Final Design / 05.03.22

Kyle Sadler
Lindsey Albin
Ben Hughes
Christopher Souvanouphong

Final Presentation

TileScad

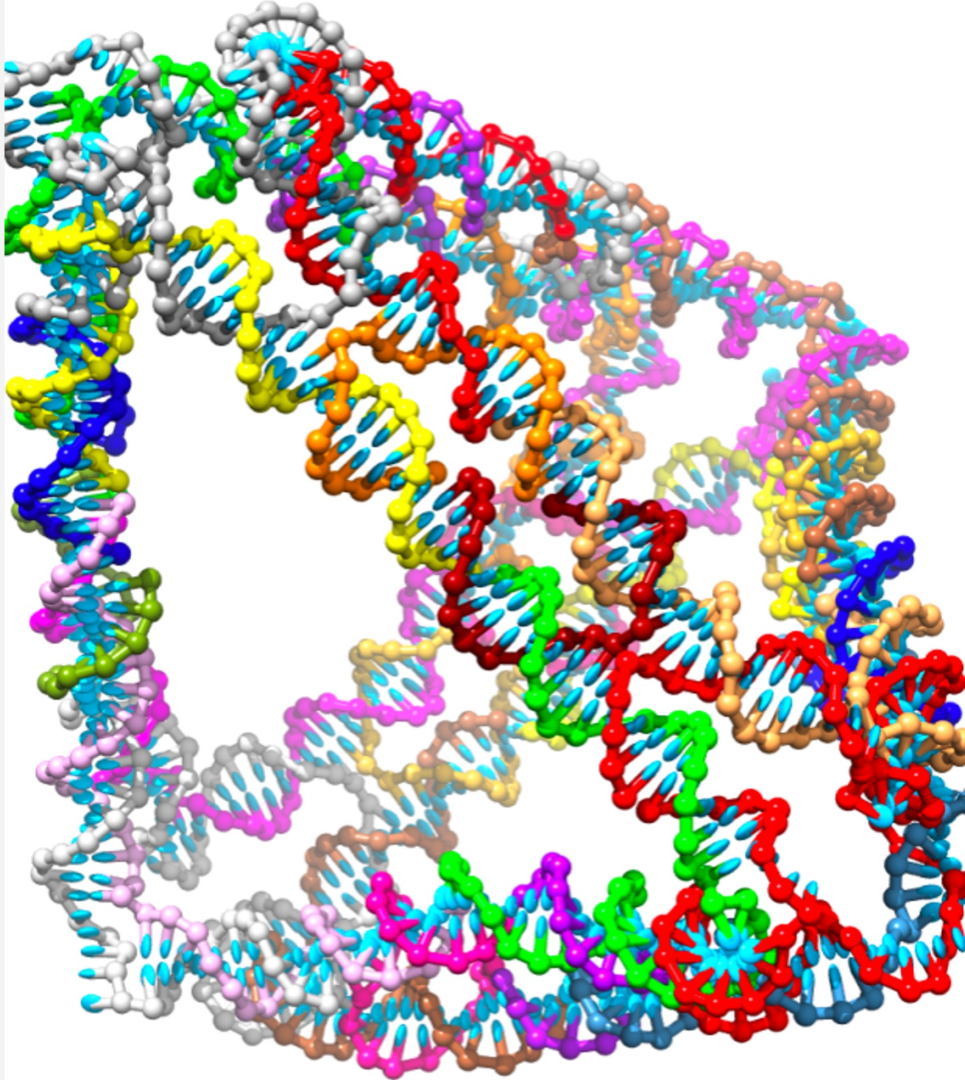
Project Overview

Dr. Trent Rogers
DNA Nanotechnology
Problems
Goals

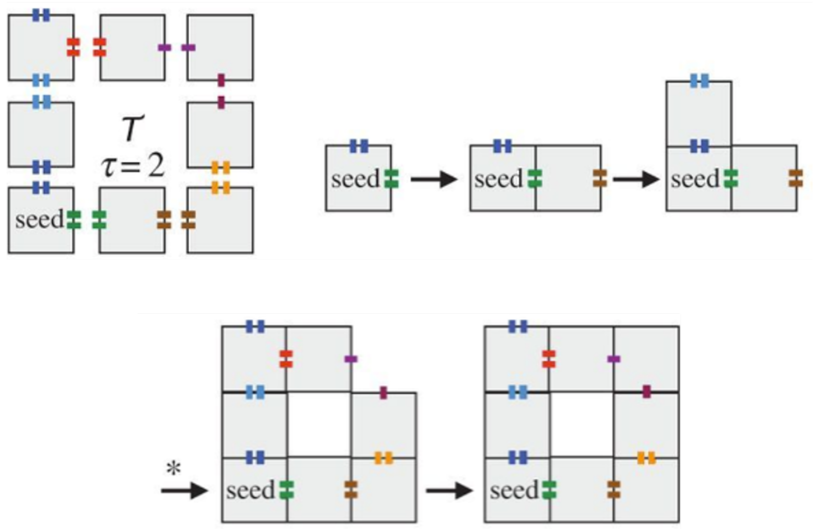
Dr. Trent Rogers

Project Sponsor

- ➔ Graduated from the University of Arkansas with a PhD in Computer Science in 2019
- ➔ Postdoctoral researcher at Maynooth University researching self assembling and self organizing systems
- ➔ National Science Foundation Graduate Research Fellow
- ➔ Recipient of the Doctoral Academy Fellowship as a PhD student



DNA Nanotechnology



Woods, Phil. Trans. R. Soc. A 2015

- ➔ Strands of synthetic DNA can be designed to bind in a controlled and predictable process which allows arbitrary DNA structures to be created at the nanoscale
- ➔ DNA strands are conceptually organized into rectangular structures called tiles
- ➔ Tiles are the fundamental building unit in tile assembly structures

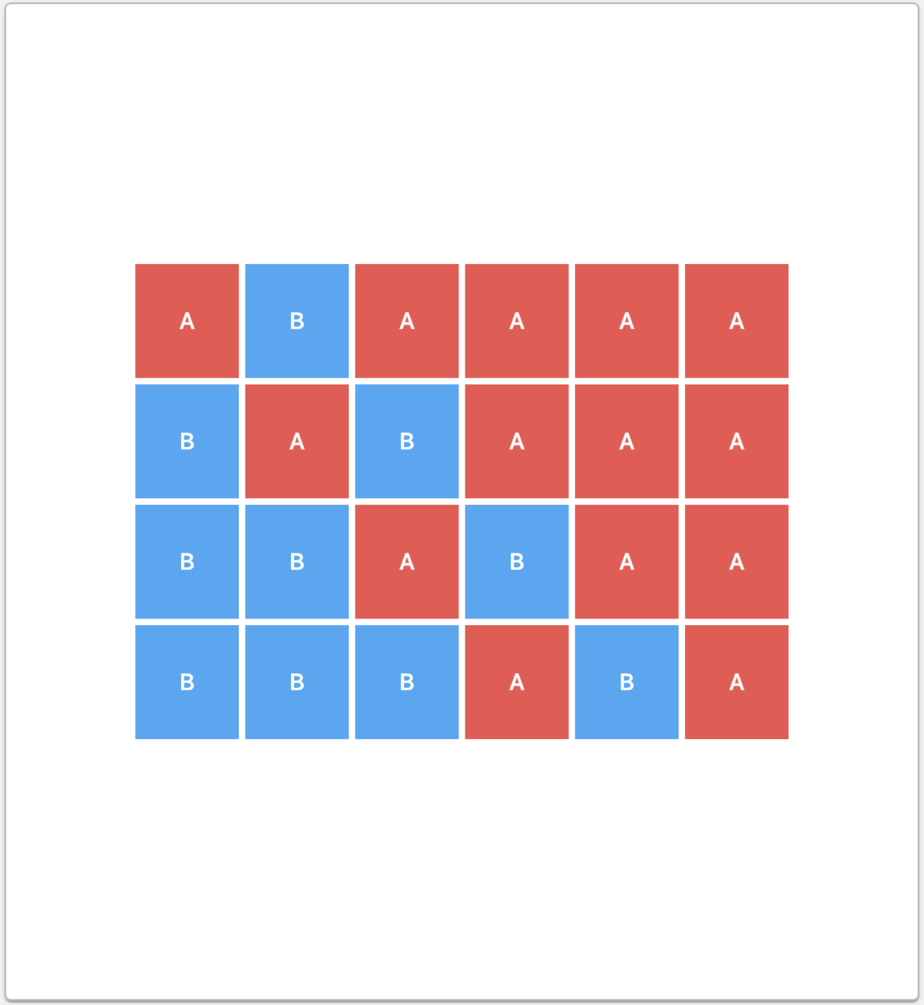
Problems

- ➔ Current process of transforming nanostructure designs into a set of DNA strands is **tedious** and requires many independent software packages.
- ➔ Researchers **waste time** importing, exporting, transferring, and re-formatting DNA strand data.
- ➔ **Errors and mismatches** in DNA strands can **waste thousands of research dollars.**

TileScad

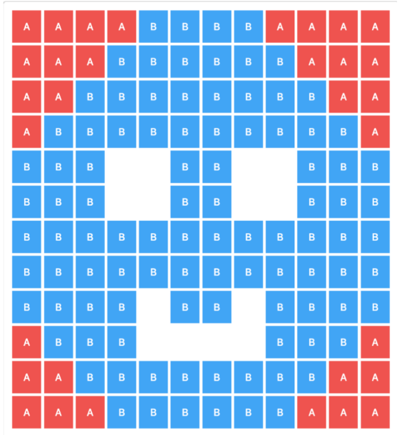
The Objectives

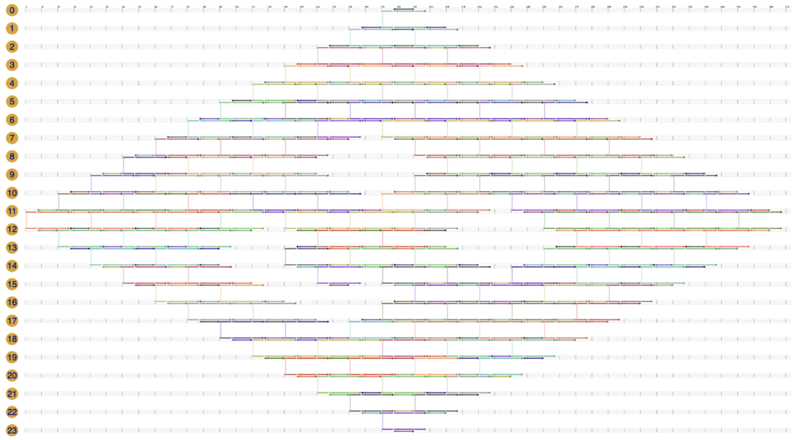
- ➔ To easily design an abstract nanostructure tile assembly
- ➔ Transform the design into a DNA strand diagram with a click of a button
- ➔ No longer having to use multiple independent software packages



Design Overview

Requirements
Architecture
Interface

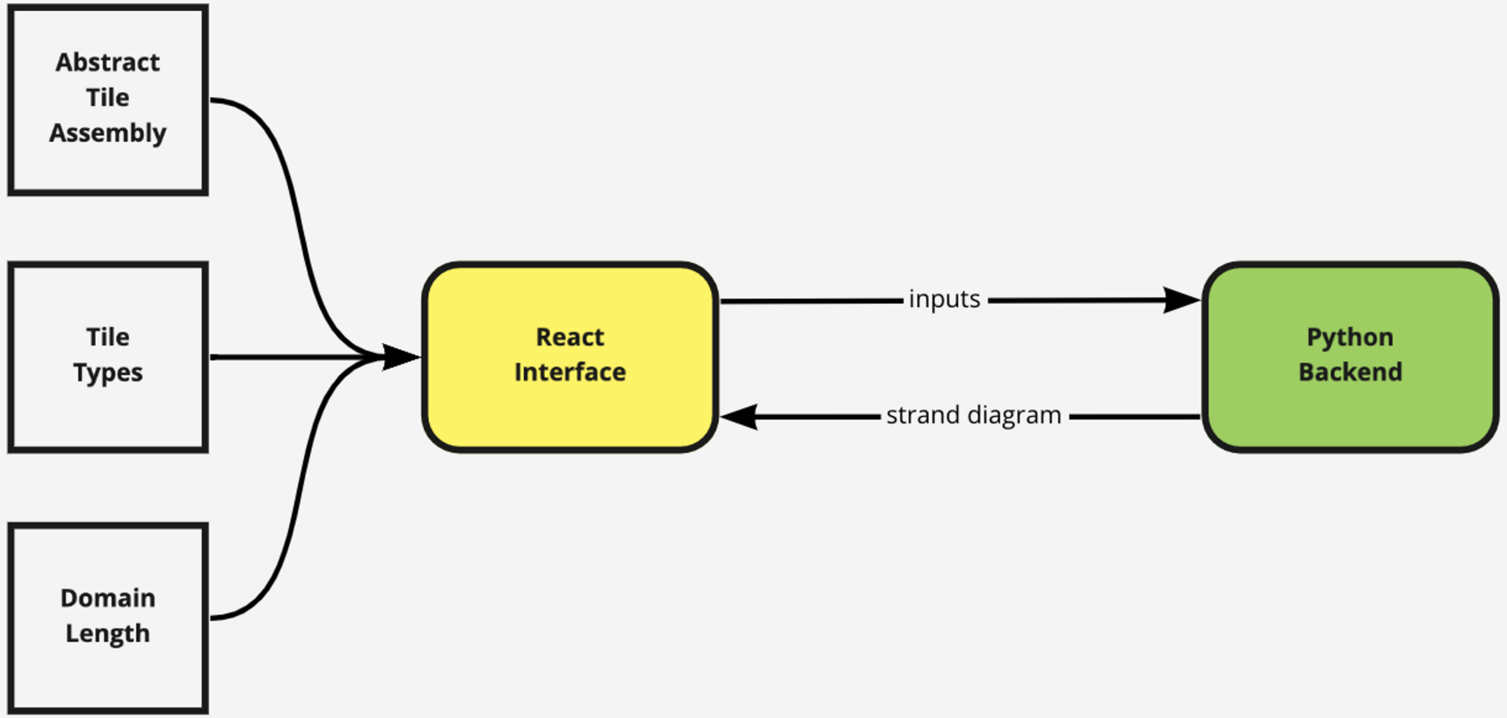

Design: Abstract tile assembly


Output: Strand diagram Scadnano file

Requirements

- ➔ User can design a DNA tile assembly on an arbitrarily sized canvas
- ➔ Each tile can added can be customized
- ➔ User receives a DNA strand diagram that can be downloaded
- ➔ App hosted on a web server
- ➔ Clear and understandable user interface and user experience

High-Level Architecture



Implementation Overview

Front-End Implementation

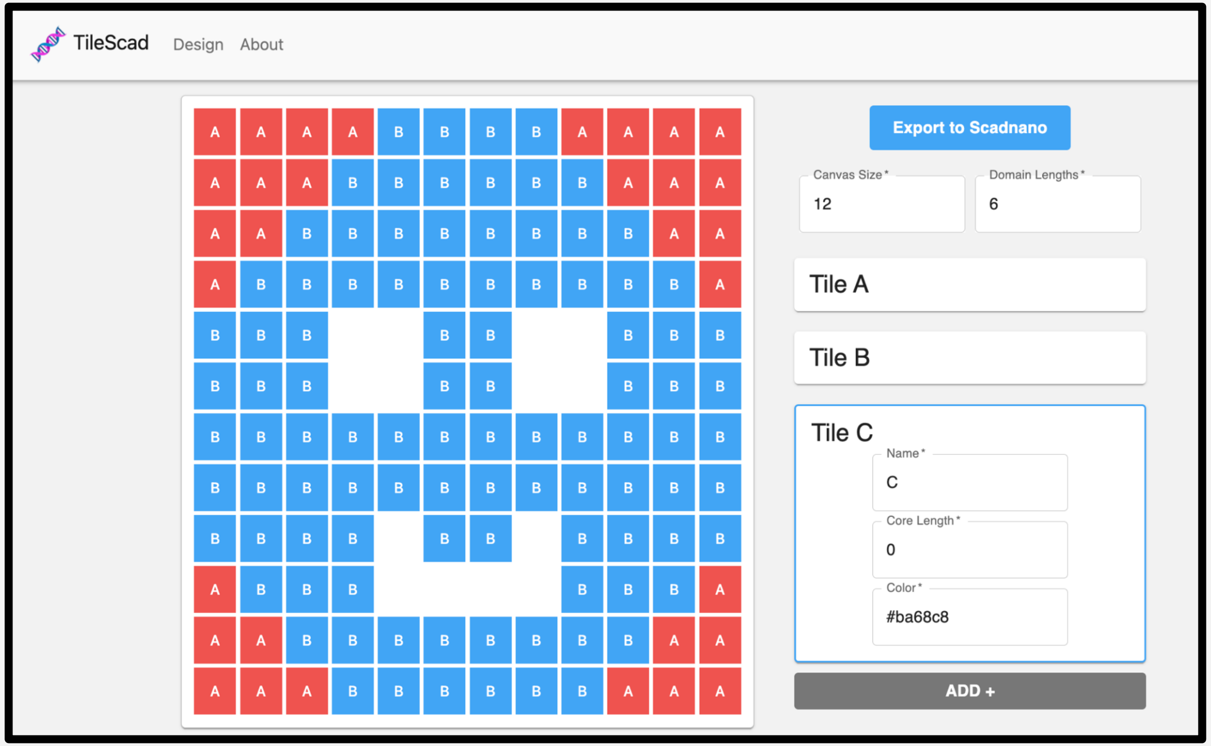
➔ The user interface is built using React 16.4.0 and has two main components: the Tile Assembly Canvas and the Tile Menu.

➔ The front end uses React Material UI 5.5.0 to customize the look and feel of the interface.

➔ **Tile Assembly Canvas:** the canvas was created using a Material UI Card component and HTML div elements. React onClick listeners were added to each tile location to add tiles when the user clicks on the canvas.

➔ **Tile Menu:** the tile menu uses Material UI Card, Button, and TextField components.

Front-End Implementation



A	A	A	A	B	B	B	B	A	A	A	A
A	A	A	B	B	B	B	B	B	A	A	A
A	A	B	B	B	B	B	B	B	B	A	A
A	B	B	B	B	B	B	B	B	B	B	A
B	B	B			B	B			B	B	B
B	B	B			B	B			B	B	B
B	B	B	B	B	B	B	B	B	B	B	B
B	B	B	B	B	B	B	B	B	B	B	B
B	B	B	B		B	B		B	B	B	B
A	B	B	B					B	B	B	A
A	A	B	B	B	B	B	B	B	B	A	A
A	A	A	B	B	B	B	B	B	A	A	A

Canvas Size * 12

Domain Lengths * 6

Tile A

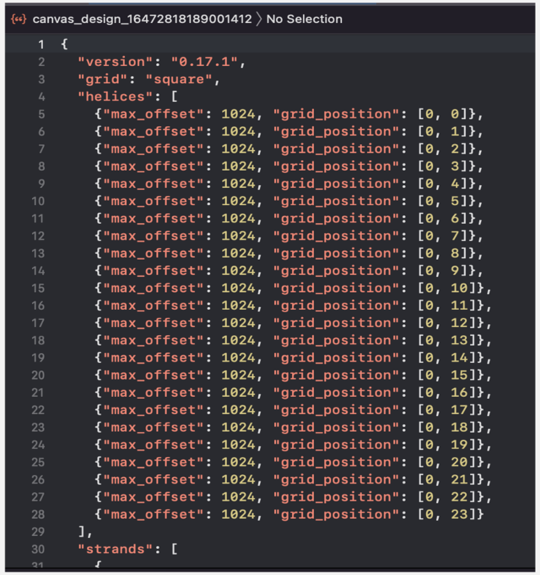
Tile B

Tile C

Name * C

Core Length * 0

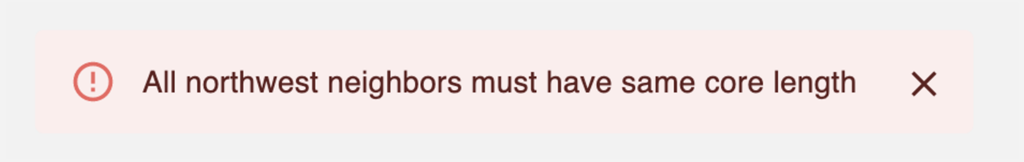
Color * #ba68c8



```
{
  "version": "0.17.1",
  "grid": "square",
  "helices": [
    {"max_offset": 1024, "grid_position": [0, 0]},
    {"max_offset": 1024, "grid_position": [0, 1]},
    {"max_offset": 1024, "grid_position": [0, 2]},
    {"max_offset": 1024, "grid_position": [0, 3]},
    {"max_offset": 1024, "grid_position": [0, 4]},
    {"max_offset": 1024, "grid_position": [0, 5]},
    {"max_offset": 1024, "grid_position": [0, 6]},
    {"max_offset": 1024, "grid_position": [0, 7]},
    {"max_offset": 1024, "grid_position": [0, 8]},
    {"max_offset": 1024, "grid_position": [0, 9]},
    {"max_offset": 1024, "grid_position": [0, 10]},
    {"max_offset": 1024, "grid_position": [0, 11]},
    {"max_offset": 1024, "grid_position": [0, 12]},
    {"max_offset": 1024, "grid_position": [0, 13]},
    {"max_offset": 1024, "grid_position": [0, 14]},
    {"max_offset": 1024, "grid_position": [0, 15]},
    {"max_offset": 1024, "grid_position": [0, 16]},
    {"max_offset": 1024, "grid_position": [0, 17]},
    {"max_offset": 1024, "grid_position": [0, 18]},
    {"max_offset": 1024, "grid_position": [0, 19]},
    {"max_offset": 1024, "grid_position": [0, 20]},
    {"max_offset": 1024, "grid_position": [0, 21]},
    {"max_offset": 1024, "grid_position": [0, 22]},
    {"max_offset": 1024, "grid_position": [0, 23]}
  ],
  "strands": [
```

Front-End Error Checking

- ➔ Determines if every vertical column in the resulting Scadnano design contains tiles of the same core width.
- ➔ We can ensure the assembly is valid by checking that each northwest-southeast diagonal in the assembly contains tiles with the same width.
- ➔ If the tile assembly is invalid, an error message appears.

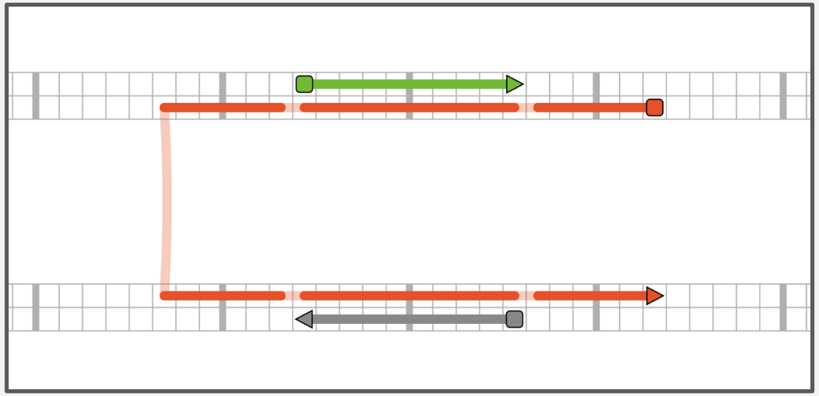

All northwest neighbors must have same core length

Back-End Implementation

➔ Validates and processes the abstract tile assembly into a scadnano file using the scadnano Python package.

➔ Allows the frontend to download the processed scadnano file.

➔ Server written in Flask 2.0.2 and hosted with Vultr.



Back-End Error Checking

➔ Checks that the input grid, domain length, and tile types are defined and within a valid range.

➔ Determines if the tile assembly is geometrically valid before processing.

Condition	**Error**
Tile grid, tile types, or domain length is null	400 Missing Data
More than 50 tile types	400 Invalid Data
Domain is length greater than 30	400 Invalid Data
Grid larger than 60x60	400 Invalid Data
Invalid tile widths	400 Invalid Data
Code exception	500 Internal Server Error

Demo

Demo

https://tilescad.org/
https://tilescad.org/smiley-face

Demo

https://drive.google.com/file/d/1Nr70RzHfnV8-NvS3fvxiMcJD1g0ziMyg/view?usp=sharing

Demo

Demo