# Project Summary

Kanoodle Jr. is a game where pieces of various colours and shapes need to be placed on a 5x5 grid. Initial puzzle configurations require certain squares to be coloured a certain way. This restricts where the pieces can be placed. This project aims to model any given initial configuration as a SAT theory, and a solution will correspond to the proper placement of all pieces in the game.

Here, you can see the board with an initial configuration and the set of pieces for the Kanoodle Jr. game.

# Propositions

There are three main propositions for the Kanoodle Jr. encoding:

* **PieceConfig(col, config)**: Piece of colour **col** is using configuration **config** (e.g., rotation). Most have 4 configurations, but some have 8 (flip it over).
* **PlacePiece(col, config, x, y)**: Piece of colour **col** and configuration **config** is at location (**x**,**y**). One location on the piece is considered the origin (0,0), and the locations on the rest of the piece are defined by its shape.
* **PlaceColour(col, x, y)**: The colour at location (**x**,**y**) is set to **col**. This is used to tie together the overlapping pieces, find the final solution, and restrict what colours are forced to be where as part of the initial puzzle.

# Constraints

## Only one configuration for a colour

## Don’t allow pieces out of bounds

## At most one piece can be placed at a location

## Can only place a piece in one place

## If a piece is picked for a location, then the appropriate configuration is chosen

## Every location can take on at most one colour

## If a piece is placed, then it forces the colour of the cells that make up the piece

## Initial board configuration is satisfied

# Model Exploration

*List all the ways that you have explored your model – not only the final version, but intermediate versions as well. See (C3) in the project description for ideas.*

## Visualizing the Model

Just printing the solution from Bauhaus was tough to see what was going on, so we used the **tabulate** library to illustrate where pieces go. For example:

Table

Description automatically generated with medium confidence

Brackets show the colour of the square, and capital shows the piece colour placed.

Ideas on how to improve this for the final:

* Move to shaded colour background for the pieces that are placed.
* ..

## Fixing the Overlap

At one point, we were facing an issue where pieces would be overlapping. An example:

A picture containing table

Description automatically generated

To fix this, we looked into adding constraints that prevented overlap. This ultimate lead to a new proposition (the third one discussed above) that says what colour a location should be.

# Jape Proofs

Because the model is too large to fit everything, we will prove sequents over a far reduced size of kanoodle board. 2x2 or 3x3 are the most likely dimensions. Also, we will only consider pairs of pieces.

To discuss the ideas, consider blue and red pieces for this 3x3 board:

Table

Description automatically generated with low confidence

Ideas for what we might want to prove in Jape:

* Placing a blue piece facing left (on the far right side) means that only one of two possible red placements are possible.
* Placing a piece in a particular location (e.g., red in the top left) will force another square to be red because of where the piece goes.
* I can deduce anything (anything!) if I end up putting two red pieces down.

# First-Order Extension

*Describe how you might extend your model to a predicate logic setting, including how both the propositions and constraints would be updated.* ***There is no need to implement this extension!***

## Predicates

These will mirror closely to the propositions defined above. We have objects in the first-order setting that correspond to colours, configurations, and coordinates. These “types” will also have a predicate defined for them.

* **Colour(col)**: object **col** is a colour
* **Config(c)**: object **c** is a configuration
* **Coord(c)**: object **c** is a coordinate
* **PieceConfig(col, config)**: Piece of colour **col** is using configuration **config** (e.g., rotation).
* **PlacePiece(col, config, x, y)**: Piece of colour **col** and configuration **config** is at location (**x**,**y**).
* **PlaceColour(col, x, y)**: The colour at location (**x**,**y**) is set to **col**.

## Constraints

### More complex predicates use the right types

* Arguments to the PieceConfig predicate are Colour and Config objects:  
   col. config. ( PieceConfig(col, config) (Colour(col) Config(config)) )
* Arguments to the PlacePiece and PlaceColour are similarly defined:  
   col. config. x.y. ( PieceConfig(col, config, x, y) (Colour(col) Config(config) Coord(x) Coord(y)) )  
   col. config. x.y. ( PieceConfig(col, x, y) (Colour(col) Coord(x) Coord(y)) )

### Every location can take on at most one colour

Variant 1:  
 x.y. ( (Coord(x) Coord(y)) ( col1.col2. (col1 = col2) (PlaceColour(col1, x, y) PlaceColour(col2, x, y))))

Interpretation:  
For all pairs of objects, if they are coordinates, then for every pair of colours, either they are the same or the PlaceColour doesn’t hold for both colours.

Variant 2:  
 x.y. ( (Coord(x) Coord(y)) ( col1.col2. ((col1 != col2) (PlaceColour(col1, x, y) PlaceColour(col2, x, y)))))

Variant 3:  
 x.y. col1.col2. ((col1 = col2) (PlaceColour(col1, x, y) PlaceColour(col2, x, y)))

There is no need to include the types of the objects x, y, col1, col2, because of the constraint above that defines the types for PlaceColour.

# Useful Notation

*Feel free to copy/paste the symbols here and remove this section before submitting.*