The Game of Sprouts

Graph theory and simulations

Zinnun Malikov

GHP July 2023

Simulation and path-finding algorithms

Introduction to Combinatorics

About Combinatorics:

- Combinatorics and graph theory are fundamental in numerous fields, including computer science, mathematics, and real-world problem-solving.
- The ability to analyze structures and connections between elements (nodes) and relationships (edges) is crucial for modeling and understanding complex systems
- Graph theory, a subset of combinatorics, enables the study of networks, relationships, and connectivity, influencing fields like social networks, logistics, and computer algorithms.

Introduction

Issues:

- Unfortunately, while combinatorics and graph theory are essential to real-world problem solving, topics related to these fields can be confusing at times.
- Many students, including me, initially found the graph theory concepts difficult in the Applied Combinatorics course at Georgia Tech.
- My project addresses the challenge of comprehending combinatorial concepts by creating an interactive game/tool, *Sprouts*.

Introduction to the Game of Sprouts

About Sprouts:

- Sprouts is a mathematical game that was invented by John Horton Conway and Michael S. Paterson in 1967.
- It is a turn-based game played by two players, where the goal is to strategically draw and connect dots to create new lines.
- The game starts with a certain number of dots, n, and players take turns drawing lines between the dots, following specific rules

Introduction to the Game of Sprouts

Rules:

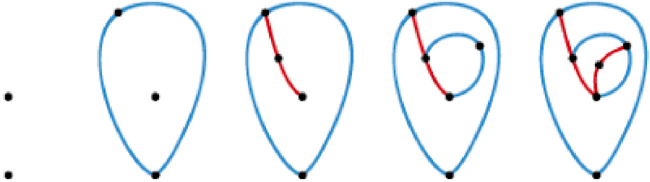
- Each turn consists of drawing a line between two spots (or from a spot to itself) and adding a new spot somewhere along the line.
- The line drawn may be straight or curved but must not self-intersect or cross any other line.
- The new spot cannot be placed on top of one of the endpoints of the new line.
- No spot may have more than three lines attached to it.
- You cannot touch a dot twice with one line then connect it to another.

Introduction to the Game of Sprouts

Terminology

- A **graph** is essentially a collection of vertices and edges that connect vertices.
- A **vertex** is a point on a graph.
- An **edge** is a line which connects two points and is incident to the vertices it touches.
- Two vertices are **adjacent** if there is a single edge connecting them.
- The **degree** of a vertex is the number of edges incident to that vertex.

Example of a Game



Simulating the Game

General Information:

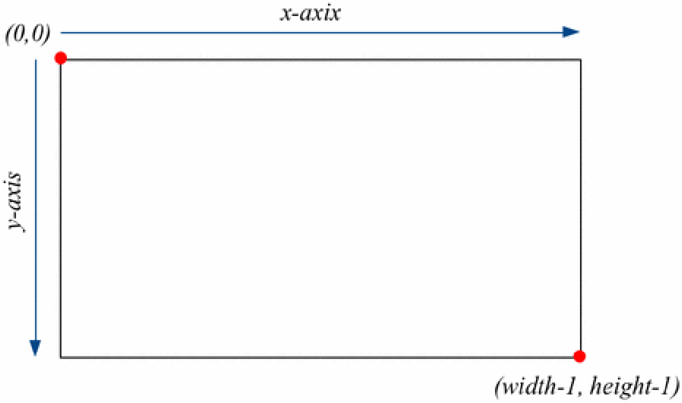
- The simulation of Sprouts runs in a Tkinter canvas using Python, allowing the user to draw curved lines.
- The main loop of the code consists of various functions to check if the rules of Sprouts are met.
- For example, a play only starts if the mouse is clicked near a point on the canvas with degree less than three.
- Moreover, a line is successfully stored only if it does not intersect any other lines and does not result in any degrees greater than 3 once drawn.

Simulating the Game

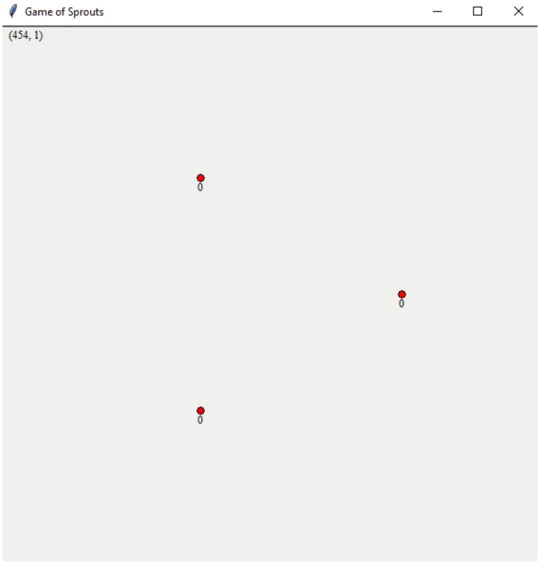
Point Display:

- All figures drawn in the simulation are stored in a coordinate point system.
- The default orientation for the Tkinter is origin $(0, 0)$ at the top left of the canvas and a point (m, n) m pixels right and n pixels down.
- Once a play is initiated, the coordinate position of the user's mouse is appended to a temporary junk storage list.
- If the play is valid, the list of coordinates is added to the main storage list for edges.
- Additionally, all vertices and their respective degrees are stored in separate lists.

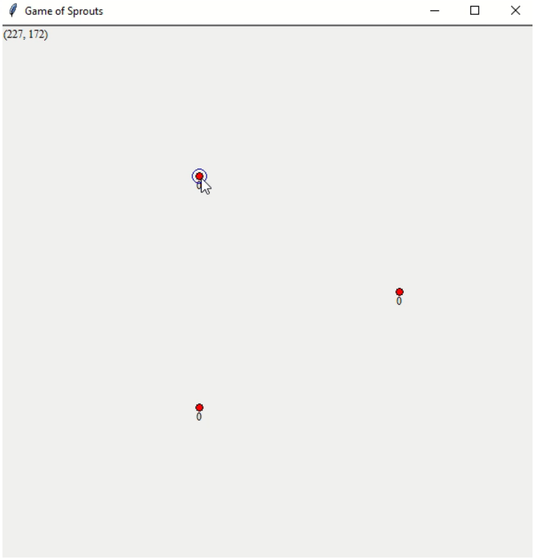
Simulation Coordinate System



Video Demonstration: Invalid Moves



Video Demonstration: Complete Game



Functionality

Data from Simulation:

- The simulation allows the user to play Sprouts, buts its most important feature is storing data representing the coordinates of the edges, the positions of the vertices, and the degrees of each vertex.

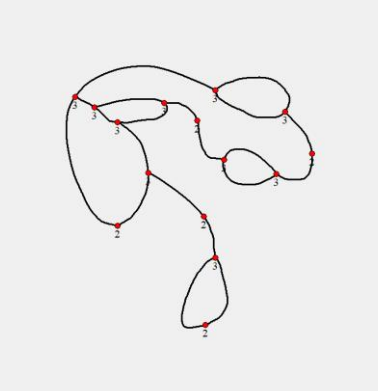
  edge = $[(x_1, y_2), ..., (x_i, y_i)]$, i is the number of points constructing an edge

  vertices = $[(a_1, b_1), ..., (a_k, b_k)]$, k is the number of vertices

  degrees = $[d_1, ..., d_k]$

- These data are stored in a csv file to be used for the path-finding and path-drawing algorithms.

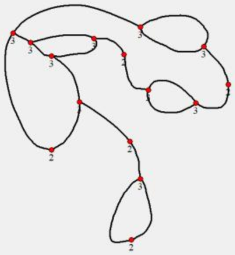
Data Display: Ex 1 – Point Storage



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	3	3	2	3	3	3	3	2	2	3	2	3	3	3	2
3	560	449.4427	270.5573	270.5573	449.4427	193.75	326.8828	431	428.0625	576	624.8828	465	229.0625	356.125	416.0313
4	400	552.169	494.0456	305.9544	247.831	260.3438	397.5391	674	477.3281	287	363.0938	374	279.0469	271.0938	303.0781
5															

points_data

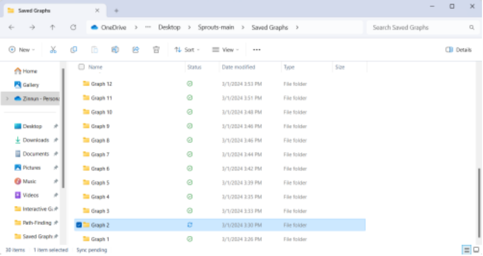
Data Display: Ex 1 – Line Storage



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	449.4427	449.4247	449.3887	449.3347	449.2626	449.1726	449.0645	448.9384	448.7943	448.6322	448.4521	448.254	448.0379	447.8038	447.5516	447.2815	446.9933
3	247.831	247.8214	247.8022	247.7734	247.735	247.6869	247.6293	247.5621	247.4853	247.3989	247.3029	247.1973	247.0821	246.9573	246.8229	246.6789	246.5253
4	193.75	193.2588	192.7764	192.3027	191.8379	191.3818	190.9346	190.4961	190.0664	189.6455	189.2334	188.8301	188.4355	188.0498	187.6729	187.3047	186.9453
5	260.3438	261.0303	261.7158	262.4004	263.084	263.7666	264.4482	265.1289	265.8086	266.4873	267.165	267.8418	268.5176	269.1924	269.8662	270.5391	271.2109
6	270.5573	270.5606	270.5674	270.5775	270.5909	270.6077	270.6279	270.6514	270.6783	270.7086	270.7422	270.7792	270.8195	270.8632	270.9103	270.9607	271.0145
7	305.9544	305.9554	305.9574	305.9605	305.9646	305.9697	305.9758	305.983	305.9911	306.0003	306.0105	306.0218	306.034	306.0473	306.0616	306.0769	306.0932
8	326.8828	326.8975	326.9111	326.9238	326.9355	326.9463	326.9561	326.9648	326.9727	326.9795	326.9854	326.9902	326.9941	326.9971	326.999	327	327
9	397.5391	397.8701	398.2041	398.541	398.8809	399.2236	399.5693	399.918	400.2695	400.624	400.9814	401.3418	401.7051	402.0713	402.4404	402.8125	403.1875
10	449.4427	448.9132	448.3873	447.8623	447.3357	446.8049	446.2675	445.7209	445.1624	444.5897	444	443.1725	442.3008	441.3956	440.4672	439.5263	438.5832
11	552.169	552.0258	552.0051	552.0881	552.2562	552.4905	552.7724	553.083	553.4036	553.7156	554	554.3675	554.7622	555.185	555.6367	556.1181	556.6301
12	431	432.4536	433.8963	435.3277	436.7474	438.155	439.5503	440.9327	442.3021	443.658	445	446.2644	447.5368	448.8088	450.072	451.3181	452.5388
13	674	673.4635	672.8623	672.2071	671.5084	670.7769	670.0233	669.2582	668.4922	667.7359	667	666.3345	665.6914	665.0593	664.427	663.7833	663.1169
14	326.8828	326.8937	326.9154	326.948	326.9914	327.0457	327.1108	327.1868	327.2737	327.3714	327.4799	327.5993	327.7296	327.8708	328.0228	328.1856	328.3593
15	397.5391	397.5444	397.5551	397.5711	397.5924	397.6191	397.6511	397.6884	397.731	397.779	397.8324	397.891	397.955	398.0244	398.099	398.179	398.2643
16	428.0625	428.1875	428.3125	428.4375	428.5625	428.6875	428.8125	428.9375	429.0625	429.1875	429.3125	429.4375	429.5625	429.6875	429.8125	429.9375	430.0625

line_data

Data Display: Ex 1 – Save Folder

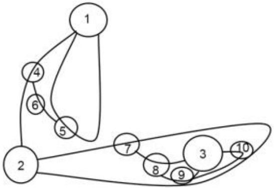


Name	Status	Date modified	Type	Size
Graph.png		2/29/2024 11:55 PM	PNG File	11 KB
line_data.csv		2/29/2024 11:55 PM	Microsoft Excel Co...	177 KB
Plain.png		2/29/2024 11:55 PM	PNG File	8 KB
play_data.txt		2/29/2024 11:55 PM	Text Document	1 KB
point_transform.csv		2/29/2024 11:55 PM	Microsoft Excel Co...	1 KB
points_data.csv		2/29/2024 11:55 PM	Microsoft Excel Co...	1 KB

Path-Existence Algorithms

Existence of Paths:

- Sprouts is said to end when no more valid paths can be drawn between two points.
- It is possible to eyeball solutions to the game, but how can you ensure no paths exist?

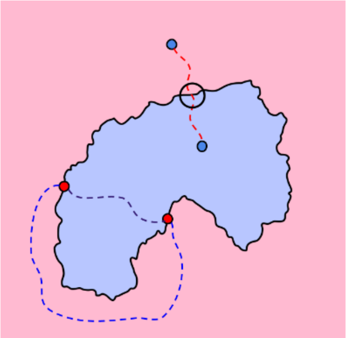


Path-Existence Algorithms

Idea:

- If two points are in the same region, or border the same region(s), then there must be a path entirely within the region(s) between those points that does not intersect any other line and region.

The two red points border the same two regions, so a path exists between these points through each region. However, the two blue points are in different regions, so a line between the two must cross more than one region.



Path-Existence Algorithms

Existence of Paths:

- Identify each distinct closed region G_i, as well as the outside area, in the graph resulting from a position in Sprouts.
- If two points are in the same region, or border the same region(s), then there must be a path entirely within the region(s) between those points that does not intersect any other line and region.
- Different regions can be determined using the flood-fill algorithm.

Path-Existence Algorithms

Flood-fill:

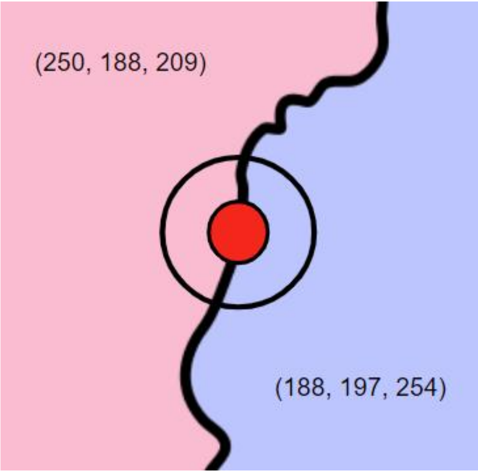
- Each closed region will have one unique color.
- If two vertices border at least one color in common, at least one path exists between these vertices.

Takeaway:

Let A and B be the set of colors two vertices border, respectively. A valid path exists between the vertices if $A \cap B \neq \emptyset$.

Existence of Paths

In the image below, the point borders two colors:



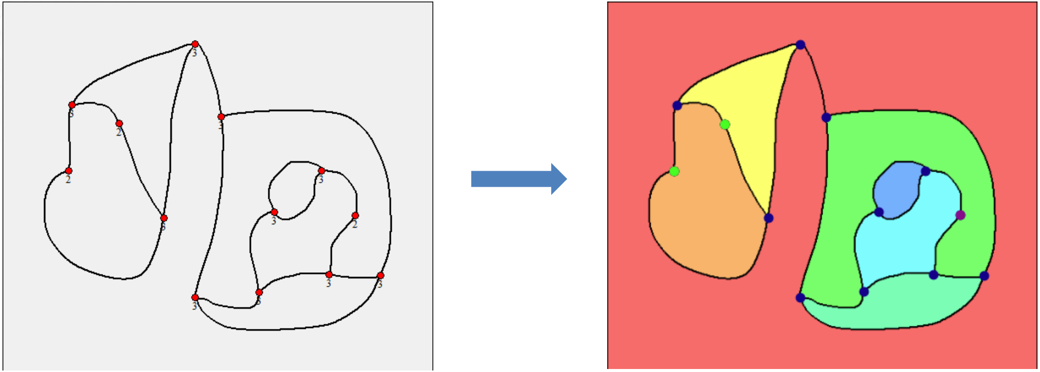
Path-Existence Algorithms

Flood-fill Algorithm

- The Sprouts game saves a picture of the current game position upon saving and updates the CSV data.
- The pixels of the image are analyzed and filled with different colors based on the flood-fill algorithm.
- In the end, every closed region is filled with a unique RGB color.

Flood-Fill

Image conversion using flood-fill



Path-Existence Algorithms

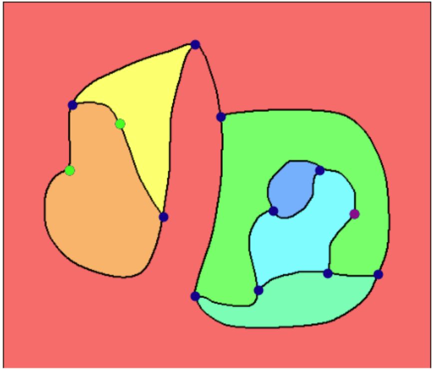
Python Algorithm

- A set of adjacent colors is created corresponding to each vertex with degree 2 or less.
- Combinations of two are created to be used to determine whether the vertices share a region.
- For each combination, the algorithm determines whether a valid path exists or not.

Flood-Fill

Path Existence Using Flood-Fill

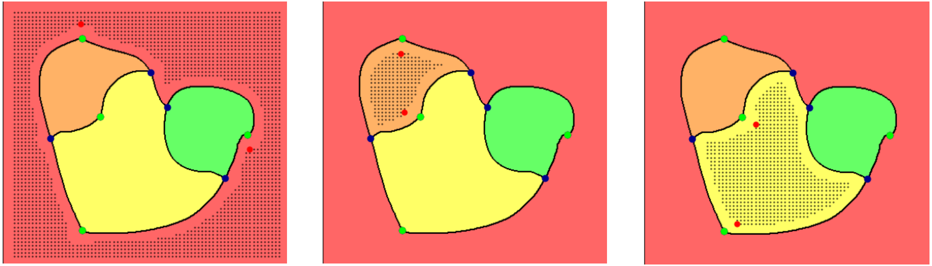
```
Path does not exist from: (78.400390625, 200.0) to (418.853515625, 252.609375)
Path exists from: (78.400390625, 200.0) to (138.470703125, 144.140625) with route {(255, 178, 102)}
Path does not exist from: (418.853515625, 252.609375) to (138.470703125, 144.140625)
* * * * * * *
completed
```



Path-Finding Algorithms

Connecting the Points: **non-closed, simple lines**

- Based on the existence of valid paths between vertices, an algorithm to connect these points can also be constructed.
- First consider the case of lines connecting different vertices. A grid of points can be drawn within the colored regions corresponding to the valid path.



Path-Finding Algorithms

Connecting the Points: **non-closed, simple lines**

- With the grid, the A-star algorithm can be implemented.
- A grid is created consisting of a matrix, where each element is either a 1 or 0.
- A 1 corresponds to a point that can be traversed (i.e. a point within the colored region), and a 0 represents point that cannot be traversed, meaning it is not in the colored region.
- Moves for the path are listed below:
  $(1,0), (-1,0), (0,1), (0,-1), (1,1), (1,-1), (-1,1), (-1,-1)$

Path-Finding Algorithms

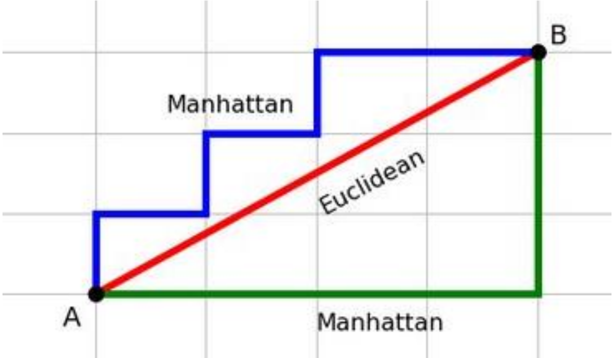
Connecting the Points: non-closed, simple lines

- The A* algorithm uses a combination of the actual cost g to reach a node and an estimated cost h from the current node to the goal node. The heuristic function guides the search towards the goal by providing an estimate of the remaining cost.
- The algorithm explores the nodes with the lowest total cost $f = g + h$ first, which tends to prioritize paths that are closer to the goal.
- Afterward, the grid points used in the path are connected and highlighted to form a path.

Path-Finding Algorithms

Connecting the Points: **non-closed, simple lines**

$$h = |x_1 - x_2| + |y_1 - y_2|$$



Path-Finding Algorithms

Connecting the Points: non-closed, simple lines

- However, there are issues with using a basic heuristic. The generated path will be too close to the existing borders.
- Consequently, it is necessary to implement a modified heuristic that considers the distance of each point to a boundary, D.
- This second component will be added to the original heuristic.

$$h = |x_1 - x_2| + |y_1 - y_2| + a(b - D)$$

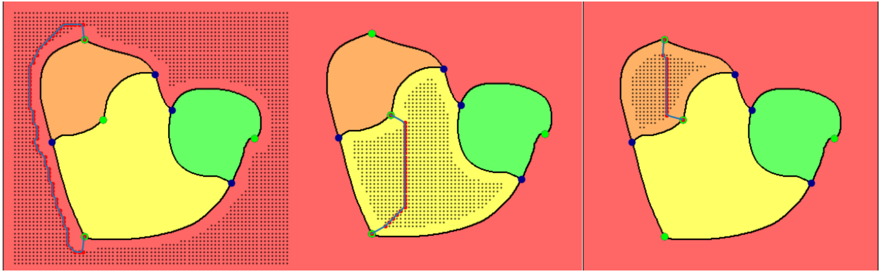
Path-Finding Algorithms

Connecting the Points: non-closed, simple lines

- Python provides an implementation of the priority queue, which is used to store and manage the open nodes during the A* algorithm.
- The priority queue ensures that the node with the lowest f is always selected for expansion.
- In the A* algorithm, the child and parent nodes are tracked using the **parent** dictionary.
- For each node, the **parent** dictionary stores the parent node from which the current node was reached.
- This information is used for reconstructing the path once the goal node is reached.

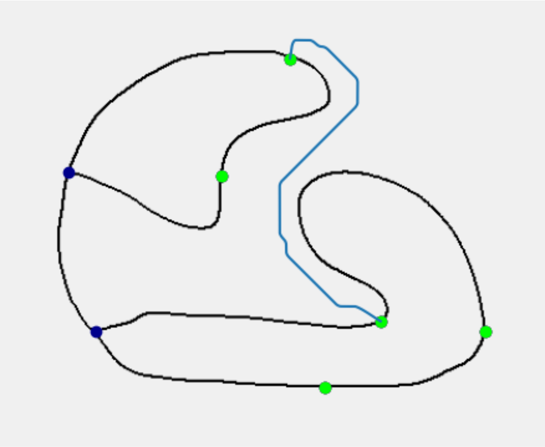
Path-Finding Algorithms

Non-closed, Simple Lines: Examples



Example 1

Basic Path Example:



Path-Finding Algorithms

Connecting the Points: closed lines

- Based on the existence of valid paths between vertices, an algorithm to connect these points can also be constructed.
- Next, consider the case of lines connecting the same vertices.

Path-Finding Algorithms

Connecting the Points: **closed lines**

- For the case of loops, where a vertex connects to itself, color analysis is not needed.

- For loop to be valid, the starting vertex must have degree one or zero. Additionally, the loop must not intersect another line, which is where the difficult comes in.

- Define a pre-determined shape for the loop, such as a tear-drop shape consisting of an arc and two lines.

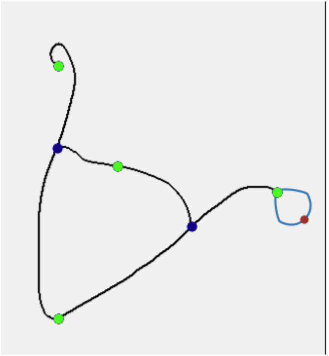
Path-Finding Algorithms

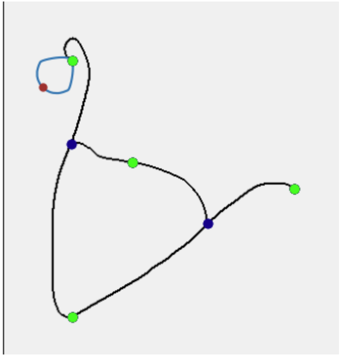
Connecting the Points: closed lines

- The algorithm analyzes the pixels of the image within a gradually decreasing radius, starting at 40 pixels and ending at 20 pixels.
- The tear-drop can have four orientations, analogous to the positions of each quadrant of the coordinate system (top right, top left, bottom left, bottom right)
- If there is a black pixel within a given position and radius, a loop is not drawn.
- A loop is drawn once the first successful iteration is met.

Path-Finding Algorithms

Closed Lines





Graph Analysis

Analyzing Graphs

- In addition to computing paths, *Sprouts* also analyzes existing graphs.
- Analysis consists of determining the properties of regions generated by graphs, showing information about vertices/edges/faces, and highlighting individual components.
- Graph analysis also shows all possible paths in a given configuration

Computer Play

Play against the computer!

- For more user interaction, you can play against the computer!
- The computer will automatically make a move and check the state of the game.
- If no moves can be made, the computer will say so and declare a winner.

Summary

Overview:

- Python simulation of the Game of Sprouts
- Flood-fill algorithm to determine existence of solutions for Sprouts
- Path-finding algorithm to construct valid edges
- Component-determining algorithm for region analysis
- Computer play
- Save file implementation

Next Steps

Expanding on the Project:

- Optimizing and refining algorithms
- Running simulations of the game to gather data about possible strategies
- Sending the game to be used in a class