1 to n2 where n2 is NOT nextToEdge
    (:action move
        :parameters     (?n1 - node
                         ?n2 - node
                         ?d - dot)

        :precondition   (and
                            (not (pipe ?n2))
                            (not (filled ?n2 ?d))
                            (selected ?d)
                            (head ?n1 ?d)
                            (not (head ?n2 ?d))
                            (adjacent ?n1 ?n2)
                            (not (nextToEdge ?n2))
                        )

        :effect         (and
                            (head ?n2 ?d)
                            (pipe ?n2)
                            (currentPath ?n2)

                            ; increment cost as ?n2 is not
nextToEdge
                            (increase (total-cost) 1)
                        )
    )
```

Figure 11: Code excerpt specifying the modified `move` action in Edgy strategy domain.

- `(moveNextToEdge ?n1 ?n2 ?d)`: this action has been added to only cater for moves where node ?n2 is next to an edge and does not contain a dot ?d. ?n2 also gets added to the set of nodes that are part of the current pipe connection (i.e., `(currentPath ?n2)` holds true). This action does not modify the cost.

```
    ; move from n1 to n2 where n2 is nextToEdge
   (:action moveNextToEdge
       :parameters      (?n1 - node
                         ?n2 - node
                         ?d - dot)

       :precondition    (and
                            (not (pipe ?n2))
                            (not (filled ?n2 ?d))
                            (selected ?d)
                            (head ?n1 ?d)
                            (not (head ?n2 ?d))
                            (adjacent ?n1 ?n2)
                            (nextToEdge ?n2)
                        )
       :effect          (and
                            (head ?n2 ?d)
                            (pipe ?n2)
                            (currentPath ?n2)

                            ; no cost increase as ?n2 is nextToEdge
                            (increase (total-cost) 0)
                        )
   )
```

Figure 12: Code excerpt specifying `moveNextToEdge` action in Edgy strategy domain.

- `(finishPath ?n1 ?n2 ?d)`: this action has been added to cater for moves where node ?n2 contains a dot ?d. It is executed at the last step of each pipe connection, where the connection finally reaches the other like-coloured dot. All nodes adjacent to nodes that are part of the current pipe connection are now considered next to an edge. The current pipe connection is also reset such that it does not contain any nodes. This action does not modify the cost.

```
    ; finish a path by connecting to second dot
    (:action finishPath
        :parameters     (?n1 - node
                         ?n2 - node
                         ?d - dot)

        :precondition   (and
                            (filled ?n2 ?d)
                            (selected ?d)
                            (head ?n1 ?d)
                            (not (head ?n2 ?d))
                            (adjacent ?n1 ?n2)
                        )

        :effect         (and
                            (head ?n2 ?d)

                            ; convert nodes adjacent to currentPath
nodes to nextToEdge
                            (forall (?cp - node ?n - node)
                                (when
                                    (and (currentPath ?cp) (adjacent
?cp ?n))
                                    (nextToEdge ?n)
                                )
                            )

                            ; reset all currentPath predicates to
false
                            (forall (?x - node)
                                (when
                                    (currentPath ?x)
                                    (not (currentPath ?x))
                                )
                            )

                            ; no cost increase as all dots will need
to execute this action regardless of initial state
                            (increase (total-cost) 0)
                        )
    )
```

Figure 13: Code excerpt specifying `finishPath` action in Edgy strategy domain.

5.3.1.2 The Edgy Strategy Problem

Under this strategy, the problem file follows the same specification as the pre-strategy domain described in Section 3.2, with a few modifications described below.

Addition of Dummy Dot Object

A dummy dot object was added to the Edgy strategy problem file to enable intelligent selection of the first coloured dots that are going to be connected, rather than arbitrarily specifying the starting selection – refer to the code excerpt below.

```
(:init
      ...
(:objects
      r y g b dummy - dot
      ...
)
...
; select dummy dot
(selected dummy)
...)
```

Figure 14: Code excerpt showing the addition of the dummy object

Statement of Initial Cost

Following the addition of the cost function in the domain file, the initial value of the cost is specified in the problem file as 0 – refer to code excerpt below.

```
; specify initial value of functions
(= (total-cost) 0)
```

Figure 15: Code excerpt specifying the initial cost.

Statement of Optimisation Method for Cost Function

To allow for the cost function to be utilised by the problem file, its optimisation method is also specified in the problem file. In this case, the cost function needs to be minimised.

```
(:metric
     minimize (total-cost)
)
```

Figure 16: Cost minimisation in Edgy strategy domain.

Expression of Goal State

The expression of the goal state has been modified under this strategy to use the `pipe` predicate instead of the `empty` predicate, for reasons explained in Section 6.3.

```
; pipe relations
 (pipe node3-5) (pipe node4-5) (pipe node5-5)
 (pipe node1-4) (pipe node3-4) (pipe node5-4)
 (pipe node1-3) (pipe node3-3) (pipe node4-3) (pipe node5-3)
 (pipe node1-2) (pipe node3-2) (pipe node4-2)
 (pipe node1-1) (pipe node2-1) (pipe node3-1) (pipe node4-1)
```

Figure 17: Expression of goal state using `pipe` predicate instead of `empty` predicate.

5.3.2 The Outside-in Strategy Specification

The Outside-in strategy proposed by Professor Leon Sterling is a strategy he uses to solve FlowFree puzzles. It centres around prioritising connections of like-coloured dots that are placed more outwardly in the puzzle. There are two rules associated with this strategy:

1. Start with the outermost layer of nodes in the puzzle grid, and check if there are any dots placed on it.
    a. If two like-coloured dots are placed on the edge, connect them by sticking to the outermost layer of nodes if possible.
        i. In the case where a dot is placed in a node that is directly adjacent to a corner node and the other like-coloured dot is placed in one of the other outermost nodes, the connection to the latter dot should traverse the node at the nearest corner to the former dot. This is to avoid the creation of ‘dead-end’ nodes, where there are no possible moves once a connection reaches that node. A ‘dead-end’ node is illustrated in Figure 18.
    b. If there is only one dot of a certain colour placed on the edge, connect it to the other like-coloured dot by sticking to the outermost layer as much as possible, and then by sticking to edges as much as possible. Refer to the Edgy Strategy Domain section for the definition of an ‘edge’.

2. Recursively repeat step 1 for the inner layers of nodes, working from the outer layers towards the centre of the grid.

For each rule, if there is more than one pair of like-coloured dots that satisfy the specified conditions, a pair can be selected at random.

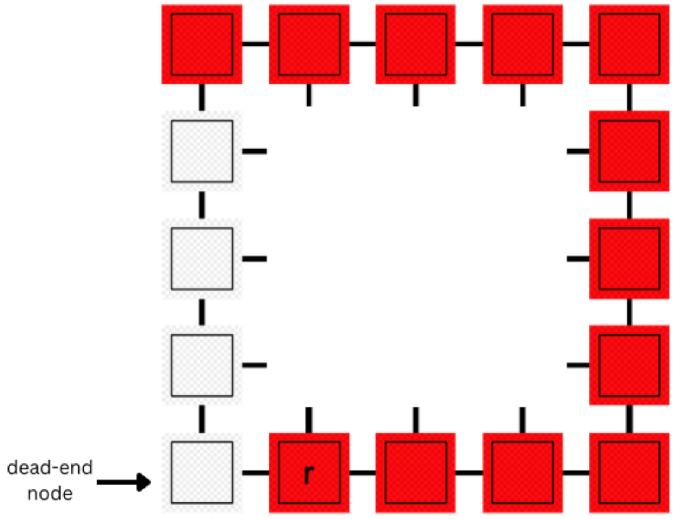

Figure 18: Illustration of a dead-end node. Inner nodes have been removed from this figure for simplicity. Connecting the two red dots in this fashion results in the dead-end node shown in the figure. No colour, other than red, would be able to use this node. Thus, connecting the red dots using the empty nodes would be the correct way of connecting them.

Figure 19 shows the order in which dots will be prioritised under the Outside-in strategy. This strategy does not include a domain definition because it has not been fully formalised. Its solutions have been manually expressed in .plan files which can be used by Planimation for their visualisation.

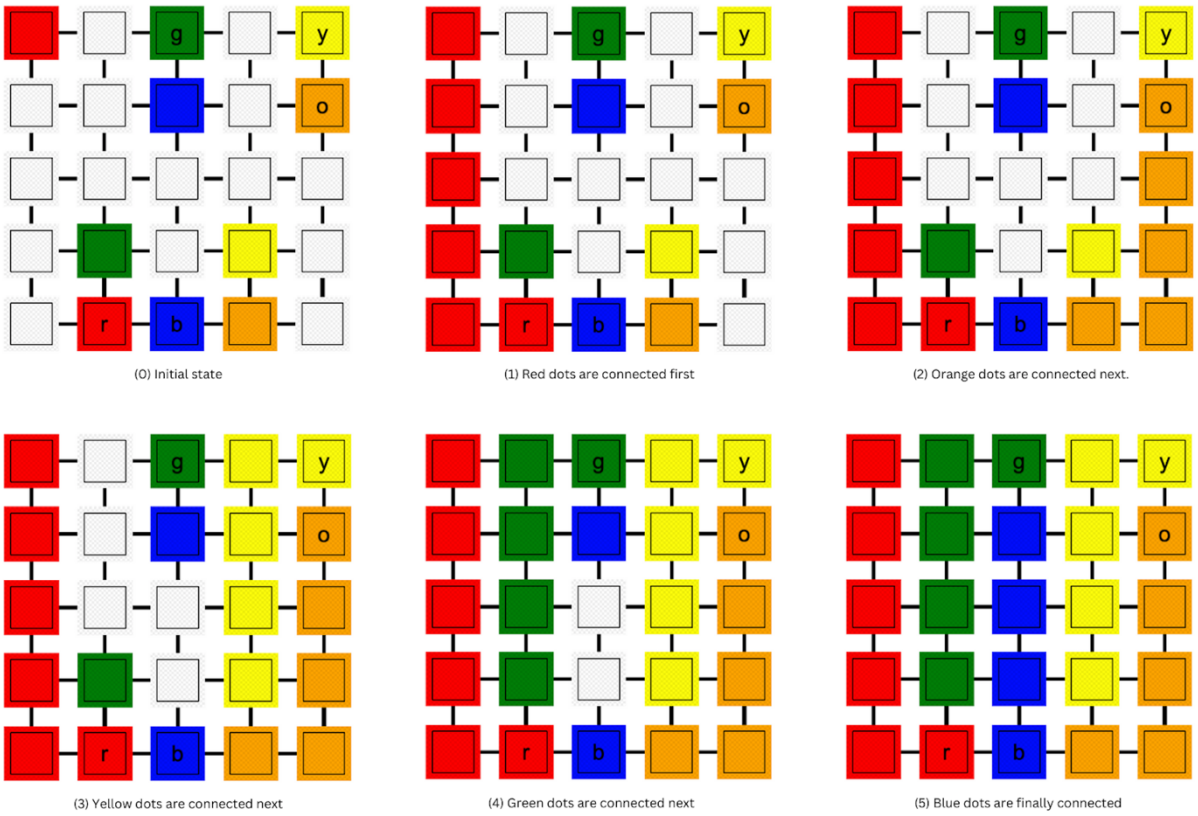

Figure 19: Connection sequence under the Outside-in strategy.

5.4 Animation Profile

The animation profile developed to visualise problems and their solutions on Planimation is shared across the pre-strategy and the Edgy strategy domains. Five predicates from these domains are visualised in the animation profile. The other predicates specified in these domains are not necessary in the visualisation of the problem or the solution.

The following predicates have been specified in the animation profile.

- (head ?n ?d): if this predicate holds true for node ?n and dot ?d, then the x and y coordinates of ?d are set equal to the x and y coordinates of ?n. The colour of ?n will also be set equal to the colour of ?d.

```
(:predicate head
    :parameters (?n ?d)
    :effect (
        (equal (?d x) (?n x))
        (equal (?d y) (?n y))
        (equal (?n color) (?d color))
```

```
        )
    )
```

Figure 20: Predicate block for `head`.

- (adjacent ?n1 ?n2): if this predicate holds true for nodes ?n1 and ?n2, then a line is drawn between ?n1 and ?n2.

```
(:predicate adjacent
     :parameters (?n1 ?n2)
     :effect (
         (action (function draw_line (objects ?n1 ?n2)))
     )
 )
```

Figure 21: Predicate block for `adjacent`.

- (filled ?n ?d): if this predicate holds true for node ?n and dot ?d, then the x and y coordinates of ?d are set equal to the x and y coordinates of ?n. The colour of ?n is also set equal to the colour of ?d.

```
(:predicate filled
     :parameters (?n ?d)
     :effect (
         (equal  (?d x) (?n x))
         (equal  (?d y) (?n y))
         (equal (?n color) (?d color))
     )
 )
```

Figure 22: Predicate block for `filled`.

- (selected ?d): if this predicate holds true for dot ?d, the colour of the current colour selection node is set equal to the colour of ?d.

```
(:predicate selected
     :parameters (?d)
     :custom color-obj
     :effect (
         (equal (color-obj color) (?d color))
     )
 )
```

Figure 23: Predicate block for `selected`.

- (position ?n): all nodes, ?n for which this predicate holds true, will be uniformly placed around a point (i.e. in a grid structure). It is important to note that all node objects in the problem file need to be named using the convention “node[x]-[y]”, where x represents the row and y represents the column, and node1-1 is at the bottom-left of the grid.

```
(:predicate position
     :parameters (?n)
     :effect (
         (assign (?n x y) (function distribute_grid_around_point `
                                 (objects ?n)))
     )
 )
```

Figure 24: Predicate block for `position`.

Visual blocks were specified for the following objects.

- node: each node is represented by a white-coloured ‘cell’ image with the following properties.

```
(:visual node
    :type default
    :properties (
        (prefabImage cell)
        (showname FALSE)
        (x NULL)
        (y NULL)
        (color WHITE)
        (width 75)
        (height 75)
        (depth 1)
    )
)
```

Figure 25: Visual block for `node` object.

- node representing the current colour selection: this node is placed at the bottom left of the visual in Planimation and bears the following properties.

```
(:visual color-obj
     :type custom
     :objects color-obj
     :properties (
         (prefabImage cell)
```

```
            (showname FALSE)
            (x NULL)
            (y NULL)
            (color NULL)
            (width 75)
            (height 75)
            (depth 2)
        )
    )
```

Figure 26: Visual block for current colour selection node.

- dot: a dot uses the 'cell' image of its own colour (i.e. a red dot would use a red-coloured cell) and has the properties shown below. The following code block is specified for each coloured dot with the appropriate fields modified accordingly.

```
(:visual dotr
        :type predefine
        :objects r
        :properties (
            (prefabImage cell)
            (showname TRUE)
            (x NULL)
            (y NULL)
            (color RED)
            (width 75)
            (height 75)
            (depth 2)
        )
    )
```

Figure 27: Visual block for dot object.

5.5 Plan Generation and Visualisation

Due to the technical challenge described in Section 6.2, the following workflow was used to generate plans for FlowFree puzzles using the Edgy strategy and visualise them using Planimation.

1. Generate plan using Fast Downward's seq-sat-lama-2011 planner, and Edgy strategy domain and problem files.

2. Transfer the generated plan to a .plan file.
3. On http://editor.planning.domains, upload the animation profile, domain, problem, and plan files.
4. Comment out any `:action-costs` related code from the domain and problem files.
5. Run Planimation using these files.

6 Technical Challenges

Links to all references in this section are included in the appendix.

6.1 Lack of Error Messages on Planimation

The first challenge involved the Planimation framework being unable to display error messages. This hindered the development workflow of the animation profile (AP) file as without error messages, it became extremely difficult to isolate and resolve any errors in this file. After some investigation it was discovered that the new Planimation frontend, developed using JavaScript (JS), had not been configured to display error messages, in contrast to the old frontend that was developed using Unity. As updating the Planimation frontend was forecasted to be a relatively time-consuming task, a workaround was implemented. Instead of using the Planimation frontend, the Postman API Platform was used to directly communicate requests to the Planimation API. The domain file, problem file and animation profile were included in the body of each request (see Figure 28 below).

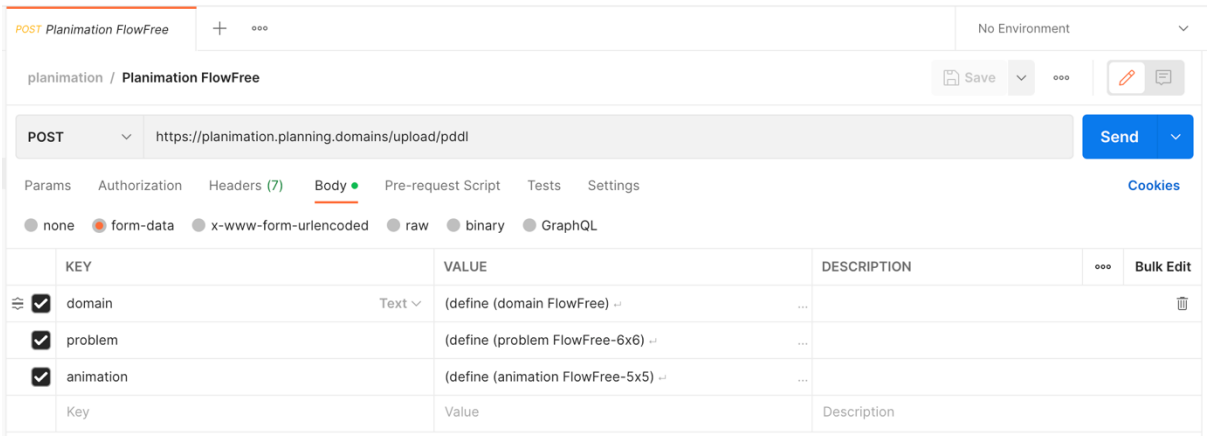

Figure 28: Postman setup for Planimation API

Using Postman, error messages could easily be received and utilised (see Figure 29 below).

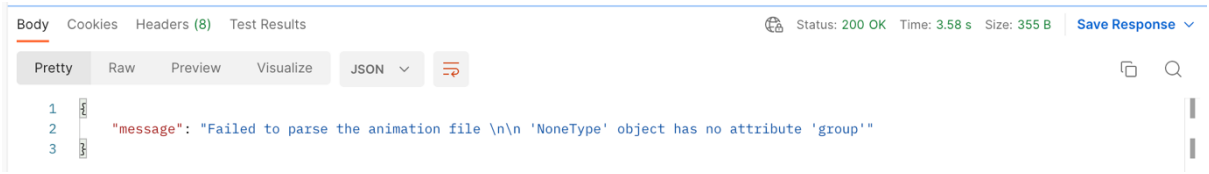

Figure 29: Sample error message from Planimation PDDL on Postman

6.2 Resource Limits on planning.domains Solver

The initial solver used for solving FlowFree puzzles in this research project was the one utilised by planning.domains. This solver uses iterative-width (SIW) and if this algorithm doesn't yield a solution, it uses breadth-first-search (BFS) to find a solution. Its implementation can be found at the LAPKT-dev Github repository. The online solver also has a time limit of 5 seconds and a memory stack limit of 500 megabytes (MB). While the online solver was sufficient in solving the pre-strategy FlowFree domain, it was inadequate for the Edgy strategy-based domain. Hence, a new solver with higher or no resource limits had to be utilised.

6.3 Fast Downward Planners

Fast Downward planners are not compatible with certain PDDL requirements. The list of incompatible requirements can be found on the Planning Wiki page. As such, the original Edgy strategy domain had to be re-written and re-organised to avoid using

PDDL requirements that are not supported by Fast Downward planners. In this case, the `:fluents` requirement had to be removed from the original Edgy strategy domain and the :action-costs requirement had to be introduced to maintain the same functionality.

Fast Downward planners also do not support negative goals in certain problem files when these negative goals give rise to axioms. This was the case for Edgy strategy problem files and as such, the empty predicate was replaced by the pipe predicate, which is its negative predicate (i.e. `(not (empty ?n))` = `(pipe ?n)`). This allowed the goal state to be expressed using predicates that are not negated.

7 Results & Analysis

7.1 Pre-strategy AI Solver vs. Human Strategies

Figures 30 and 31 show how the pre-strategy AI solver and the Edgy strategy solve a FlowFree puzzle, respectively. The Edgy strategy and the Outside-in strategy yield the same solution for this puzzle, and thus when comparisons are made between the pre-strategy AI solver and human strategies, only the Edgy strategy will be used for argument.

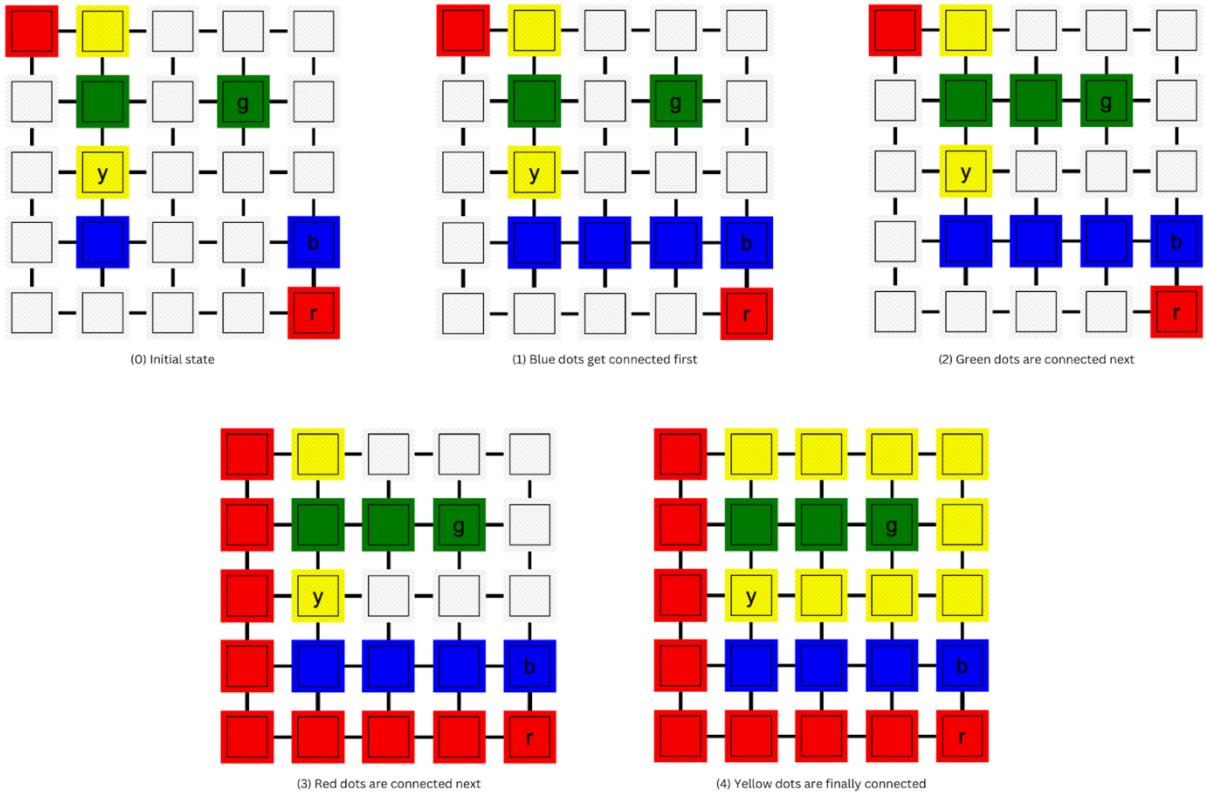

Figure 30: Solution produced by pre-strategy AI solver

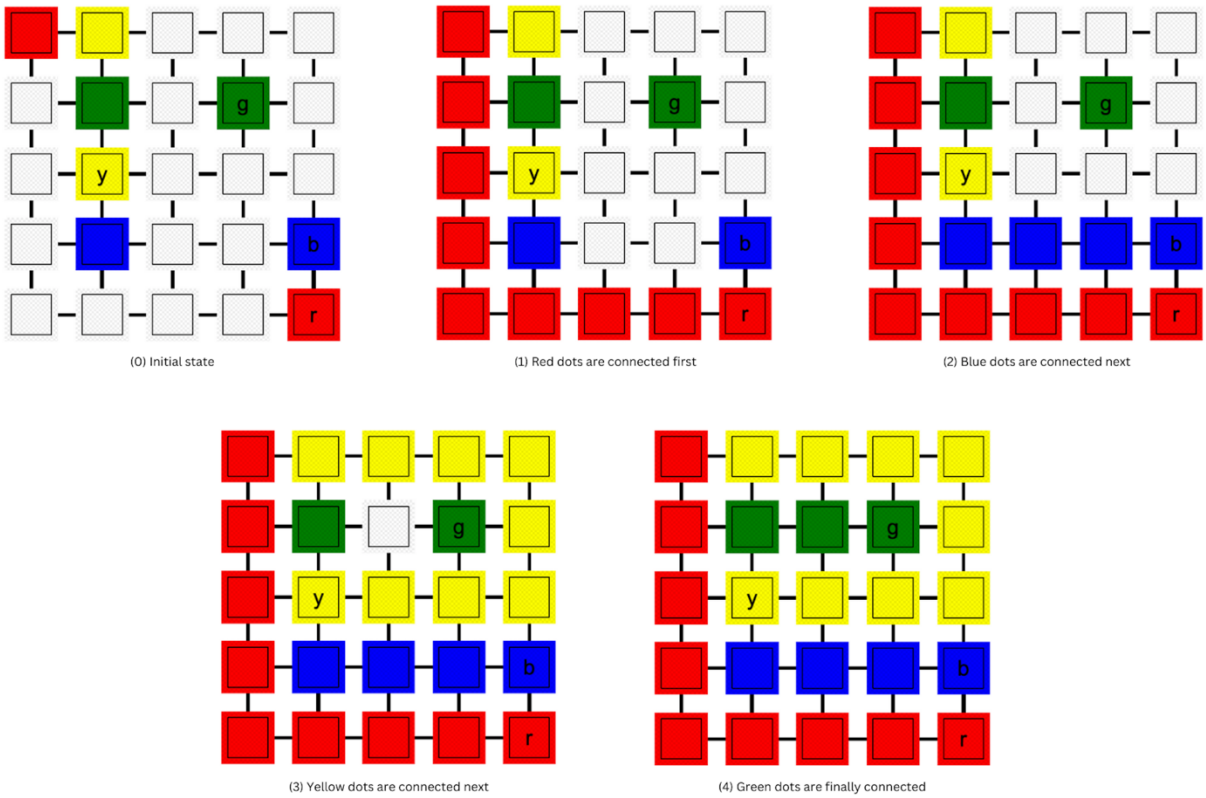

Figure 31: Solution produced by Edgy strategy (and Outside-in)

It can be argued that the reasoning behind each of the moves produced by the pre-strategy AI solver is unclear, especially to humans outside the field of AI planning. In step 1 of Figure 30, the blue dots are connected in the manner shown. While this connection is correct, it may not be obvious to a human player at this stage of the puzzle as there are other means of connecting the blue dots. Some form of heuristic is required for a human to determine that this connection is indeed the correct one. When the Edgy strategy is used, the blue dots are connected after the red dots are connected (by entirely sticking to edges). Once the red dots are connected, it might give the human player more confidence that the blue connection mentioned earlier is correct. This is because the blue connection sticks entirely to edges once the red dots have been connected. A similar argument holds for the connection between the green dots in step 2 of the pre-strategy AI solver and step 4 of the Edgy strategy. In both cases, the connection of the yellow dots in the manner shown can be reasoned with confidence.

For more examples on how the pre-strategy AI solver and Edgy strategy solve FlowFree puzzles, please refer to the source code in the appendix.

7.2 Edgy Strategy vs. Outside-in Strategy

The puzzle shown in Figures 30 and 31 is an example of a puzzle where both the Edgy strategy and the Outside-in strategy can yield the same solution.

Each strategy is also capable of producing more than one valid solution for certain puzzles. For example, the Edgy strategy can solve the puzzle shown in Figure 32 in more than one way. In this case, the AI solver has chosen one of these solutions arbitrarily (the one shown in the figure). The alternative solution under this strategy is, in fact, the solution produced by the Outside-in strategy, as shown in Figure 33. The only differences between the two strategies are in steps 2-4.

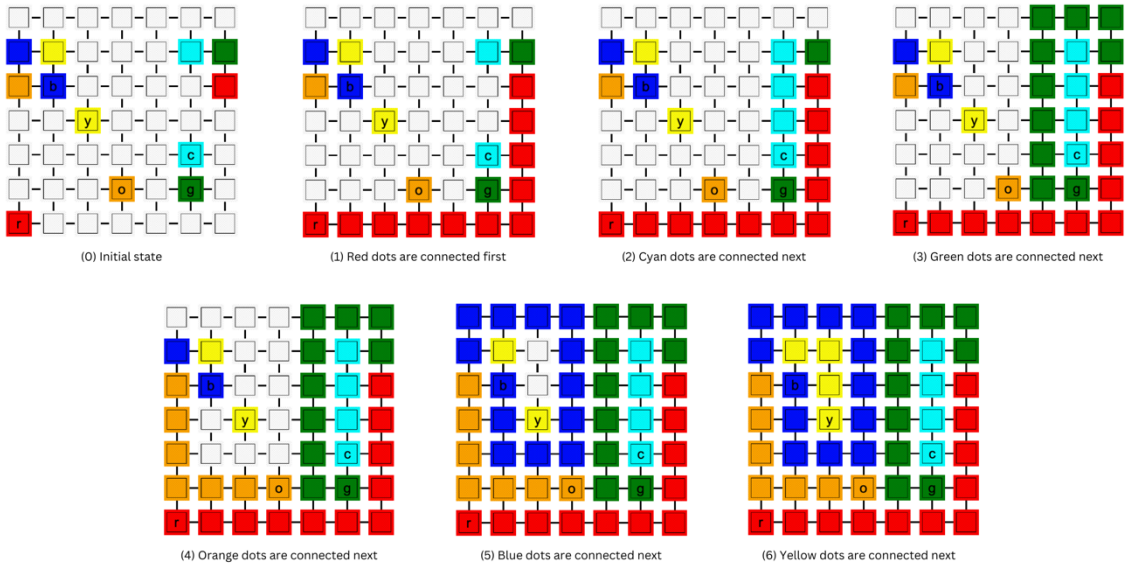

Figure 32: Solution produced by Edgy strategy for a 7x7 FlowFree puzzle

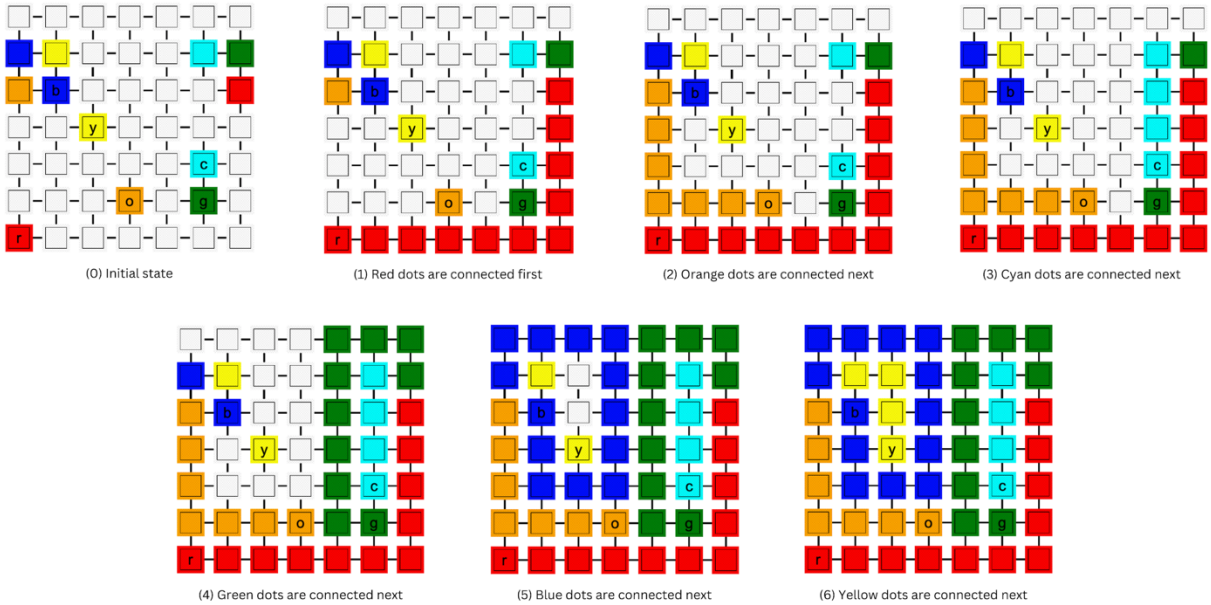

Figure 33: Solution produced by Outside-in Strategy for a 7x7 FlowFree puzzle (can also be considered as an alternative solution by Edgy strategy).

The Outside-in strategy, with its current level of definition, is not freestanding in certain cases. Figure 34 shows the solution produced by the Edgy strategy for a 6x6 FlowFree puzzle. The Edgy strategy is freestanding in this case and can solve the puzzle without using any other strategy. However, the Outside-in strategy cannot do so without employing the Edgy strategy; this is observed when connecting the red and green dots. The correct pipe connection for these dots can only be determined by using the Edgy strategy, as the Outside-in strategy mainly provides a definition for

which colours should be prioritised. It does not necessarily provide a rule for how pipe connections should be formed. This is where the Edgy strategy proves useful. In a way, the Outside-in strategy can be considered a sub-strategy of the Edgy strategy, placing more priority on dots that are placed more outwardly in the puzzle grid and inherently making pipe connections stick to edges as much as possible.

It is important to note that the Edgy strategy may not always be freestanding, especially for larger puzzles. It may need to either be coupled with other strategies or human intuition to optimally solve a puzzle. Despite this, an AI solver using the Edgy strategy can still solve a puzzle successfully due to its computational and memory capacities, but perhaps at the cost of interpretability.

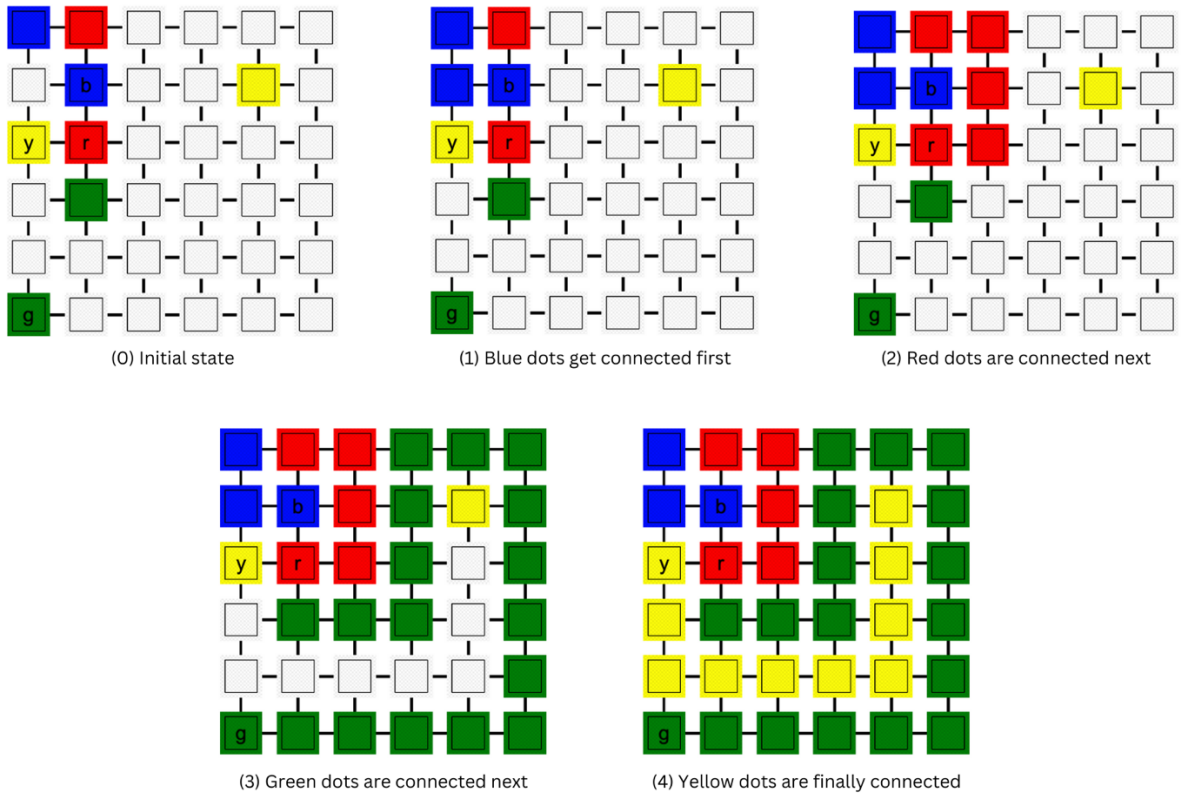

Figure 34: Solution produced by Edgy strategy on a 6x6 FlowFree puzzle. It is also the solution produced by the Outside-in strategy if it is used in conjunction with the Edgy strategy.

8 Discussion

The analysis of the strategies in Section 7 was made easier with Planimation by expressing the solutions they yielded as a series of puzzle states. There was no need to revisit code, and the analysis could be done in plain English rather than in technical terms. Instead of only visualising plans generated by pre-strategy AI solvers, Planimation now visualises plans generated by AI solvers using a human strategy, adding an extra layer of interpretability.

The cost function proved to be very useful in directing the solver to generate actions that a human player would use when using the Edgy strategy. Using a cost function could potentially be a standard way of defining human strategies, as most strategies favour certain actions over others even though these actions might all be equally reasonable. As such, each type of action can be assigned a certain cost, with the goal being to minimise the cost as much as possible. In the case of the Edgy strategy, actions that move the head of a pipe connection to a node that is not next to an edge incur a cost of one.

The pre-strategy domain, described in Section 5.1, does not make use of any human strategies. It uses the 'black-box' search algorithm native to the AI solver and its computational and memory capacities to solve puzzles. The addition of the human strategy (Edgy) to the pre-strategy domain still uses the solver's search algorithm and capabilities, but with higher interpretability. The solver remains a 'black box', but the solutions it produces are more explainable since they resemble the solutions produced by a human player using the Edgy strategy.

While predicates would most likely differ depending on the domain, a few of the design concepts applied to the Edgy strategy domain can be generalised to human strategies used in other domains. Cost functions, as explained earlier in this section, is one such concept. Another concept is the splitting up of an action into multiple actions. Often, when action costs are state-dependent, it is favourable to have multiple versions of the same action – each having a different cost and pertaining to a different set of preconditions (or state).

The Edgy and Outside-in strategies are human strategies that are arguably relatively simple to understand when visualised in Planimation. However, the same cannot be claimed for strategies in domains more complex than FlowFree (e.g., chess) at this stage. Creating a PDDL domain inclusive of human strategies for a complex game like chess and using Planimation to create effective visualisations that make these strategies interpretable could increase confidence in Planimation's capabilities and vision of making planning problems more amenable to humans.

9 Significance & Implications (Contribution)

Through this research, FlowFree has become the first domain whose human strategies have been successfully visualised in Planimation. Planimation has been a major tool in identifying the similarities and shortcomings of these strategies through its visualisation capabilities, and hopefully this domain is the first of many.

The pre-strategy domain created as part of this research lends itself to users who are interested in encoding strategies they use in FlowFree to assess and analyse them, and possibly enables them to improve their strategies as a result of their analysis.

This research is another example of collaborative learning – combining the powers of AI systems and human strategies to achieve robustness and interpretability simultaneously. AI systems bring their computational and memory capacities to the solution, while human strategies bring intuition and interpretability.

10 Future Work

The Outside-in strategy has not been fully formalised as part of this research. Sufficient formalisation of this strategy will allow for encoding of this human strategy onto the pre-strategy domain. This will enable complete comparison with the Edgy strategy. Other human strategies could also be formalised and encoded for further comparison.

A human player using the Edgy strategy would sometimes find it more intuitive to start a pipe connection from one dot rather than the other like-coloured dot. Currently, the Edgy strategy domain does not incorporate this intuition. It requires the creator of the PDDL problem file to arbitrarily choose a dot from each pair of dots from which pipe connections should start. Formalising this intuition and including it in the domain would potentially make the AI solver more amenable and interpretable to humans.

The pre-strategy and Edgy strategy domains have only been tested on a small number of FlowFree puzzles as part of this research. There is no guarantee that these domains will successfully yield a solution for all FlowFree puzzles. Mathematically proving that these domains can solve any FlowFree puzzles in the manner intended is advantageous as they can confidently be used to solve larger and more complex puzzles than the ones demonstrated in this thesis for proof of concept.

To formally prove claims in this thesis that the human strategy visualised in Planimation (i.e., the Edgy strategy) is more amenable to humans than the 'normal' solution used by the AI solver, controlled experiments need to be conducted. One way to conduct such an experiment could be to gather a statistically significant group of people and have each person watch the solution produced by a 'normal' AI solver, recount the sequence of moves as accurately as possible, and then do the same with a solver using a human strategy. The hypothesis is that the group will be able to recollect the sequence of moves produced by the solver using the human strategy more accurately than that produced by the 'normal' AI solver.

Instead of recollecting the sequence of moves displayed by the two solvers, the group could be asked to conjecture the reasoning behind each move for each of the solvers. The hypothesis here is that the group will be able to guess the reasoning of each move more accurately for the solution produced by the solver using the human strategy than the one produced by the 'normal' AI solver.

A human strategy was successfully encoded into a PDDL domain and visualised on Planimation as part of this research. However, there was no systematic approach to

creating this encoding. Further research into whether a systematic method exists or if one can be created could be valuable as this would considerably reduce the amount of time spent on encoding each human strategy.

11 Conclusion

The aim of this research was to advance XAI by expressing and visualising human strategies in Planimation. A human strategy used to solve FlowFree puzzles was encoded in PDDL and its sequential solutions were visualised in Planimation. Another human strategy for FlowFree was specified sufficiently to manually create plans and visualise them in Planimation. The solutions yielded by these strategies were compared with one another and with the solutions yielded by the 'normal' AI solver. The visualisation capabilities of Planimation allowed for critical analysis of each strategy and identification of what the 'normal' AI solver lacks in terms of interpretability.

The pre-strategy FlowFree domain designed in this research lends itself to users who are inclined to encode their own strategies to analyse or improve them.

This research is an example of collaborative learning, where the powers of AI systems and human strategies are combined to yield robust yet comprehensible solutions. While FlowFree is the first domain for which human strategies have been visualised in Planimation, it hopefully sets the stage for the exploration of other domains.

12 References

Chen, G., Ding, Y., Edwards, H., Chau, C. H., Hou, S., Johnson, G., Syed, M. S., Tang, H., Wu, Y., Yan, Y., Tidhar, G., & Lipovetzky, N. (2020). *Planimation*. Planimation; planimation.github.io. https://planimation.github.io/documentation/

Akula, A., Liu, C., Todorovic, S., Chai, J., & Zhu, S.-C. (n.d.). Explainable AI as Collaborative Task Solving. In *IEEE Xplore*.

Kim, J., Banks, C. J., & Shah, J. A. (2017). Collaborative Planning with Encoding of Users' High-level Strategies. In *31st AAAI Conference on Artificial Intelligence*. Massachusetts Institute of Technology, Norfolk State University.

Bian, S. (2021), A Case Study in Using Planimation Software for Solving Planning Problems (Masters research thesis), University of Melbourne

13 Appendix

- Source code: https://github.com/abasu89/planimation-strategy-encoder
    - The source code contains PDDL domains, problems, animation profile and problem generation template (for the pre-strategy domain) of FlowFree puzzles.
- Planimation API: https://planimation.planning.domains/upload/pddl
- Postman API platform: https://www.postman.com/api-platform/
- planning.domains: http://editor.planning.domains
- LAPKT-dev Github repository: https://github.com/LAPKT-dev/LAPKT-public/tree/Devel2.0/src/planner/siw_plus-then-bfs_f
- Fast Downward: https://www.fast-downward.org/
- Incompatible PDDL requirements for Fast Downward planners: https://planning.wiki/ref/planners/fd#support

Word count (excluding code excerpts and this line): 6602