Engaging Students during Research through the Use of Games

Francisco Gonzalez

The University of Texas Rio Grande Valley

ENGAGING STUDENTS DURING RESEARCH THROUGH THE USE OF GAMES

A Thesis

by

FRANCISCO GONZALEZ

Submitted in Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE

Major Subject: Computer Science

The University of Texas Rio Grande Valley

May 2022

ENGAGING STUDENTS DURING RESEARCH THROUGH THE USE OF GAMES

A Thesis
by
FRANCISCO GONZALEZ

COMMITTEE MEMBERS

Dr. Robert Schweller
Chair of Committee

Dr. Timothy Wylie
Committee Member

Dr. Bin Fu
Committee Member

Dr. Emmett Tomai
Committee Member

May 2022

ABSTRACT

Gonzalez, Francisco, Engaging Students During Research Through The Use of Games. Master of Science (MS), May, 2022, 50 pp., 1 table, 44 figures, references, 32 titles.

Engaging students during a research seminar/meeting can be a difficult challenge, and as as student myself, I can attest to how difficult actively listening to a presentation can be. As such, upon researching more ways to have an audience engaged, one of the most promising concepts is the use of games. Games, in any form, can be very engaging to a person, and even more so if there is active engagement and participation within an audience group. With this concept in mind, I decided to take it upon myself to create a game based around a theoretical computer science model, and see if I can have newcomers learn how the theoretical model works faster than during a normal presentation. I have worked with the concept of games various times before, and as such, I will include that work in this thesis for the sake of theme, and to argue that learning with games tends to be a lot easier than traditional forms of learning.

DEDICATION

I’d like to dedicate this to my family, my friends, and my loving girlfriend; the support I received during this wild ride is something I will hold dear to my heart forever.

ACKNOWLEDGMENTS

I'd like to acknowledge Timothy Gomez, David Caballero, Dr. Timothy Wylie, and Dr. Robert Schweller, for being fantastic research colleagues and mentors. I would not be here academically if it weren't for them.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

1.1 Problem Statement

One general problem in many, if not all, areas of research is audience retention. Especially within the undergraduate area, if we take an example scenario of a class going to a research presentation, after the presentation, how many students would consider taking the time off their day to continue going to said research presentations? Student engagement has been studied extensively in many different subjects, and of course, including Computer Science [25, 20, 27]. With this in mind, and knowing the difficulty of my research group's problems that are looked at, I want to see if we can increase student engagement within this field of theoretical research by using games to teach difficult concepts.

1.2 Statement of Purpose

The main reason for this research is to analyze student engagement within the field of theoretical computer science research; a field filled with tons of interesting problems, but due to the difficulty of picking up on these concepts on first encounter, I would like to make things exciting for students to come back and keep learning. Being able to boost student engagement during theoretical research meetings, it can prove to be beneficial to everyone involved, and at the very least, leaving the student with something that they were able to learn.

1.3 Motivation

There have been instances where I have brought friends over to a research meeting, and while I'll have a basic understanding of what's being presented, I ask my peers after the presentation to see if they got anything from what was presented. More often than not, the ideas/concepts go over their head, which is understandable, as there can be an overwhelming amount of information in one presentation. As such, the whole motivation behind this research was to mitigate the feeling of being overwhelmed, and to put an effort towards making these subjects a bit more digestible through games. At the very least, I'd want new coming students to have learned one key concept of what's being presented.

1.4 Outline

1.4.1 Chapter 1

In this chapter, I include a problem statement on the reason why I decided to do this research; this is something that has affected me on a personal level, and so I wanted to survey students to see how many people had similar feelings, and how we can increase overall engagement in a classroom setting.

1.4.2 Chapter 2

In this chapter, I go over the research that I had done with some colleagues along with some results that were found in terms of games. Most of this is pretty fun stuff that I enjoyed doing with my team, and decided to include it in its own chapter just to drive home how engaging games can be, even when it comes to some difficult concepts.

1.4.3 Chapter 3

In this chapter, I go over some research that I did when it came to tile-based self assembly. This, in general, was a more difficult topic for me to digest whenever I first started learning it, and to this day remains an awe-inspiring topic. I tried to do some research, and I was able to implement some cool stuff, but was able to go nowhere in terms of theoretical results.

1.4.4 Chapter 4

In this chapter, I finally start talking about the problem that I decided to choose to turn into a game. The main goal was to choose a theoretical concept that would be new to most, if not all, students, and to turn this concept into a game. It wasn't the most easy task, but I managed to find out a way to game-ify the Pattern Self-Assembly Tile-Set Synthesis, or PATS, problem.

1.4.5 Chapter 5

In this chapter, I start to go into detail about the game/application that I started to build for the game version of the PATS problem. In here, I go into the specifics of what programming language framework I used, and how I ended building up the whole app.

1.4.6 Chapter 6

With the game being in usable condition, in this chapter, I record my results of my testing that I conducted, and finally concluding what I saw from these results.

CHAPTER II

GAMES BASED RESEARCH

Games based research has been ever present in the realm of theoretical computer science. There have been proofs about some Nintendo games being hard; proofs of PSPACE-Completeness of Sliding-Block Puzzles, similar to the occupancy problem that I go over in the next section. Games are super fun, and the proofs themselves are very conceptually interesting and fun to learn [7, 8, 19, 16, 1].

2.1 Occupancy

2.1.1 Information

This problem was worked on during a research hack-a-thon (HackR) that was conducted at The University of Texas at Rio Grande Valley during the Fall semester of 2019. **Collaborators:** Isabel Sanchez, and Francisco Gonzalez

2.1.2 Overview

With the occupancy problem, we investigated some simple variants of robot motion planning with global control; more specifically, we focused on line occupancy; checking whether a given configuration has a step sequence that will occupy positions of a $1 \times n$ line and square occupancy, which focuses on checking whether a given configuration has a step sequence that will occupy positions of an $n \times n$ square.

2.1.3 Introduction to the Problem

The tilt model, first proposed in [5], has foundations in classical motion planning. A couple of natural problems that arise in these computational systems are those of relocation and reconfiguration. *Relocation* is the problem of whether a sequence of tilts exists to relocate a tile from location a to location b. *Reconfiguration* asks if a sequence of tilts exists to transform board A to board B (specifying the location of all tiles). These were shown to be PSPACE-complete in 4-directions [9].

Here, we discuss a variant of this model introduced in [6] where particles exist within a board and move, in uniform, single unit distances (rather than maximally). This variant is referred to as the *single step* model. Figure 1 shows a simple example. The relocation and reconfiguration problem in this model were shown to be NP-Complete when limited to two perpendicular directions [10, 4].

2.1.4 Proposal

The goal is to investigate one of the simplest variants of the model. We obtain a simpler variant by constraining parameters of the problem input. The constraints to consider are the number of usable directions, and the complexity of the board. We will consider the problems in which *only* two perpendicular directions are usable, without loss of generality let these be East and South. There are many problems to consider, which are defined below. Despite the very simple nature of the problems, the computational complexity of the Reconfiguration and Relocation with these constraints have been open for over a year. The following occupancy variants have not yet been investigated and seem to be very approachable. Prove a complexity result for one or multiple of these problems. They are all known to be in NP (See Theorem 2). Can you reduce from a known NP-Complete problem to prove NP-completeness? Or on the other hand, can you show a polynomial time algorithm to also show membership in P? Note that for each problem, only one of these is possible unless $P = NP$

2.1.5 Problem Definitions

For the following problems, assume *B* is rectangular and that the available directions are South and East.

Line Occupancy: Given a configuration *C* = *(B,P)* where *P* consists of *n* single tiles, does there exist a step sequence to turn *C* into a configuration C' in which the tiles occupy all positions of a $1 \times n$ line. See Figure 2.1.

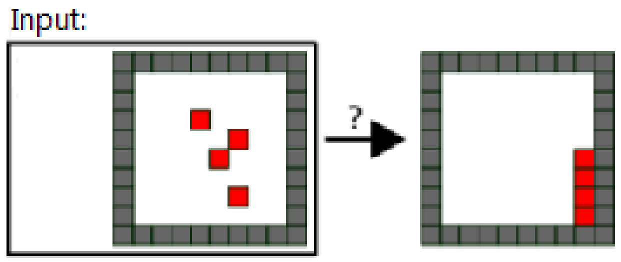


Figure 2.1: Example of Line Occupancy

Square Occupancy: Given a configuration *C* = *(B,P)* where *P* consists of n^2 single tiles, does there exist a step sequence to turn *C* into a configuration C' in which the tiles occupy all positions of an $n \times n$ square. See Figure 2.2.

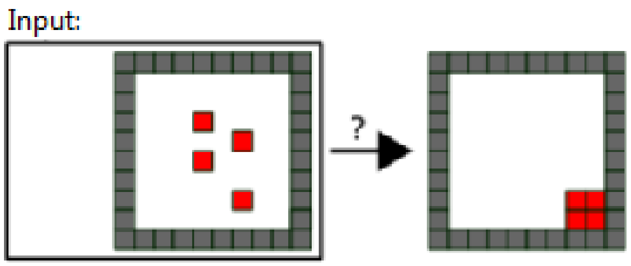


Figure 2.2: Example of Square Occupancy

2.1.6 Line Occupancy Results

Here we detail our results on line occupancy.

Lemma 1. *Given a configuration C, if* $\exists$ *a row with* $>$ *1 tiles, then C cannot be reconfigured into a configuration* C'*, where no rows with* $>$ *1 tiles exist.*

Proof. Through a series of moves, assume that a configuration C, which contains a row with more than 1 tile, can be transformed into configuration C', which does not contain any rows with more than 1 tile. To turn C into C', we would have to separate the 2 tiles that share the same row. To do so, we could attempt a down movement; however, since a down signal triggers all of our tiles to move down once in unison, the tile we wish to separate is now in another row with another tile, and so, there will always exist a row containing more than 1 tile. Since we know that configuration C cannot be transformed into configuration C' despite of the movements chosen, then our assumption is proved wrong via contradiction. □

Theorem 2. *Line Occupancy with 2 directions in a rectangular board is solvable in $\mathcal{O}(n)$ such that n is the number of tiles.*

Proof. Given an input encoding for a configuration C, map all of the tile's x and y coordinates. From here, only 2 cases can occur. In case 1, $\exists$ a row or multiple rows where there is > 1 tile. By following Lemma 1, if any tiles share the y coordinate, then we know that there is > 1 tile in that row, and we can return that the given configuration cannot be reconfigured into C'. In case 2, $\nexists$ a row where there is > 1 tile. If none of the tiles have a shared y coordinate, then we can return that the given configuration can be reconfigured into C'. □

2.1.7 Square Occupancy Results

Here we observe some things we found out about square occupancy.

Lemma 3. *Given a configuration C, if in any given row or column $\exists$ more than n tiles, then C cannot be reconfigured into a configuration C' where all of C' tiles occupy an $n \times n$ square.*

Proof. Direct from Lemma 1 □

Conjecture 1. *In a given configuration C, if $\exists$ both a row and a column that contain n tiles such that they do not share a tile, then we can show that the given configuration will never be solvable* *because eventually, we will make a move resulting in a row or column containing more than n tiles. An example of this can be seen in Figure 2.3.*

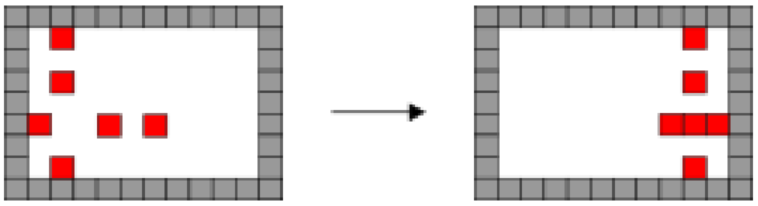


Figure 2.3: An example of a configuration C that contains n tiles on both a row and a column, but that do not share a tile. Eventually, we will make a move that makes a row or a column have more than n tiles

Observation 1. *We observed some board configurations in which we can easily verify that they can be reconfigured into C'. One case is when a configuration C has n tiles on n lines only. An example of such configuration can be seen in Figure 2.4.*

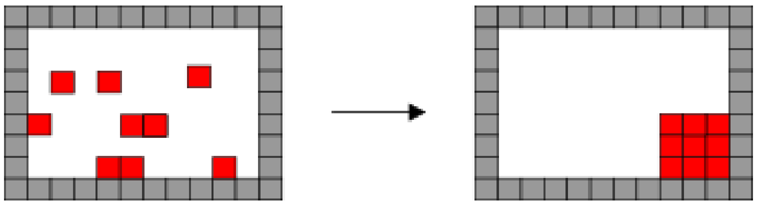


Figure 2.4: An example of a configuration C with n tiles on n rows that can be reconfigured to a configuration C' with an nxn square.

2.1.8 Results

We showed that Line Occupancy is solvable in polynomial time. We have provided an algorithm that is able to decide whether a given configuration C is able to be reconfigured to C' in $\mathscr{O}(n)$, where n is the number of tiles. We have also explored some findings dealing with the Square Occupancy problem, though we were not able to prove $\mathscr{P}$ membership.

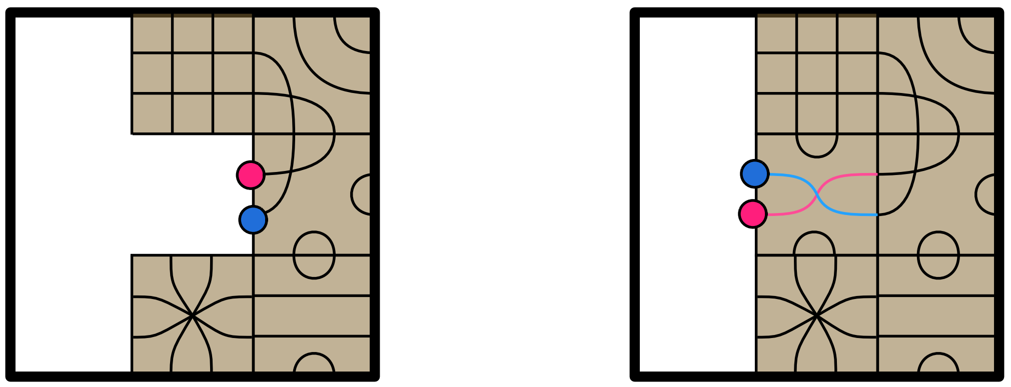
2.2 Tsuro

2.2.1 Information

This game problem was worked on during a Games, Puzzles, and Computation course given at The University of Texas at Rio Grande Valley during the Fall semester of 2019. **Collaborators:** Timothy Gomez, David Caballero, Isabel Sanchez, and Francisco Gonzalez.

2.2.2 Overview

Tsuro is a tile-based board game for two to eight players. Each player initially places a token on the beginning of a path which are located along the edges of the board. Throughout the game, players place a tile in front of their token. These tiles consist of paths in which your token follows after placement. See Figure 2.5. The goal of the game is to remain the last one standing. A player loses when the path they are on leads to either run into another player or fall off the board. As the board fills up, the game becomes harder because fewer empty spaces exist. Furthermore, another player's tile may lead your token in a path you'd rather not go.



(a) (b)

Figure 2.5: A board configuration before (a) and after (b) tile placement. Each token follows its respective path until it reaches an empty position or goes off the board.

In total there exists 36 different tile types. This includes every possible combinations of paths between 8 different points on the tile. In the normal version of Tsuro, each player has 3 cards in their hand, and they draw a new tile every time a tile is placed. In the combinatorial version, all tiles are available for placement from the beginning. Normally, the game allows for up to 8 players, in our investigations, we focused on the 2 player version of the game.

2.2.3 Optimal Strategies

At the start of the game the first player places his token first. If the first player places his token closer to the corner then it is possible for the second player to block him off from the rest of the board and limit his moves. Placing in the middle allows for locations with a higher value.

The second player wants to place on the ’outside’ of the first player, meaning the second player wants to place their token adjacent to the first player on the side with more available spaces. See Figure 2.6. This will allow them to begin to wall off locations and limit the moves that first player can make.

In the version of Tsuro where all pieces are always available good pieces to pick at the beginning allow the player to move straight forward while blocking the location on the side of the opponent. Pieces that aren’t great to select at the beginning include the piece that is only straight lines. This piece allows for your opponent to jump on the other side of you.

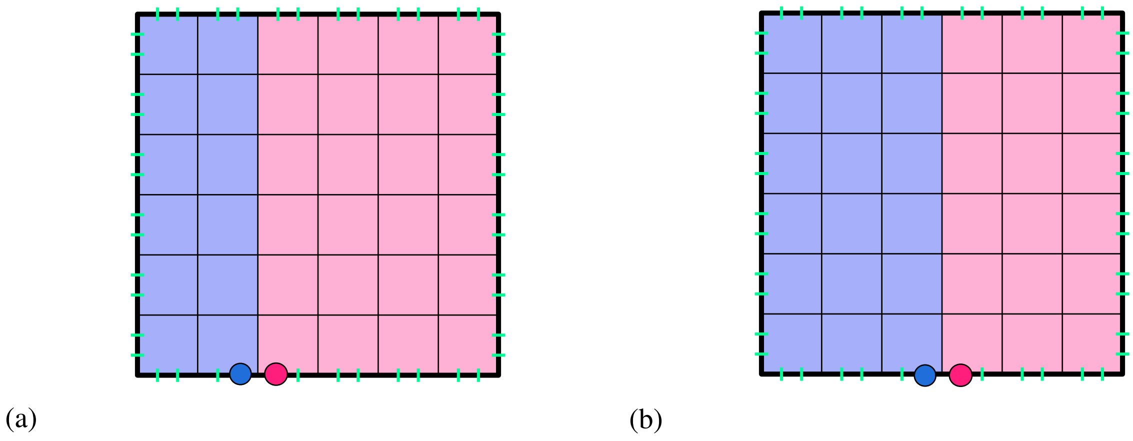

(a) (b)

Figure 2.6: Example opening placements, pink as second player

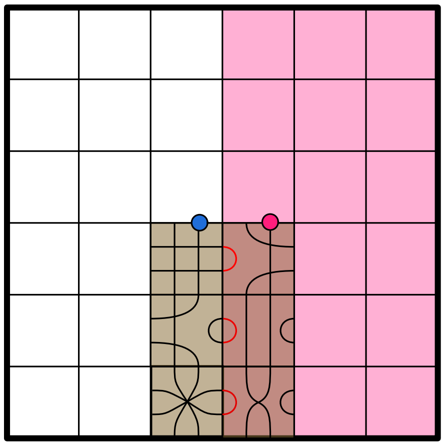


Figure 2.7: Wall-off strategy, Pink as second player

2.2.4 Second Player

The combinatorial version of Tsuro is a second player win. The first player can place his stone anywhere on the edge of the board. No matter where they places it the second player will place their stone adjacent to the opponent on the side with more available spaces, as mentioned earlier.

The second player will want to wall off the first player by placing tiles that do not allow the first player to cross the board. If the second player builds a wall all the way to the end of the board and then turns both players will have an equal amount of spaces to play. If both players have an equal amount of space and each move takes up a space, first player will run out of moves one turn before second player does. This strategy can be see in Figure 2.7 where second player's area can be seen in pink.

First player does not have a counter to this strategy. There are enough tiles the second player can use such they can always wall off the first player. First player must continue straight. If they try to turn away from second player the wall will still be built. If they turn into the second player they are leaving themselves open to be forced off the board and losing.

2.2.5 End Game Positions

Type P Tsuro endgames having type P would mean that the next player to place a tile would leave the board and lose. When there is only a move left, this is the most common endgame type. See Figure 2.8

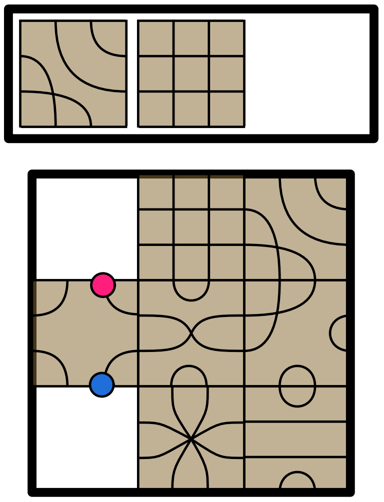


Figure 2.8: Type P Tsuro Endgame

Type N Tsuro endgames having type N would mean that the next player to place a tile would be able to move the opponent off the board. This is endgame type is usually avoidable as it requires ending up on the same tile as your opponent, which an optimal strategy would avoid.

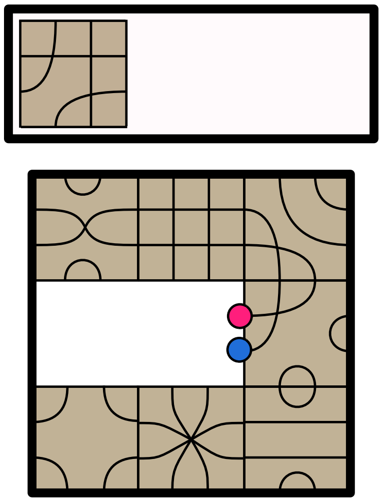
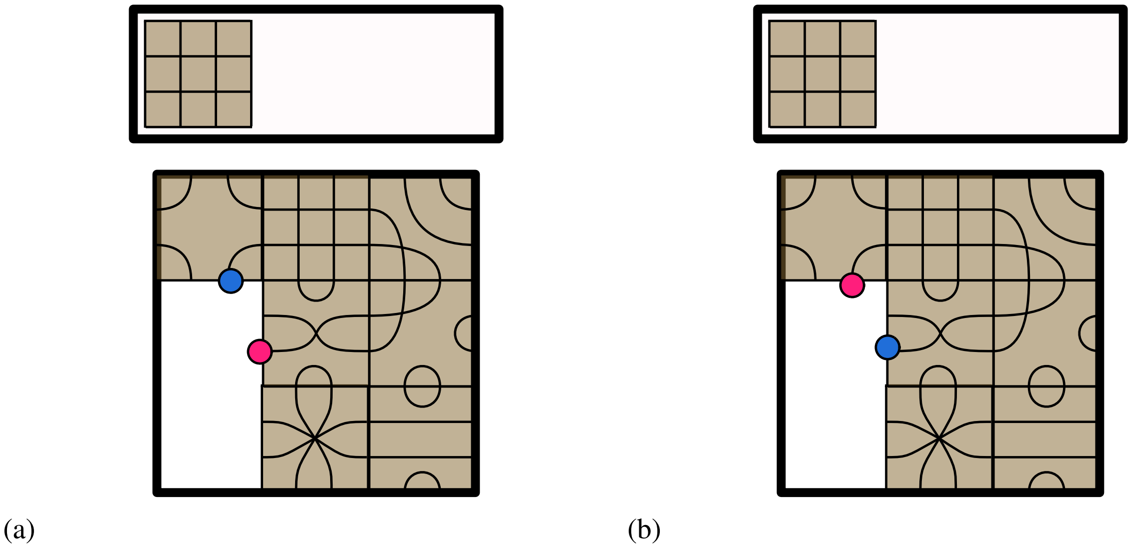


Figure 2.9: Type N Tsuro Endgame. Either player can send themselves straight while turning their opponent left/right and sending them off the board

Type L/R Tsuro endgames having type L/R would mean that the no matter who plays next the specified player will win. This is also not a very common endgame type when played normally. But with optimal strategy the game does start as type L.



(a) (b)

Figure 2.10: (a) Type R Endgame, placing the only tile left will always send L (pink) off the board. (b) Type L Endgame, placing the only tile left will always send R (blue) off the board.

2.2.6 PSPACE-Completeness

We show that generalized combinatorial Tsuro is PSPACE-Complete.

Tsuro Decision Problem: Given an instance of a generalized combinatorial Tsuro game T, can Player 1 force a win from T?

We are reducing from the Geography game, planar max degree-3. Proved to be PSPACE-Complete in [15]. To do this assume we are given a planar max degree-3 graph G. For each vertex in G we create a vertex gadget, see Figure 2.11. Each vertex gadget has four open spaces.. Space A is used to traverse the gadget normally. The tiles in Figure 2.12 are the only two tiles available for placement, and we assume there are sufficient counts of each. When the pieces are in the positions shown in Figure 2.11, the player whose turn it is can chose to move across one of the other two edges, by choosing a tile that lets them turn, or choosing the tile that lets them go straight.

Once an edge is traversed the pieces cannot move back along that path. If a player makes a move that sends the pieces to a vertex that already has a tile in space A, the pieces will end up in space B. The player whose turn it is now does not have any legal edges to traverse and in the geography game would lose. The way that is enforced in Tsuro is the rule stating that players may not make moves to force themselves to lose. The only option the player has is to place the tile that will send both pieces to space C. From here the winning player can then send the other players piece upward off the board and their piece will be safe is space D.

2.2.7 Proof using the gadgets

Lemma 4. *Generalized Combinatorial Tsuro is in PSPACE.*

Proof. This shown with a polynomial space algorithm that decides the problem. Given an instance of generalized combinatorial tsuro T, consider the game tree of possible moves from configuration T. We will recursively check every child node of T, solving each subproblem and returning a 1 or 0 depending if that node is satisfied or not. A node being satisfied depends if it is a universal node (2nd players turn) or existential node (1st players turn). Since we are solving every branch

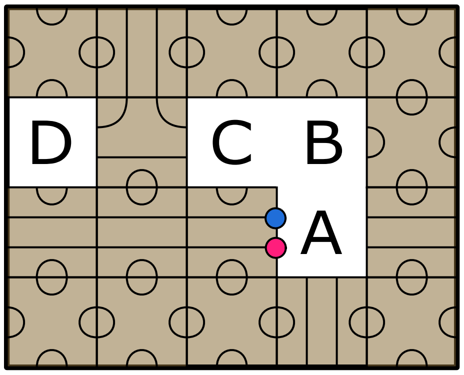


Figure 2.11: Vertex Gadget

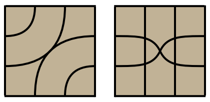


Figure 2.12: Tiles Available in Reduction

individually the amount of space used is polynomial to the depth of the game tree. Since this game is bounded we know that the depth has a polynomial bound. □

Theorem 5. *Generalized Combinatorial Tsuro is PSPACE-Complete*

Proof. The rules of the game ensure that no edge is used more than once. The gadgets force a player who has no available moves in the geography game to lose in Tsuro. If Player 1 has a forced win in the geography game, they can select tiles that represent edges to force Player 2 to a vertex that has no available edges. Then Player 2 will be forced to place tiles that will result in them being forced off the board. If Player 1 has a forced win in Tsuro, then there exists a movement through the gadgets that leaves Player 2 to force themselves off the board. The only way Player 2 will force themselves off the board if they end up in a gadget with no other available moves which means Player can make selections in the geography so Player 2 has no edges to select. This shows that generalized combinatorial Tsuro is PSPACE-Hard. In combination with Lemma 4, we show PSPACE-Completeness. □

2.2.8 Overview of Software

To allow for the game to be played digitally we created a simulator in Python. The simulator runs on Python 3.6, and uses the tkinter GUI library, as well as the PIL library for image modification. The simulator first asks for Player 1 and Player 2 to select their starting locations. Then Player 1 and Player 2 can alternate selecting tiles from the available tile pool on the right. The tiles on the right can be rotated with right click, and selected with left click.

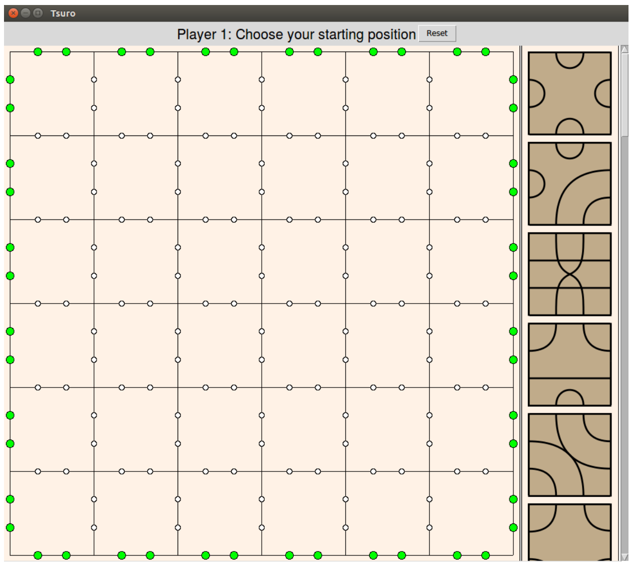


Figure 2.13: Image of simulator starting state

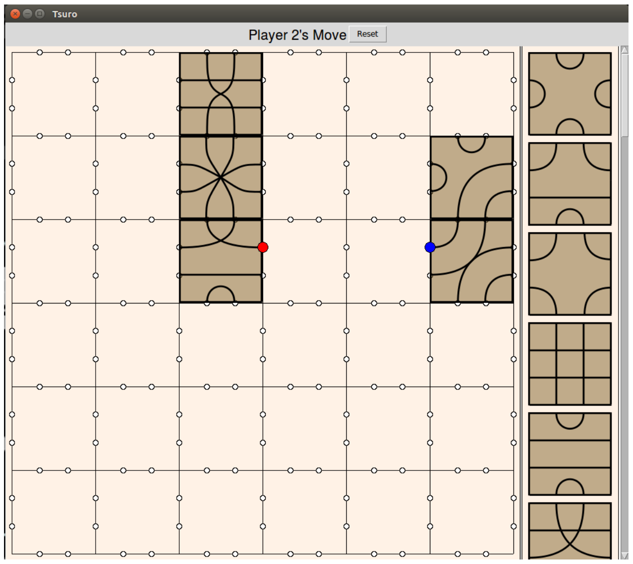


Figure 2.14: Image of simulator in use

2.2.9 Results

With this, we were able to detail what the game Tsuro is; how it's played, along with some optimal strategies. We were able to show that the game is PSPACE-Complete, and along-side that result, we showed how the game is played via a programmed Python game.

2.2.10 Difficulty

Theoretical research does not come easy to me like it does to my colleagues, but here is a really cool topic that I was able to get invested in, and I was able to solve a couple of cool problems with my team. I found these topics to be a bit easier to comprehend in comparison to other difficult theoretical concepts.

CHAPTER III

TILE-BASED SELF-ASSEMBLY

3.1 Theory

I joined the Algorithmic Self Assembly Research Group (ASARG) back in 2019 and since then, I’ve worked on a couple of things with them. The main research topic that the group works on in self-assembly.

3.1.1 Self-Assembly

Self-assembly is a natural process by where a monomer can autonomously come together to form complex shapes/structures. It was first studied by Erik Winfree in his paper "Algorithmic Self-Assembly of DNA" [31] that poses whether these molecules can be used for computational purposes. Based on Wang tiling [29], the Tile Assembly Model gives way to an abstraction of crystal growth through the two dimensional self-assembly of square units that are referred to as "tiles." The area of self-assembly is very large and extensive research has been done to take advantage of the self-assembly model, like, using it to create new methods of microscopic medicine delivery and/or the creation of nanotechnology [3, 30, 2, 22, 21].

3.2 Tile Automata

The Tile Automata model was first seen in [12], and this model was largely inspired by the nubots model in [32], with one small caveat. This is in regards to a limitation to a movement rule that is important in the nubots model itself. A Tile Automata system is a combination of cellular automata and 2-Handed self-assembly; a system that consists of an alphabet of state types, a set of transition rules, a set of initial assemblies that are assigned states from the alphabet, the affinity functions, and the stability threshold. An example, taken from [12], is shown in Fig 2.1.

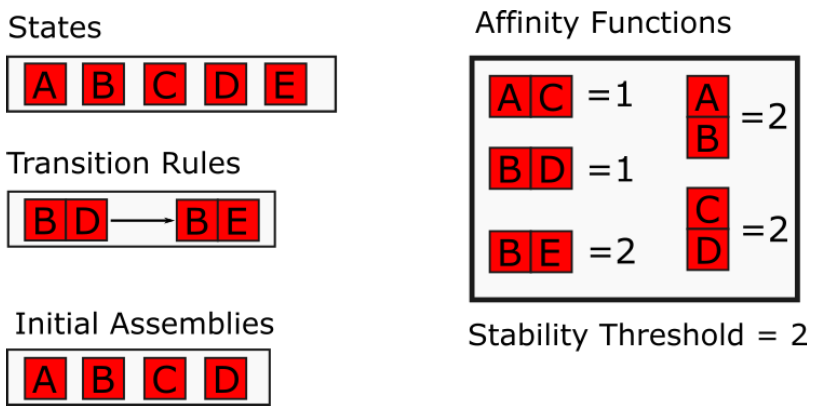


Figure 3.1: Tile Automata System

Assemblies in the model are created from the set of initial assemblies, like the one shown in Fig. 2.1, and combining previously built assemblies given if they're able to bind together through the affinity functions. Currently existing assemblies have the possibility of changing states if any of the tiles inside are able to change states as per the given transition rules.
In Fig. 2.2, also taken from [12], shows the set of producibles, what's able to be made through combinations, and the set of terminals, the combinations that are set in stone and can no longer change no matter what is done, from Fig. 2.1.

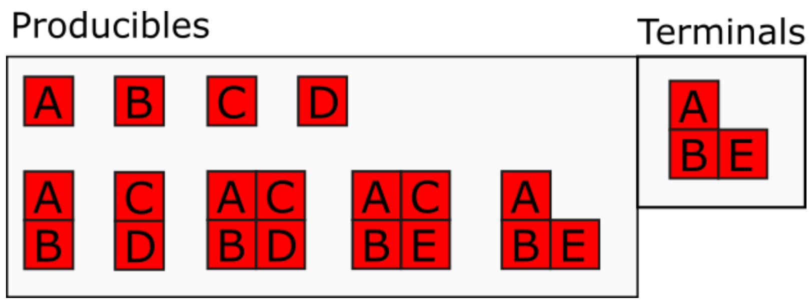


Figure 3.2: Producibles and Terminals of Fig. 2.1

There's work that has been done that covers various self-assembly models, which includes models like 2HAM (2 Handed Assembly Model), which is one of the simplest and most studied models, the staged model, which is more of a generalization of 2HAM, and the signal tile model, which is a model of self-assembly that considers monomers that are semi-intelligent [28, 11, 18, 17, 14, 13].

Keeping this model in mind, most of the work that I have done with the research group has revolved around the Tile Automata model and variations of said model. For example, one of my early contributions to the group was on a variation of the Tile Automata model called Discrete Count Tile Automata.

3.3 Discrete Count Tile Simulator

3.3.1 Information

This problem was worked on for regular research outside of class; taken on as a project to see if my colleague and I could work out a result. **Collaborators:** Mason Garza, and Francisco Gonzalez.

3.3.2 The Question

The topic of research that I was looking into was on the tile based self assembly problem. In our case, this is our example scenario; we have 3 main tile types: S, A, and T. Every tile type has different glue configurations, in which these tiles can stick to each other. S has a glue "a" on its east side. A has a glue "a" on its west, and east sides. T has a glue "a" on its west side. What this means is that these tiles can stick together to form "ST" lines. E.g. [S,T], [S,A,T]. An example of this is shown below:

With this in mind, we had this question to ask, given a set of these tile types with certain counts for each tile type, what is the expected length of these ST lines, and what is the percentage of ST lines of that length? That's the question we aimed to solve, and so, a colleague of mine and I got to work on a Python project to calculate this. We had chosen Python to work on this project due to the data science tools that the Python programming language provides and the general ease of making a basic front-end interface with it.

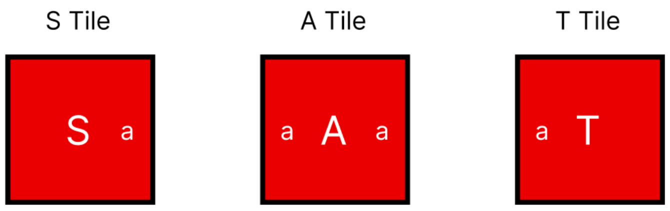


Figure 3.3: S, A, and T Tiles

3.3.3 The Python Program

The interface of the program didn't have to be all that complex since we only needed four inputs:

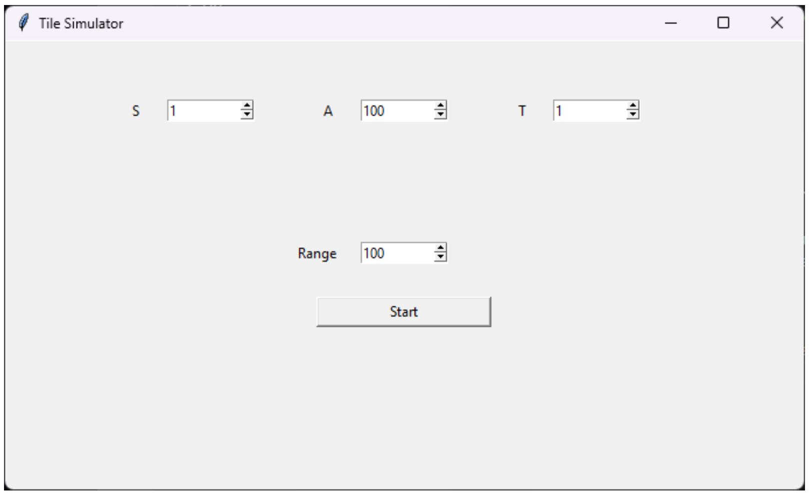
1. The number of S tiles
2. The number of A tiles
3. The number of T tiles
4. The number of consecutive tests

With these inputs in mind, we designed a very basic interface with those four inputs, as shown below:

Once the end user chose the appropriate numbers, they could press start and the program would begin to calculate the expected length of these ST lines, and the percentage of ST lines of that length. Once the program crunched the numbers, it displays the information calculated on a bar graph, as shown below:

3.3.4 Results

The results from our continuous testing with the program brought forth major confusion. This is mainly because we originally thought that our simulation output would resemble a normal



S 1 A 100 T 1

Range 100

Figure 3.4: Simulator Interface

distribution curve. Instead, there was a huge bias towards building the longest possible line, and this bias is showcased in Figure 5.2. Considering Algorithm 1, my colleagues and I determined that there were no major flaws with the algorithm, and many tests were made with the same biased result. To truly prove this, we'd have to manually and mathematically prove this to be right, but as of the writing of this document, we have not pursued this.

3.3.5 Difficulty

Due to the mathematical complexity of these problems; I have general difficulty in coming up with proofs completely, and hence why, for me, and I'm sure for countless others, games can be a fun way to interact with difficult problems. With this in mind, I decided to choose a good problem that can be turned into a game, and that's when I found the PATS problem.

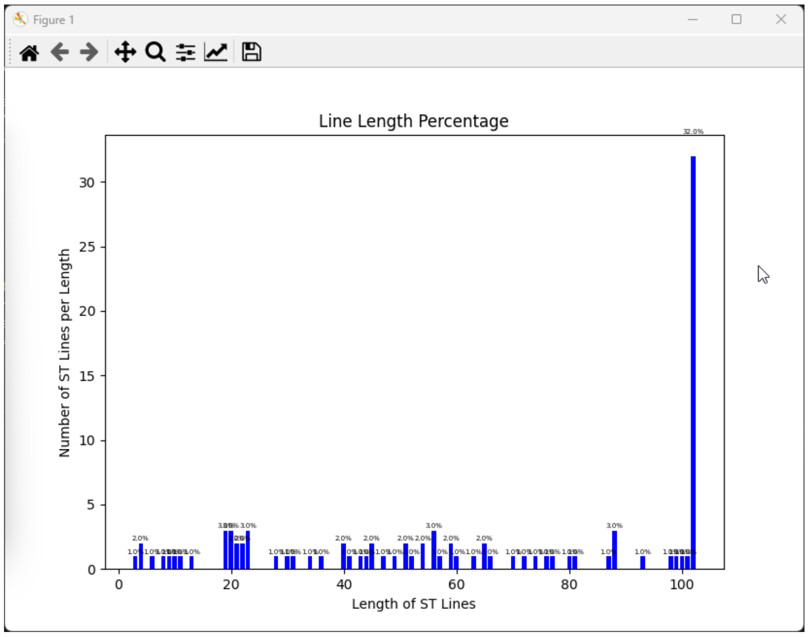


Figure 3.5: Simulator Output

Algorithm 1 Algorithm for building SAT lines

```
$X$ = list of S, A, and T tiles
$Y$ = list of ST lines
$a$ = number of S tiles
$b$ = number of T tiles
while $a \neq 0$ and $b \neq 0$ do                         ▷ while S and T counter are not 0
    $x \leftarrow [s,t]$ from $X$          ▷ sample two random assemblies from X and put into x
    if assemblies in $x$ can be glued together then
        glue them together
        remove individual tiles from $X$
        place glued assembly back into $X$
        if glued assembly becomes an ST line then
            $Y \leftarrow x$
            decrement $a$ and $b$
        end if
    end if
end while
```

CHAPTER IV

THE PATS PROBLEM

4.1 Overview of the Problem

The Pattern Self-Assembly Tile-Set Synthesis, or PATS for short, problem was first introduced in [23, 24], and was placed inside of the Tile Assembly Model (TAM) by Winfree in [26]; the overall concept of the problem is not too complex. Here's what the PATS problem asks; given a *k*-colored pattern, can we make an optimally small tile set and an L-shaped seed to self-assemble the pattern?

Algorithm 2 The PATS Problem

```
Input:
  $X$                                         ▷ k-colored pattern
  $Y$                                         ▷ l-shaped seed
  $Z$                                         ▷ user-defined tile set
Output:
  $A$         ▷ k-colored pattern made from l-shaped seed and tile set
```

4.1.1 Attachment

The implication of this problem is that from the user-defined tile set, selected tiles must be able to attach to the L-shaped seed; with the aim of building the given *k*-colored pattern, but how do these tiles even attach? Glues are the answer here.

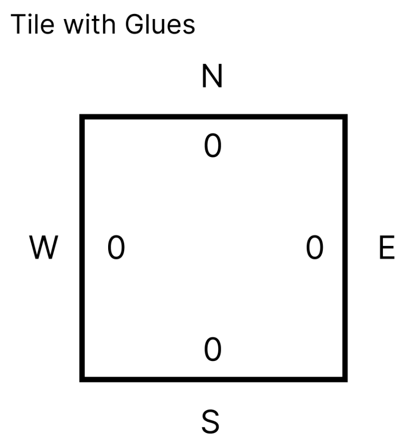


Figure 4.1: Simple tile with glues

Looking at Figure 4.1, it showcases a simple tile with glues. We have a north, east, south, and west side of the tile that can be populated with a numerical value that indicates its "glue". In order for a tile to attach to another tile, the tile wanting to attach must have two sides that satisfies its glues. As an example, look at Figure 4.2

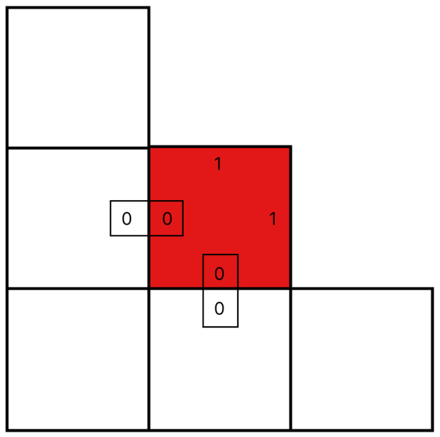


Figure 4.2: A tile attaching to an L-shaped assembly

In the case of Figure 4.2, we can see that the south and west side of the red tile's glues are being satisfied by the L-shaped assembly, and since this is the case, the tile successfully attaches to the assembly. Knowing this, we can lay down the algorithm for building a pattern.

4.1.2 Building an Example

Given an L-shaped assembly and a tile set:

Algorithm 3 Algorithm for building an assembly

```
Input:
  $X$                                                        ▷ l-shaped seed
  $Y$                                                        ▷ user-defined tile set
  $Z$                                                        ▷ k-colored pattern
  for $x = 0$ to $x <$ length of $Y$ do
      grab corresponding tile in $Y$
      check if it attaches to $X$
      if it doesn't attach to $X$, move to the next tile
      if it does attach, attach the tile to $X$
  end for
  verify that the assembly in $X$ matches the given pattern in $Z$
```

Using Algorithm 3, we can try to build a pattern using a trivial example; we're going to be using the pattern shown in Figure 4.3.

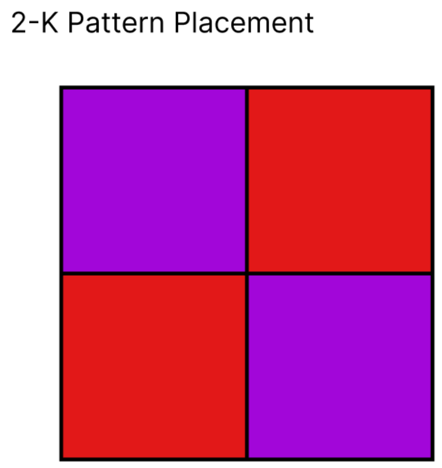
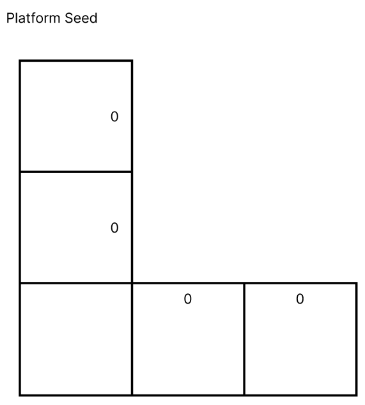
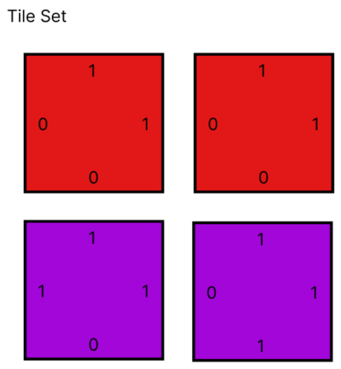


Figure 4.3: Example pattern

Additionally, since we were given Figure 4.3, we're going to generate our own seed assembly and our own trivial tile set to build the pattern in that figure.



(a)



(b)

Figure 4.4: A seed assembly (a), and a tile set to build an assembly (b)

If we follow Algorithm 3, then we are provided with the following finished assembly.

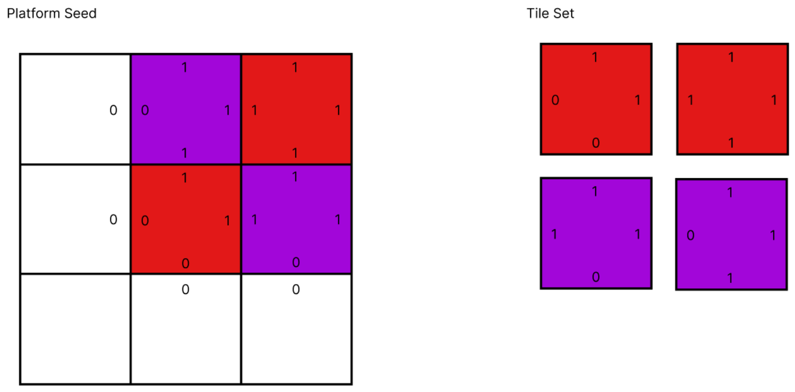


Figure 4.5: Finished assembly

Taking a look at Figure 4.5, you can make out all of the places where the tiles are appropriately attached, and one can see that all the glues match one another.

4.1.3 The Perfect Problem

The PATS problem stuck the most out to me since it's not too difficult of a concept, and turning it into a game was actually not too difficult. With the goal to teach students a new theoretical concept with a game, this problem was a perfect candidate to test out my theory.

CHAPTER V

THE GAME

Having chosen the problem to convert to a game; I now needed to figure out the logistics towards building this game, and what I needed to use in order to realize this vision. As such; I got to work with figuring out what to use.

5.1 User Story

Before getting into building a whole program/application to make a game, I needed to come up with what an end-user will actually be doing in the game. This step in the process heavily helps with figuring out what needs to be implemented, and it helps mitigate unused code.

Table 5.1: Main User Story

Feature	**Accessible by User**	**Accessible by Admin**
Must be able to see home screen	Yes	Yes
Must be able to access and see any added pattern	Yes	Yes
Must be able to add a solution to any added pattern	Yes	Yes
Must be able to add a new pattern to the home screen	No	Yes
Must be able to remove patterns from the home screen	No	Yes
Must be able to be added to the leaderboard	Yes	Yes

The features listed in 5.1 are the "core" features of the main application that needed to get done. With this in mind, and with the preliminary things out of the way, I needed to find what I was going to use to build up the application.

5.2 Tech Stack

Having had experience with a wide variety of programming languages and frameworks, this is what I came up with.

1. Dart/Flutter Framework (For building up the game and interface)
2. Firebase (Amazon owned nosql database that works great the Flutter framework)
3. Firebase Hosting (For hosting the actual game for anyone to access)

5.2.1 Flutter

Flutter is a framework that was first released in May of 2017, with Dart, a programming language created by Google, used as the underlying programming language.

Figure 5.1: Flutter banner

Its main goal is to have one singular code-base that can natively compile to all sorts of platforms; from

1. Android
2. iOS
3. MacOS

4. Windows

5. Web

When it was first originally released, it only supported native compilation to the Android/iOS operating systems, but since its release, it has added everything else to the platforms that it can natively compile to. As such, the hassle of having to deal with multiple code-bases for multiple platforms becomes a lot more simple. This was the main reason why I chose Flutter as the main framework to build the game with. Along with its extremely customizable UI; the underlying Dart language is very versatile and easy to write logic for.

5.2.2 Firebase

Having chosen the framework to build up the main application, now what was needed was a database to handle accepting incoming patterns to add to the home screen and to handle new incoming solution submissions for each of the patterns present in the game.

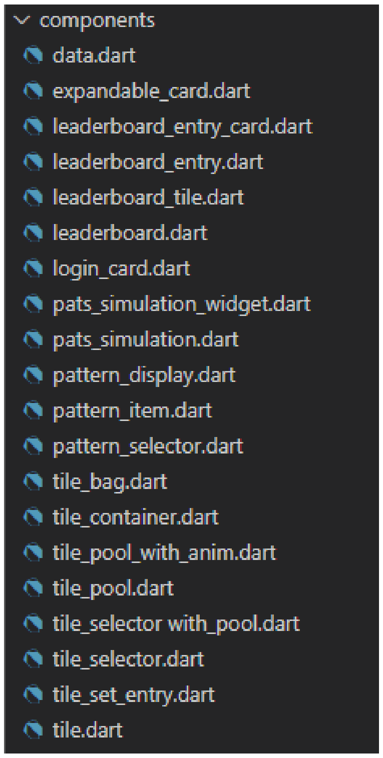
Figure 5.2: Firebase banner

Firebase was a really good choice to handle this small-scale application as it provides a lot of functionality alongside the Flutter platform; having a readily available library to access all of that functionality directly inside of Flutter at no cost. Additionally, Firebase includes the ability to host the application directly within its framework, and in this case, it'd be able to host the game in its website form at no cost as well. Having both the framework to build the application and the means to store and retrieve the necessary data, it was now time to build the app!

5.3 Implementation

5.3.1 Components

Having had plenty of software experience, I employed many strategies and common practices to facilitate building the application in a scalable manner. Since the game was going to be built from scratch, I started to build all of the separate components that I was going to use for the UI into their own separate classes. In the end, I ended up custom writing twenty different components that are all used within the main application; you can see these components in Figure 5.3.



```
components
  data.dart
  expandable_card.dart
  leaderboard_entry_card.dart
  leaderboard_entry.dart
  leaderboard_tile.dart
  leaderboard.dart
  login_card.dart
  pats_simulation_widget.dart
  pats_simulation.dart
  pattern_display.dart
  pattern_item.dart
  pattern_selector.dart
  tile_bag.dart
  tile_container.dart
  tile_pool_with_anim.dart
  tile_pool.dart
  tile_selector with_pool.dart
  tile_selector.dart
  tile_set_entry.dart
  tile.dart
```

Figure 5.3: Final Components List

I made these components to be very easy to place and remove wherever they might be needed, and this method of building them helped me out greatly for when I started to build the main pages for the application.

5.3.2 Main Pages

There are four main pages within the application:

1. Main Home Page
2. Main Login Page
3. Page To Add A Pattern
4. Page to View A Pattern

Three of the four pages are essential to the game and are finished, and the only unfinished page is the login page due to time constraints. These pages are pretty self explanatory, but I shall go over them to detail the goal of each page.

5.3.3 Main Home Page

I wanted to create a home page that was easy to navigate and that was able to display all of the new and available patterns that were in the database. As such; I adopted a grid to place all of the patterns and designed a component to display such pattern.

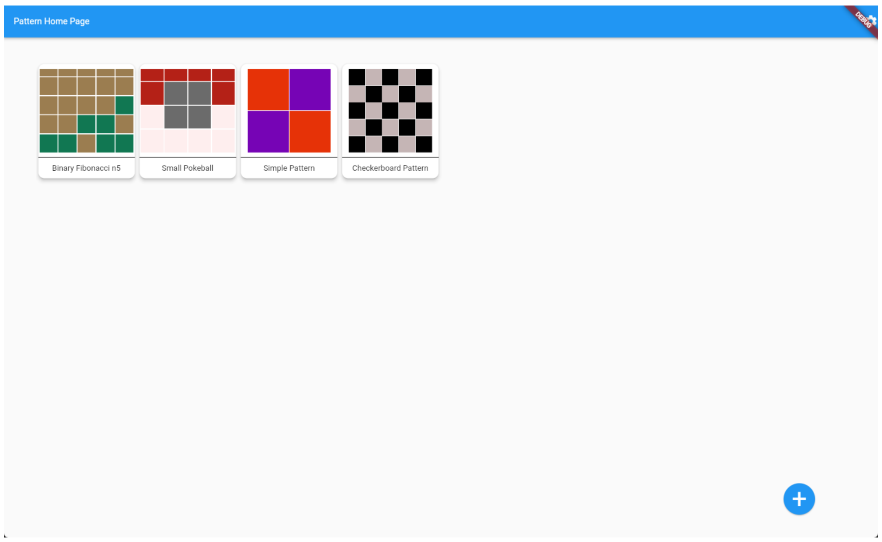


Figure 5.4: Application Home Page

The home page visible in Figure 5.4 showcases the design I was going for. The main space is taken up by where the patterns would be visible, and if you have admin rights, then the blue button on the bottom right would be present for you to click and be taken to the "Add Pattern" page. All of the patterns are clickable and if you click on them, then they take you to the "View Pattern" page.

5.3.4 Add Pattern Page

When you click on the blue button present in Figure 5.4, then you are routed to the "Add Pattern" page. The purpose of this page was simply to be able to add a new pattern.

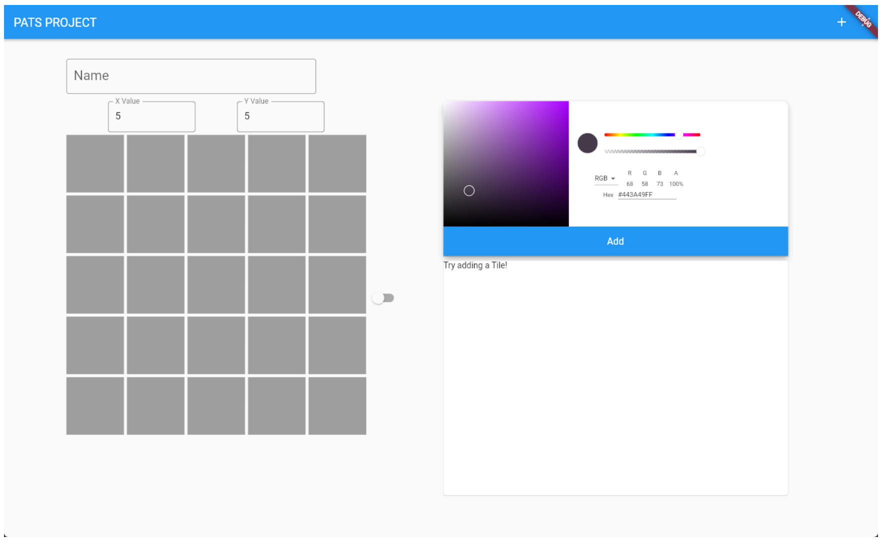
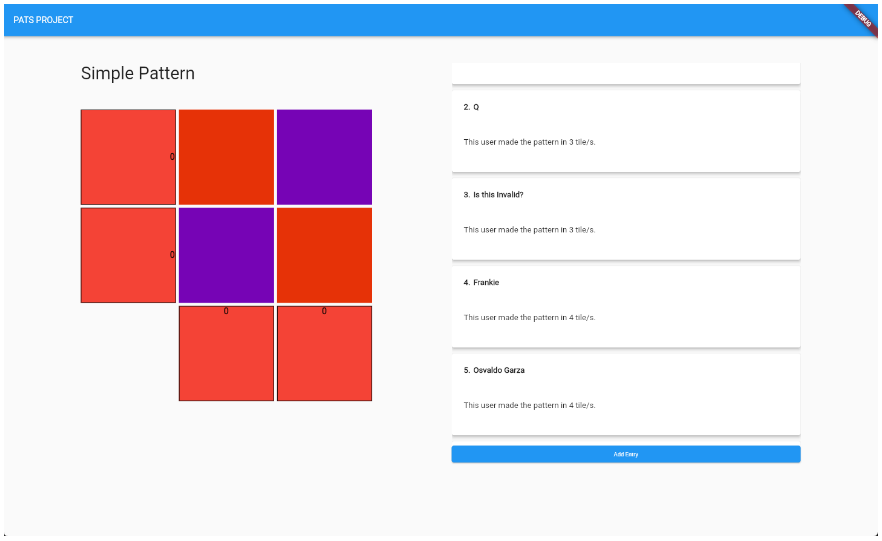


Figure 5.5: Add Pattern Page

In the design of the "Add Pattern" page seen in Figure 5.5, I added a main grid that its dimensions can be changed by the two text boxes above it. On the right side of the screen is a color picker where you can choose the color you want to paint with, and below it is a tile pool where it tracks what colors you have used. You can interact with the grid by individually clicking on the tiles, while having a color you picked selected, and the color will change from the default grey to the selected color. Once you are happy with the pattern on the grid, you can click on the "+" on the top right of the screen, and the data will be passed along to the database. Once it has successfully passed the data, it will boot you back to the home screen and you will be able to see the newly added pattern.

5.3.5 View Pattern Page

The "View Pattern" page is the most substantial page; as it shows the current pattern that you clicked on, and it has a leaderboard included that shows you who has attempted to solve the selected pattern and with how many tiles they attempted with. Additionally, it allows you to add an entry of your own in the same page.



2. Q

This user made the pattern in 3 tile/s.

3. Is this Invalid?

This user made the pattern in 3 tile/s.

4. Frankie

This user made the pattern in 4 tile/s.

5. Osvaldo Garza

This user made the pattern in 4 tile/s.

Figure 5.6: View Pattern Page

As you can see in Figure 5.6; you can see the main pattern that was selected, along with the leaderboard on the right side of the screen, with an "Add Entry" button at the bottom of the leaderboard. Upon clicking the button, the leaderboard component transitions to the component to add a new tile set to solve the pattern.

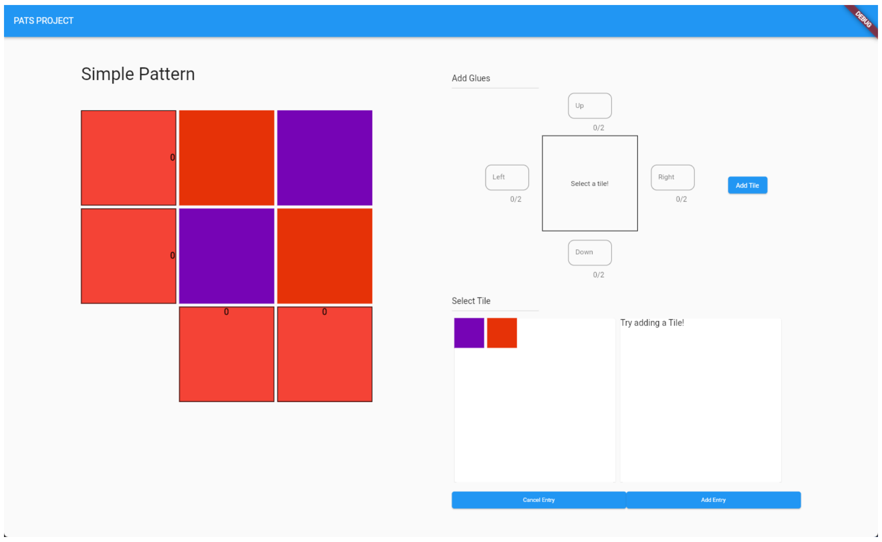


Figure 5.7: View Pattern Page (Adding a Pattern)

In Figure 5.7, you can see the leaderboard was replaced with the component to add a tile set to solve the pattern. You can select any color of the pattern, and you can add glues on either four sides of the tile. Once you are happy with the glues, you can click on the "Add Tile" button to the right, and it will be added to the tile pool located on the bottom right. Once you are happy with the tile set that is inside of the tile pool, you can go ahead and attempt to run the simulation to see if it is a verifiable solution.

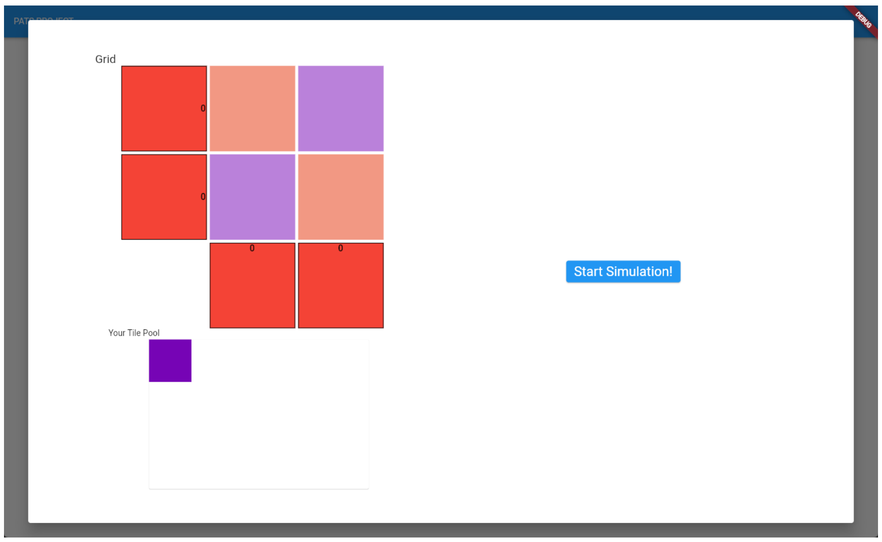


Figure 5.8: View Pattern Page (Try Solving Pattern)

Shown in Figure 5.8, this is the main component that does the simulation aspect of the application. It shows you the pattern that you're trying to build, greyed out, and it shows you the tile set you provided below in the tile pool. Once you confirm that the tile set is correct, to the right, you can click on the "Start Simulation" button to begin the simulation. Upon clicking the button, the program will be using Algorithm 3 to try and build the pattern.

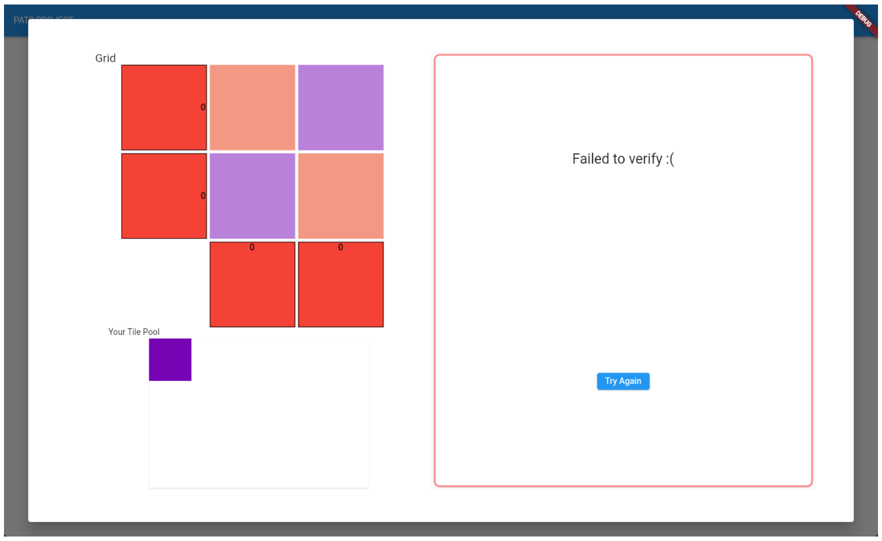


Figure 5.9: View Pattern Page (Failed Attempt)

Whenever you provide a tile set that fails to build the pattern, the game will tell you that it failed to verify the pattern, shown in Figure 5.9, and wherever the pattern building stops is what's going to be shown in the grid. In the case of the figure, no tile was able to be attached, and so the entire grid remains greyed out.

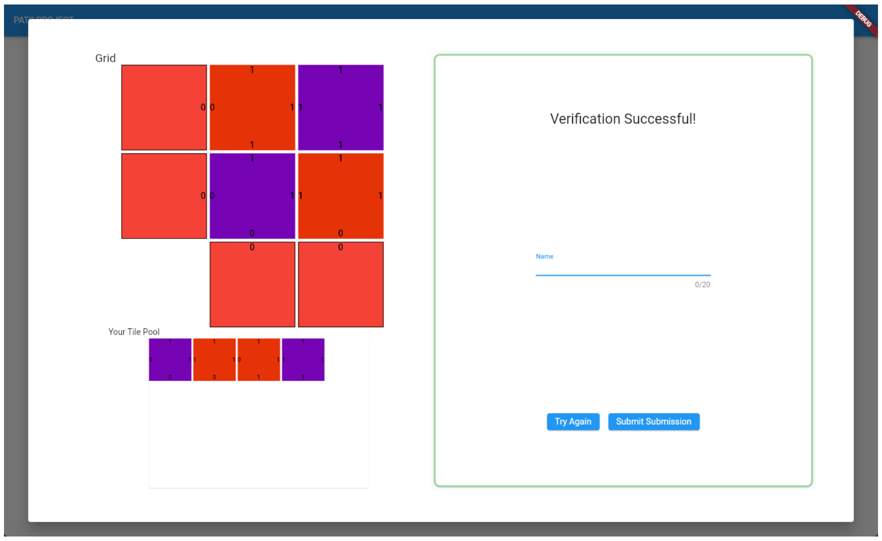


Figure 5.10: View Pattern Page (Passed Attempt)

In the case of Figure 5.10, whenever you provide a tile set that successfully builds the pattern, then the game will tell you that it was able to verify the pattern, and will allow you to input a name and a button to submit your submission. Additionally, it shows you the completed built pattern, and you can see where all the glues match each other. If you think you can do better, you can immediately try again, but the option to submit is there as well.

5.4 Challenges

5.4.1 State

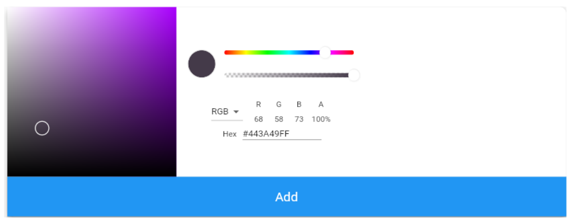
Building this application from the ground up was quite the undertaking for a project I only had a semester to complete and test. One of the biggest challenges was keeping up with all of the application state, and being able to manipulate the state from all the different pages, and so learning how to create hooks was one of the best learning experiences for me, as it facilitated, for example, calling a function to add a pattern to the database all the way from a completely different component class that didn't have this function.

5.4.2 UI Design

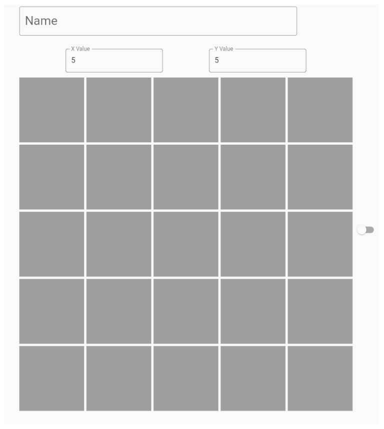
Usually, whenever a new project is being undertaken, a prototype application is made using software like "Figma" or "Adobe XD"; iterating through page designs until you are satisfied with the design of the project. Considering that I was doing all of this myself, and considering the time limit that I had, I felt like I couldn't afford to spend time prototyping the application, and so I did everything as I went; relying on my eye alone to figure out whether or not something looked off.

5.4.3 Component Design

Since all of the components are custom made, I had to make sure that all the components were able to talk to each other effectively, without doing some insane stuff. From the get go, I had to have this mentality in place; taking into account what components need to talk to each other, and where the components are going to be placed. Here are some visual examples of the components that I designed:

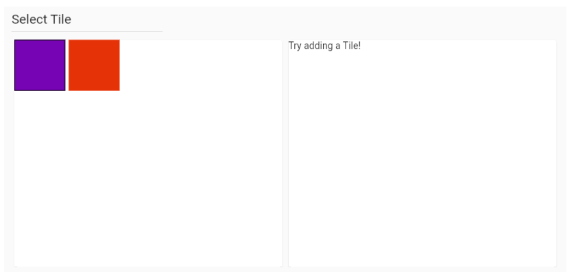


(a)

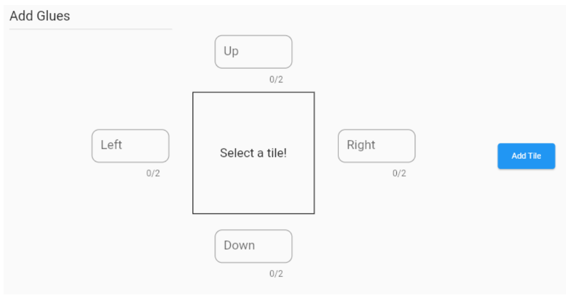


(b)

Figure 5.11: a) Color Selection Component and b) Tile Placement Grid Component



(a)



(b)

Figure 5.12: a) Final Tile Selection Component and b) Tile Glue Component

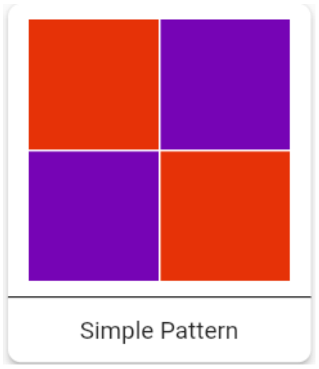


Figure 5.13: Home Page Item Component

5.5 Finished Build

The finished build of the game is the culmination of all of the figures in this chapter. It was a long and challenging task to build the game, but I had extreme passion in building it, and I'm proud of the way it came out. I hope to buff out any bugs, and in its current form, it is absolutely usable.

CHAPTER VI

TESTING

With the main application mostly done and working, I now needed to test its effectiveness, and so I devised my plan to have two separate groups that I'd be presenting to.

6.1 Engagement Hypothesis

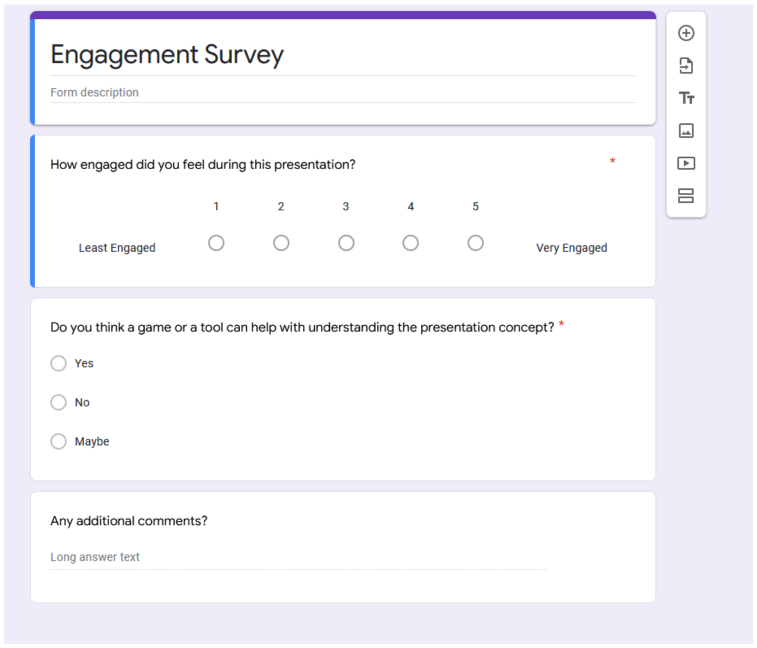
Given the option to teach a new concept with purely book + slideshow presentation, or book + slideshow presentation along with a tool or game to teach the new concept, which of the two will garner the most student engagement? This is the main question that I pose for this area of research. My immediate hypothesis would be that a presentation with a tool or a game would definitely increase student engagement; mainly because of person experience and preference. My goal is to collect data and either support or deny my hypothesis.

6.2 Testing Groups

6.2.1 Control Group

With this group, I needed to present the PATS problem using only a slideshow presentation. This was to gauge engagement from a simple presentation, and to gather information using a small survey that I created using Google Forms. The survey for the control group can be found below

The main area that I wanted to gather data from was student engagement when being confronted against a new problem, and so the main question that I wanted to get from the students was how engaged they felt during the presentation.


Engagement Survey

Form description

How engaged did you feel during this presentation? *

	1	2	3	4	5	
Least Engaged	○	○	○	○	○	Very Engaged

Do you think a game or a tool can help with understanding the presentation concept? *

○ Yes

○ No

○ Maybe

Any additional comments?

Long answer text

Figure 6.1: Survey (No Site)

6.2.2 Testing Group

With this group, I needed to present the PATS problem using both the application that I had made along with the presentation. The end goal being to compare the survey results against each other.

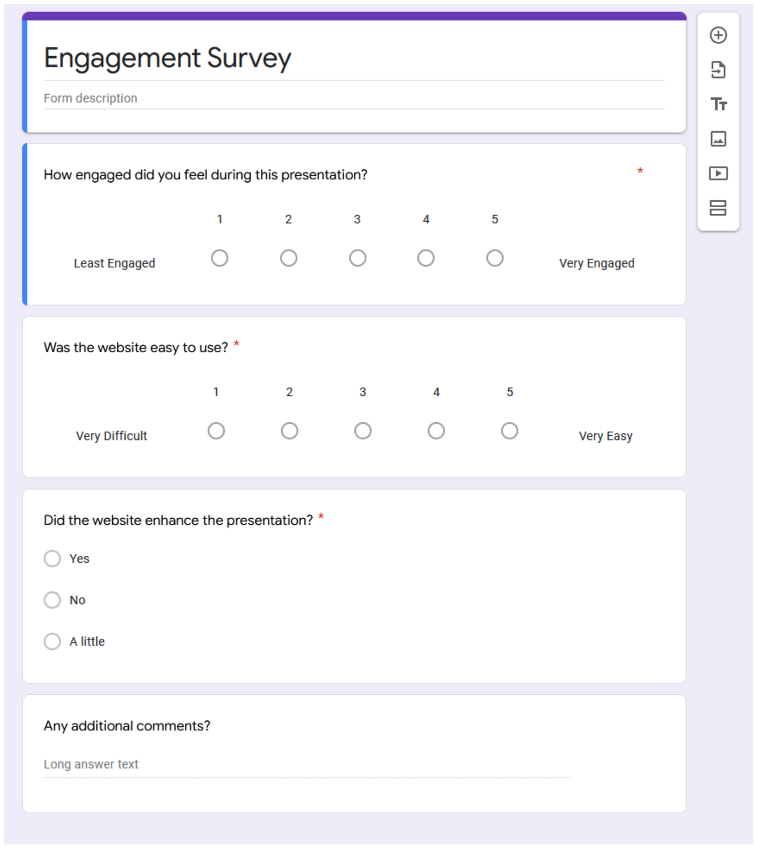
The main question here remains the same, with the exception of gauging whether or not the application was easy to use and if it enhanced the presentation itself.

6.3 Testing Results

Here I showcase the data that I collected when giving the two presentations; one with the application, and one without the application.

6.3.1 Results Using The Site

In Figure 6.3, I ask the students how engaged they felt during the duration of the presentation; out of the 9 who answered, 4 answered a 3/5, 2 answered a 4/5, and 3 answered a 5/5 engagement.


Engagement Survey

Form description

How engaged did you feel during this presentation? *

	1	2	3	4	5	
Least Engaged	○	○	○	○	○	Very Engaged

Was the website easy to use? *

	1	2	3	4	5	
Very Difficult	○	○	○	○	○	Very Easy

Did the website enhance the presentation? *

○ Yes

○ No

○ A little

Any additional comments?

Long answer text

Figure 6.2: Survey (No Site)

These results are not too bad in terms of student engagement when learning about a new theoretical topic.

In Figure 6.4, I asked the students if the site was easy to use. Out of the 9 who answered, 1 answered a 3/5, 3 answered a 4/5, and 5 answered a 5/5. This is a really good result in terms of the website, as the majority voted that the ease of use was relatively high.

When asking the students if the website enhanced the presentation that I was giving on the PATS problem, all 9 students voted "Yes", as shown in Figure 6.5. This ultimately leads to me conclude, in this case, that using tools or games can most definitely enhance presentations when showing students new concepts.

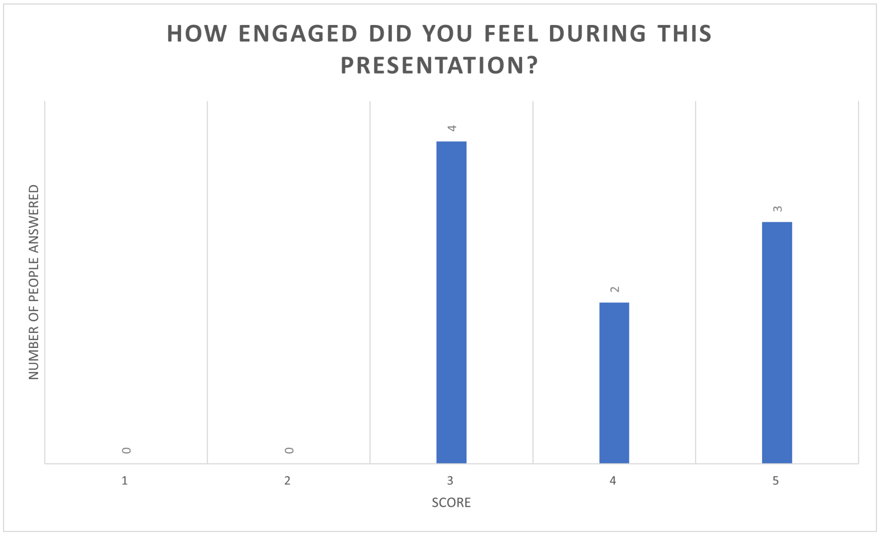


Figure 6.3: How engaged did you feel during this presentation? (Site)

6.3.2 Results Not Using The Site

Giving my presentation without the website, shown in Figure 6.6, I asked the new group of students the same question on how engaged they felt. Out of the 35 students that answered this question, 1 answered 1/5, 1 answered 2/5, 11 answered 3/5, 17 answered 4/5, and 5 answered 5/5. This distribution is quite nice, as it showcases good engagement with the students even without using a tool or a game.

In Figure 6.7, I ask the students whether or not they think a game or a tool can help with understanding of presentation concepts, and out of the 35 students that answered, 29 answered "Yes", while 6 answered "Maybe". I think this further reinforces my idea that games or software tools are just excellent supplementary material for students when they're learning new concepts.

6.3.3 Summary

In summary, considering the data that I collected during my two presentations to two separate classes, I can comfortably say and reinforce the idea that games or tools while teaching can enhance the learning experience for students that are learning new material. This is supported by the evidence in Figure 6.3, 6.4, 6.5, 6.6, and 6.7.

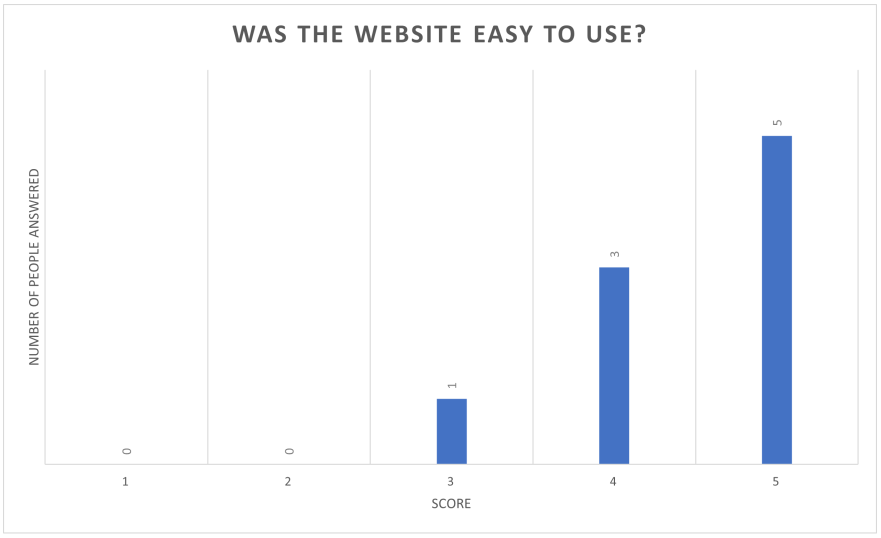


Figure 6.4: Was the website easy to use? (Site)

6.3.4 Future Work

Future work in this category of research, and at least, in my case, would be to continue implementing and creating tools for students to use in difficult courses, trying to create games to nail in a theoretical concept that can help the students learn them. The opportunity to create such games is vast and immense, and I think it would be a very fun time and activity for students; especially those that might be experiencing difficulty with certain concepts, and while it might be very difficult or in vain to create a game or tool for every subject, it might be worth to try to create these experiences in the areas that students struggle the most, and so knowing what these areas are might be a next step forward.

6.4 Conclusion

To conclude this piece of work, I was able to show some results that my team and I were able to come up with in terms of games, and some observations that were also made when it came to tile based self-assembly. I was able to explain the PATS problem, and explain why I ended up choosing that problem to create a game and show why games are such a really good medium to learn new concepts; even in academia. I showcased the game that I created, going over the details

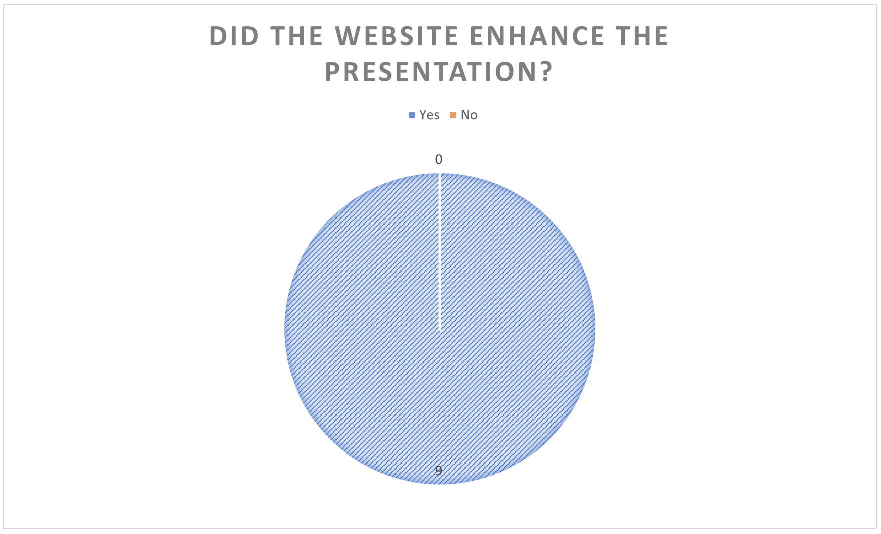


Figure 6.5: Did the website enhance the presentation? (Site)

of the software and how I built it from the ground up. After showing how I built the game, I showed how I tested the program, and the results that came from testing the game in two different class environments. Finally, I concluded that, based on the results that I collected, I can confidently say that games and/or tools are favorable amongst students and that they can help increase overall student engagement when learning new concepts.

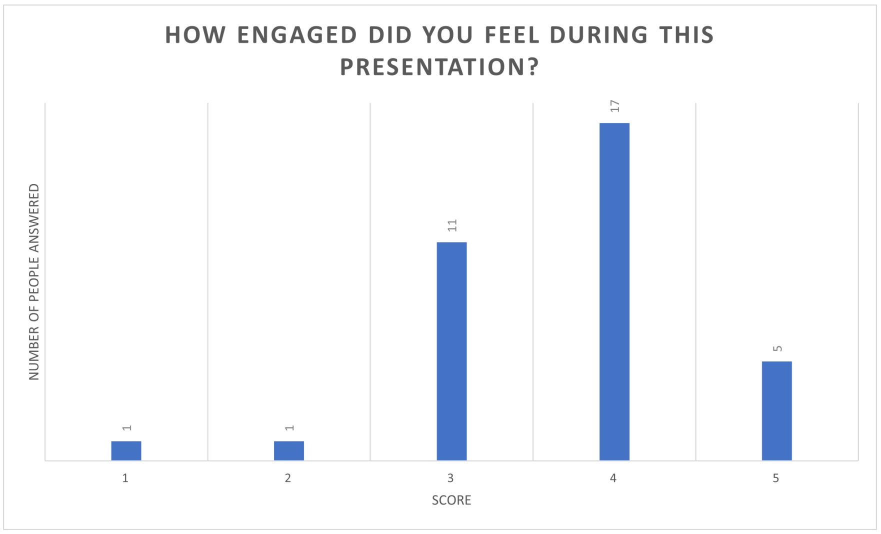


Figure 6.6: How engaged did you feel during this presentation? (No Site)

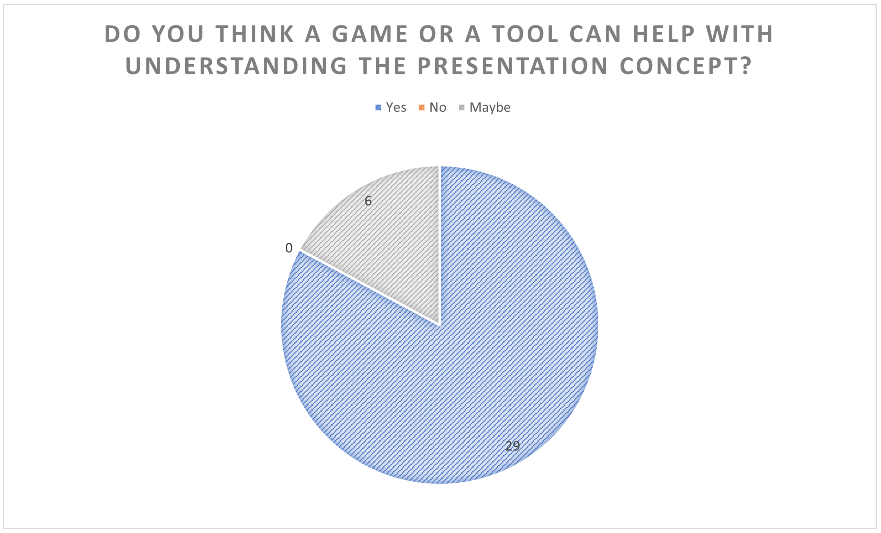


Figure 6.7: Do you think a game or a tool can help with understanding the presentation concept? (No Site)

REFERENCES

[1] G. Aloupis, E. D. Demaine, A. Guo, and G. Viglietta, *Classic nintendo games are (computationally) hard*, 2012.

[2] B. Amir Parviz, D. Ryan, and G. M. Whitesides, *Using self-assembly for the fabrication of nano-scale electronic and photonic devices*, IEEE Transactions on Advanced Packaging, 26 (2003), pp. 233–241.

[3] E. Andersen, M. Dong, and M. Nielsen, *Self-assembly of a nanoscale dna box with a controllable lid*, Nature, 459 (2009), pp. 73–76.

[4] J. Balanza-Martinez, D. Caballero, A. Cantu, T. Gomez, A. Luchsinger, R. Schweller, and T. Wylie, *Relocation with uniform external control in limited directions*, The 22nd Japan Conference on Discrete and Computational Geometry, Graphs, and Games, JCDCGGG.

[5] A. Becker, E. D. Demaine, S. P. Fekete, G. Habibi, and J. McLurkin, *Reconfiguring massive particle swarms with limited, global control*, in Algorithms for Sensor Systems, P. Flocchini, J. Gao, E. Kranakis, and F. Meyer auf der Heide, eds., Berlin, Heidelberg, 2014, Springer Berlin Heidelberg, pp. 51–66.

[6] A. Becker, G. Habibi, J. Werfel, M. Rubenstein, and J. McLurkin, *Massive uniform manipulation: Controlling large populations of simple robots with a common input signal*, in 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2013, pp. 520–527.

[7] J. Bosboom, J. Brunner, M. Coulombe, E. Demaine, D. Hendrickson, J. Lynch, and E. Najt, *The legend of zelda: The complexity of mechanics*, 03 2022.

[8] J. Brunner, L. Chung, E. Demaine, D. Hendrickson, A. Hesterberg, A. Suhl, and A. Zeff, *1 x 1 rush hour with fixed blocks is pspace-complete*, 03 2020.

[9] D. Caballero, A. A. Cantu, T. Gomez, A. Luchsinger, R. Schweller, and T. Wylie, *Relocating units in robot swarms with uniform control signals is pspace-complete*, Canadian Conference on Computational Geometry.

[10] D. Caballero, A. A. Cantu, T. Gomez, A. Luchsinger, R. T. Schweller, and T. Wylie, *Hardness of reconfiguring robot swarms with uniform external control in limited directions*, CoRR, abs/2003.13097 (2020).

[11] S. CANNON, E. D. DEMAINE, M. L. DEMAINE, S. EISENSTAT, M. J. PATITZ, R. T. SCHWELLER, S. M. SUMMERS, AND A. WINSLOW, *Two hands are better than one (up to constant factors)*, CoRR, abs/1201.1650 (2012).

[12] C. CHALK, A. LUCHSINGER, E. MARTINEZ, R. SCHWELLER, A. WINSLOW, AND T. WYLIE, *Freezing simulates non-freezing tile automata*, in DNA Computing and Molecular Programming, D. Doty and H. Dietz, eds., Cham, 2018, Springer International Publishing, pp. 155–172.

[13] C. T. CHALK, E. D. DEMAINE, M. L. DEMAINE, E. MARTINEZ, R. T. SCHWELLER, L. VEGA, AND T. WYLIE, *Universal shape replicators via self-assembly with attractive and repulsive forces*, CoRR, abs/1608.00477 (2016).

[14] C. T. CHALK, E. MARTINEZ, R. T. SCHWELLER, L. VEGA, A. WINSLOW, AND T. WYLIE, *Optimal staged self-assembly of general shapes*, CoRR, abs/1510.03919 (2015).

[15] M. S. DAVID LICHSTEINE, *Go is polynominal-space hard*, Journal of the Association for Computing Machinery, 27 (1978), pp. 393–401.

[16] E. DEMAINE, R. HEARN, AND M. HOFFMANN, *Push2-f is pspace-complete*, (2002), pp. 31–35.

[17] D. DOTY, *Producibility in hierarchical self-assembly*, CoRR, abs/1304.7804 (2013).

[18] D. DOTY, M. J. PATITZ, D. REISHUS, R. T. SCHWELLER, AND S. M. SUMMERS, *Strong fault-tolerance for self-assembly with fuzzy temperature*, in 2010 IEEE 51st Annual Symposium on Foundations of Computer Science, 2010, pp. 417–426.

[19] R. HEARN AND E. DEMAINE, *Pspace-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation*, Theoretical Computer Science, 343 (2002), pp. 72–96.

[20] B. HOFFMAN, R. MORELLI, AND J. ROSATO, *Student engagement is key to broadening participation in cs*, in Proceedings of the 50th ACM Technical Symposium on Computer Science Education, SIGCSE '19, New York, NY, USA, 2019, Association for Computing Machinery, p. 1123–1129.

[21] H. HU, M. GOPINADHAN, AND C. OSUJI, *Directed self-assembly of block copolymers: a tutorial review of strategies for enabling nanotechnology with soft matter.*, Soft matter, 10 22 (2014), pp. 3867–89.

[22] A. KHALED, S. GUO, F. LI, AND P. GUO, *Controllable self-assembly of nanoparticles for specific delivery of multiple therapeutic molecules to cancer cells using rna nanotechnology*, Nano letters, 5 (2005), pp. 1797–808.

[23] X. MA AND F. LOMBARDI, *Synthesis of tile sets for dna self-assembly*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 27 (2008), pp. 963–967.

[24] X. Ma and F. Lombardi, *On the computational complexity of tile set synthesis for dna self-assembly*, IEEE Transactions on Circuits and Systems II: Express Briefs, 56 (2009), pp. 31–35.

[25] S. Marwan, G. Gao, S. Fisk, T. W. Price, and T. Barnes, *Adaptive immediate feedback can improve novice programming engagement and intention to persist in computer science*, in Proceedings of the 2020 ACM Conference on International Computing Education Research, ICER '20, New York, NY, USA, 2020, Association for Computing Machinery, p. 194–203.

[26] P. W. K. Rothemund and E. Winfree, *The program-size complexity of self-assembled squares (extended abstract)*, in Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, STOC '00, New York, NY, USA, 2000, Association for Computing Machinery, p. 459–468.

[27] J. Sinclair, M. Butler, M. Morgan, and S. Kalvala, *Measures of student engagement in computer science*, Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education, (2015).

[28] D. Soloveichik and E. Winfree, *Complexity of self-assembled shapes*, CoRR, abs/cs/0412096 (2004).

[29] H. Wang, *Proving theorems by pattern recognition — ii*, The Bell System Technical Journal, 40 (1961), pp. 1–41.

[30] G. M. Whitesides and B. A. Grzybowski, *Self-assembly at all scales*, Science, 295 (2002), pp. 2418–2421. 793.

[31] E. Winfree, *Algorithmic self-assembly of dna*, (1998).

[32] D. Woods, H. Chen, S. Goodfriend, N. Dabby, E. Winfree, and P. Yin, *Active self-assembly of algorithmic shapes and patterns in polylogarithmic time*, CoRR, abs/1301.2626 (2013).

BIOGRAPHICAL SKETCH

Francisco Gonzalez is from Monterrey, Mexico; ever since High School, he pursued Computer Science as a passion career, and graduated from The University of Texas at Rio Grande Valley with his Bachelor's Degree in 2020, and with his Master's Degree in 2022. He continues to pursue his passion to this day. He can be reached via his personal e-mail at pacog879@hotmail.com for any questions!