# Pockets Pack Levels (1-30) | Flow Free Wiki | FandomProject Summary

*Flow free is a single player mobile game developed by Big Duck, an American studio. Players are faced with a grid of different-colored pairs of dots. The goal is to connect the right dots while filling up every square on the grid. We connect enough dots so that the entire grid has a color and all cells are filled. While there are more complex versions of this game (with bridges and blockades) we wish to focus on the very basic iteration of flow free that was derived from the Number link logical puzzle.*

# Propositions

1. *Path(color, cell) 🡪 True, if a cell belongs to a path of a specific colour.*
   1. *For example, Path(“blue,” “c11”) is true if cell (1,1) belongs to the blue path.*
2. *Connection(color, cell1, cell2): 🡪 True, if there is a connection between cell1 and cell2 for a specific color.*

# Constraints

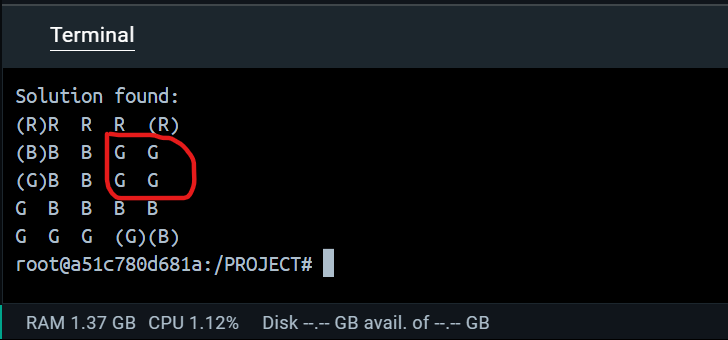
1. *Path Formation 🡪 Paths must form a continuous connection from start to end points.* 
   1. *Starting and ending cells of a color must belong to that color’s path* 
      1. ***Path(color, start\_cell) ∧ Path(color, end\_cell)***
   2. *If a cell is part of a path, at least one neighbouring cell must also belong to the same path*
      1. ***Path(color, cell1) 🡪 Connection(color, cell1, cell2) ∧ Path(color, cell2)***
2. *Cannot have more than two connections at the same time.*
   1. ***¬(Connection(color, c, neighbor1) ∧ Connection(color, c, neighbor2) ∧ Connection(color, c, neighbor3))***
3. *If cell1 and cell2 are adjacent, they must belong to the path of the same color.* 
   1. ***Connection(color, cell1, cell2) 🡪 (Path(color, cell1) ∧ Path(color, cell2))***
4. *If c is part of a path color, then there must exist at least one neighbour n such that c is connected to n for that color.* 
   1. ***Path(color, cell) 🡪 (Connection(color, cell, cellneighbor1) V Connection(color, cell, cellneighbour2) V Connection(color, cell, cellneighbour3) V Connection(color, cell, cellneighbour4)***
5. *If the cell is part of the color path, then it needs to have at least one valid connection to a neighbouring cell.* 
   1. ***Connection(color, cell, neighbor) 🡪 (Path(color, cell) ∧ Path(color, neighbor)***
6. *If there is a connection between the cell and its neighbor, then both the cell and its neighbor must belong to the path for the given color.* 
   1. ***Connection(color, cell, neighbor) 🡪 (Path(color, cell) ∧ Path(color, neighbor))***
7. *If there is a connection from cell1 to cell2, there must also be a connection from cell2 to cell1.* 
   1. ***Connection(color, cell1, cell2) 🡨🡪 Connection(color, cell2, cell1)***

# Model Exploration

*We used the Python library to solve our model and obtain example solutions for the constraints. The solutions were visualized using a grid-based approach, where each cell was annotated with its assigned color path, and endpoints were marked distinctly. This helped us verify the following aspects:*

* ***Path Continuity:*** *Paths between specified start and end points for each color were successfully established in most solutions.*
* ***Exclusivity:*** *Each cell was assigned to at most one path, confirming that our exclusivity constraints were functioning as expected.*

***Observation:*** *While most paths adhered to the defined constraints, we discovered an issue where in some cases, some cells formed disconnected sub-paths or freestanding cycles that were not connected to any valid endpoint. These cells were erroneously assigned to a path, even when they were unreachable.*

*A screenshot of a computer program

Description automatically generated*

***Testing Constraints***

*To debug and validate our constraints, we performed the following steps:*

1. ***Negating Suspected Redundant Constraints:***
   * *We tested whether the exclusivity constraint (at most one path per cell) was truly necessary by negating it and checking if a solution still existed.*
   * ***Result:*** *No solution existed, confirming that this constraint was integral to the model.*
2. ***Analyzing Partial Assignments:***
   * *To investigate the disconnected sub-path issue, we added constraints to force specific cells into paths and observed the impact on the overall solution.*
   * *For example, we forced certain unreachable cells to be excluded from any path to test whether their assignments were redundant.*
3. ***Cycle Detection Testing:***
   * *We manually inspected solutions for cycles or disconnected sub-paths. By adding negated clauses for detected cycles, we tested whether the model could generate solutions without these invalid structures.*

*To better understand the model’s behavior under specific conditions, we also tested partial assignments. For example:*

* *Forcing particular start or end points to be unreachable to verify how the model adjusted the paths.*
* *Assigning specific cells to a path while excluding others to evaluate continuity constraints.*

***Result:*** *These tests revealed that while the connectivity constraints ensured valid paths between start and end points in most cases, the freestanding cells issue persisted.*

*We have attempted several solutions to fix this issue, such as conditioning path membership on reachability, but have not yet found a comprehensive solution.*

# Jape Proofs

# First-Order Extension

Extending our projecting into First-order logic can be done by describing the constraints that we have implemented in code, in predicate logic. For that, we need first to describe the domain of discourse for the Flow Free game and then formalize the game’s constraints in predicate logic.

The following is a semantic key describing the notation for the domain and functions we will use for the predicate logic:

* **A** – Domain of discourse (A is a finite set that contains 2n cells where n is the dimension of the grid.) **Note: We assume all grids are square as defined by the game**
* **G(x) –** x is a flow free grid
* **P(x)** – x is a cell(point on the grid)
* **E(x)** – x is an endpoint (Point of origin for the line)
* **C(x)** – x is a color
* **Colored(x,y)** – x is colored y
* **Conn(x,y) -** x is connected to y

**Now we use the functions above to describe our constraints in predicate logic:**

* **All endpoints have exactly one non-endpoint neighbor.**

* **All non-endpoint cells have exactly two neighbors.**
* **All cells have one color.**
* **All connected cells are connected bidirectionally (symmetric connection constraint).**
* **All connected cells are the same color.**