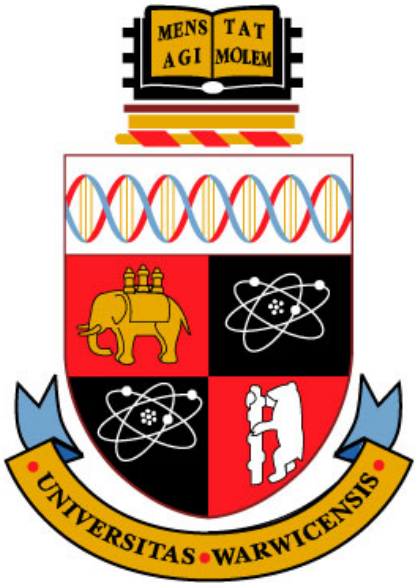
CS344: Discrete Mathematics Project

A Game to Teach Combinatorics at University

Author: **Joshua Humphriss**

Supervisor: **Alexander Dixon**



Department of Computer Science
University of Warwick

Abstract

Teaching at university often relies on lectures and seminars, however games are well known to be more effective than traditional teaching methods. Combinatorics is a mathematical field of study often taught at university, concerned with counting problems, graph theory and other similar problems within a discrete system. This project is a video game as a companion teaching resource for combinatorics at university level, designed to sit alongside lectures and seminars.

The game helps students practise how to find bijections, prove the planarity and non-planarity of graphs and find proper colourings in a graph. The game helps students by giving instant validation, automating tedious aspects of problem solving, and guiding players towards a solution. Video games are also inherently more engaging as a medium for education, motivating players to learn more. The project also contributes new results and new implementations of mathematical ideas.

User testing consistently found the game to be fun and useful in education, both for those studying combinatorics and a wider audience. This demonstrates the feasibility of video games in teaching mathematics at university, and provides a fun puzzle game for use outside of education. Finally, the project opens many avenues for future work to incorporate more areas of combinatorics into the game.

Keywords: *Combinatorics, Graph Theory, Educational Tool, Video Game, Godot.*
Mark Achieved: *79% (1st).*

Acknowledgements

I would like to thank my supervisor, Alex Dixon, for consistently being enthusiastic to test out the game and giving useful feedback throughout the process. I also would like to thank Rob Silversmith for teaching me combinatorics in an engaging way, and for meeting with me and offering some advice for the project. Finally, I would like to thank everyone who has voluntarily tested out the game, whether part of evaluation day or other testing. In particular, Saumya Shah, Samuel Humphriss, Alice George and Josh Heng for giving up their time to test the project in depth, despite having deadlines of their own.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

All topics in mathematics have elements of memorising content and elements of practising techniques. Combinatorics [19] [36] is a field of study that is primarily focused on practising techniques, meaning the best way to learn is through practice. Furthermore, games such as The Witness [56] demonstrate the wide appeal and fun of combinatorial problems [16] in a less formal setting. It may be surprising to learn that most students do not practise these techniques beyond the required assignments. This project aims to change this by creating a puzzle game that will increase motivation for students, and help them to focus on the mathematical aspects of the problems by automating tedious aspects of using paper.

Finding a bijection between two different counting problems is a common technique to prove that they are equivalent [44]. The first topic allows students to practise this technique through some hand-selected problems. The second topic extends this idea by offering randomly-generated problems from the Catalan numbers [54]. Both topics include a graphical input method which helps students to develop intuition, and a parser for a custom programming language which offers near-perfect validation for the bijections.

The third and fourth topics involve graphs. While many games already exist that ask players to prove a graph is planar, none were found that also ask the player to prove a graph is non-planar. This topic presents a random graph, upon which the player must deduce its planarity and construct a proof, either by rearranging the vertices or using the Kuratowski-Wagner theorem [39] [60]. The graph colouring topic also presents a random graph, but instead asks for a proper vertex colouring of minimum size. The player may also establish bounds on the chromatic number, including with the four colour theorem.

All of these topics pose problems very similar to those commonly found in exams, so they are useful for university students. Immediate validation gives students instant feedback without needing to check their solutions themselves. Tedious aspects of solving problems on paper such as writing out small cases or manipulating graphs are automated. Constraints are imposed to naturally guide players towards a solution, and many problems are randomly-generated. Finally, the visuals, narrative and world design come together to make the game more engaging for studying as well as a fun game to be played in free time.

As a mathematical field, combinatorics is primarily studied without computers. This project contributes to the field by applying computational approaches to well-known ideas. For example, the Catalan number zone contributes new bijections and implementations for some well-known bijections. It also contributes drawing algorithms in Godot for a wide range of elements. A well-structured graphs library for Godot and a parser for a custom programming language also form part of the project's contribution. This project demonstrates the feasibility of video games in higher education for mathematics, an area that is under-explored.

Chapter 2

Background

We first seek to understand each topic in more depth, and then consider the use of games in education in general and any similar games.

2.1 Topics

This section provides a detailed explanation of each topic.

2.1.1 Bijections

A counting problem is a problem about counting. For this topic and the next, we will use a narrower definition. A counting problem is a sequence $(A_n)_{n\in\mathbb{N}}$ where each A_n is a finite set ($|A_n|$ is the answer to the counting problem). We may refer to n as the problem size, and for some fixed $m \in \mathbb{N}$, all $a \in A_m$ are called elements. Two examples of such counting problems are given below.

Counting Problem 1. $A_n = \{$*integer partitions of* n *where each part is odd*$\}$.

Counting Problem 2. $B_n = \{$*integer partitions of* n *where each part is distinct*$\}$.

We are often interested in the relationship between different counting problems. In particular, proving that two problems are equivalent.

Definition. *Two counting problems* A_n *and* B_n *are equivalent if* $\forall\, n \in \mathbb{N}$, $|A_n| = |B_n|$.

There are two main approaches for proving that two counting problems are equivalent. One approach is to generate a closed form expression for $|A_n|$ and $|B_n|$, and show that they are equal. However, this is often difficult, and sometimes no such closed-form expression exists. Instead, we can find a bijection between the elements of A_n and the elements of B_n. If such a bijection exists for all n, then $|A_n| = |B_n|$. This topic allows the player to practise this well-established and important technique on some hand-selected problems.

2.1.2 Catalan Number Zone

The Catalan number zone extends the bijections topic by offering procedurally-generated problems. The Catalan number are a famous sequence counted by over 214 [54] equivalent counting problems, meaning the game may simply choose any two random problems and ask for a bijection between them. The final game includes 8 counting problems, giving $\binom{8}{2} = 36$ total bijections to find.

2.1.3 Planar Graphs

Many games exist which ask the player to rearrange a given graph into planar form. However, none were found that also ask the player to prove that a graph is non-planar. This topic will present players with a random graph, to which players must deduce the planarity and then provide a proof. Proving that graphs are planar is done by rearranging the vertices to find a planar drawing. To prove a graph is non-planar, we will use the Kuratowski-Wagner theorem[1].

Theorem (Kuratowski 1930 [39], Wagner 1937 [60]). *A graph is non-planar if and only if it has a K_5 or a $K_{3,3}$ as a minor.*

Definition (Minor). *A graph G' is a minor of a graph G if it can be obtained from G by a sequence of vertex deletions, edge deletions and edge contractions.*

Definition (Edge Contraction). *An edge contraction of $(u, v) \in E$ in $G = (V, E)$ produces a graph $G' = (V \cup \{v'\} - \{u, v\}, E')$ such that $N(v') = N(u) \cup N(v)$.*[2]

The Kuratowski-Wagner theorem is suitable here because:

- It uses an algorithmic process which is amenable to validation.
- For any non-planar graph, a proof using this theorem will exist.
- It is commonly used to prove non-planarity in an exam setting.

2.1.4 Graph Colouring

Definition (Graph Colouring). *A graph colouring, proper vertex colouring or k-colouring of a graph $G = (V, E)$, is a function $c : V \to \{1, ..., k\}$ such that for every $(u, v) \in E$, $c(u) \neq c(v)$.*

Definition. *The chromatic number of G, $\chi(G)$, is the minimum number of colours required to colour G.*

Graph colouring is an important problem in combinatorics. The topic presents players with a random graph G and has two parts: finding an optimal graph colouring of G, and establishing bounds on the chromatic number $\chi(G)$.

Sometimes, the end goal is to obtain a bound on $\chi(G)$ and the actual graph colouring merely serves as an upper bound (if G admits a k-colouring, then $\chi(G) \leq k$). Other times, the bounds are useful to help find the optimal graph colouring (if G has a 4-clique, then there is no point attempting to find a 3-colouring). There are four theorems that bound the chromatic number. One of these is the four colour theorem, providing a useful link to planar graphs.

[1] The Kuratowski-Wagner theorem exists in many forms. We will be using Wagner's theorem [60] which takes strong inspiration from Kuratowski's theorem [39] published 7 years earlier.

[2] Here, $-$ is the set difference operator and $N(v)$ denotes the neighbourhood of a vertex v.

2.2 Games in Education

Video games have often been found to be effective in education. While most studies concern primary and secondary students, some studies also exist for higher education.

One study was conducted on a group of secondary school maths students [64]. The study concluded that games were somewhat effective for all students, but that they were particularly effective for students with a strong interest in maths. As university students have chosen their course, it is expected they will have a strong interest in their subject area. The study also emphasised the need for games to fit into the existing pedagogy rather than replace existing resources, as students prefer interaction with teachers and parents do not support pedagogical change. This project will not be aiming to replace existing resources. Instead, this project is designed to be an additional resource, as all students benefit from receiving information in multiple ways [31, p. 45].

One of the studies in higher education is by Warwick Business School [30], specifically on the use of games in mathematical fields. The study found improved learning outcomes, student satisfaction and attendance (see table 2.1). It demonstrates the utility of games at university, however these games were done within seminar groups and not independently.

Criterion	Control	Treatment
Median mark	60%	69%
Fail rate	20%	7%
Student satisfaction[3]	3.91	4.76
Mean attendance[4]	88.5%	94.4%

Table 2.1: Results of the Warwick Business School study.

A study by the University of Glasgow [17] explored the utility of games in improving skills such as communication, resourcefulness and adaptability. The study was done on humanities students using commercial video games. After 8 weeks, participants reported an increase in communication, critical thinking and reflective learning skills, all skills which are commonly desired by employers. While this project will be focused on exam results, this demonstrates the potential for video games to also improve soft skills. The study remarked that developing imaginative problem solving may be unique to gaming, and that games may be able to teach things that are currently impossible to teach in higher education.

2.3 Existing Solutions

Similar games include The Witness, ProofBlocks, Jason Davies' Planar Graphs and Chromagraph.

2.3.1 The Witness

The Witness [56] is a popular puzzle game released in 2016. Players are presented with constraint-satisfaction puzzles involving drawing lines across a grid. While these are not intended as problems

[3]Results when students were asked to rate how much they agree with the statement, "I am satisfied with the teaching", from 1 to 5 with 1 being the worst and 5 being the best.

[4]Mean attendance at seminars 2-9, divided by the attendance at seminar 1.

Figure 2.1: A screenshot from The Witness, showing some advanced puzzles. [2]

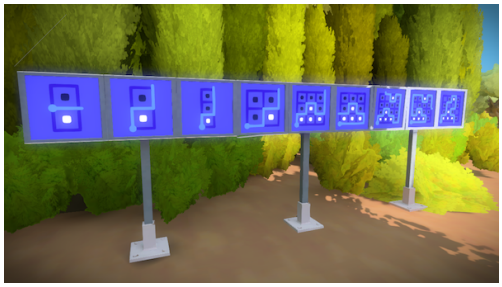


Figure 2.2: A screenshot from The Witness, demonstrating how it teaches mechanics. [3]

relating to formal combinatorics, a group of researchers studied these problems [16] and found relation to many problems in combinatorics such as Hamiltonian cycles, planar graphs and counting subsets. The Witness is widely renowned as a fun video game [1], demonstrating the wide appeal and enjoyment available in combinatorial video games. The Witness is also very successful at teaching players how to solve problems of increasing difficulty, demonstrating the ability for video games to be an effective resource in education.

One particular aspect of The Witness is its approach to teaching mechanics. The Witness explains puzzle mechanics through posing simple problems, expecting players to spot patterns and intuit how the mechanic works, and then apply this to more complex problems involving the same mechanic. This approach is very effective at teaching mechanics in an engaging way, testing the player's problem solving skills. This is similar to the approach that is often used to find bijections - write out the small cases, spot patterns, and then extrapolate on more complex versions of the problem. However, this approach will not be suitable for the graph topics, where it is important that students become familiar with terminology instead of mere associations to symbols used in the game.

2.3.2 ProofBlocks

ProofBlocks [4] is a printable exercise for use in secondary schools. Blocks are provided which each represent a technique that students can use to solve a problem. Students must rearrange these blocks to construct a solution to a problem. This is similar to solving counting problems from combinatorics - combining techniques from a toolbox to solve a problem.

This is a useful example as it demonstrates that extensive automated checking or game aspects aren't required for a product to make an impact in education. It demonstrates the value that comes from merely constraining the player to help guide them towards a solution. All of the topics in this project will impose constraints on the player to guide them towards a solution.

2.3.3 Jason Davies' Planar Graphs

Jason Davies' Planar Graphs [5] is one of many web-based games about rearranging graphs into planar form. While this game is very fun, there are two main weaknesses.

Graph Manipulation Sometimes, when moving lots of vertices inside, the available space becomes limited. A tool to expand the whole graph would give more space. Sometimes, when bringing

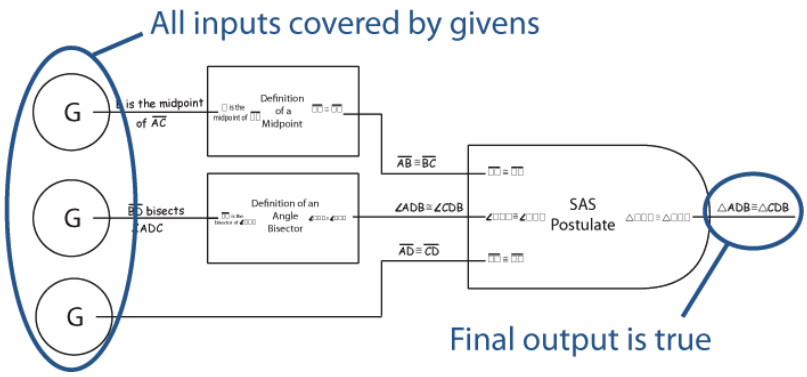


Figure 2.3: An image of ProofBlocks demonstrating a simple proof using multiple blocks.

vertices outside, there is insufficient space within bounds to place the vertex. A tool to shrink the whole graph would help with this. Sometimes, it is also beneficial to completely redraw a graph to view it in a different way. The limited graph manipulation in this solution was an obstacle in engaging with the mathematical aspects of the problem.

Non-Planar Graphs It is rare for exam questions to only present planar graphs. Often, graphs are presented of unknown planarity and the student has to deduce its planarity and then construct a proof. The existing planar graph games only cover half of the problem.

2.3.4 Chromagraph

Chromagraph [6] is a puzzle game that presents various graph colouring problems. While it begins with unconstrained graph colouring, there are many additional mechanics that are combined together to create more complex puzzles. While these puzzles are generally more similar to formal combinatorics than The Witness, they are still not similar enough to be useful in education. Chromagraph takes inspiration from how The Witness teaches new puzzle mechanics, which is effective although still does not use the relevant terminology. Chromagraph further reinforces the effectiveness of a puzzle game in formal combinatorics.

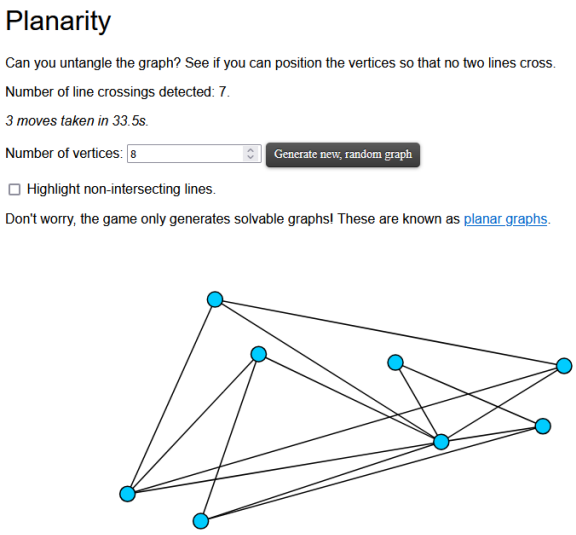


Figure 2.4: A screenshot from Jason Davies' planar graphs game.

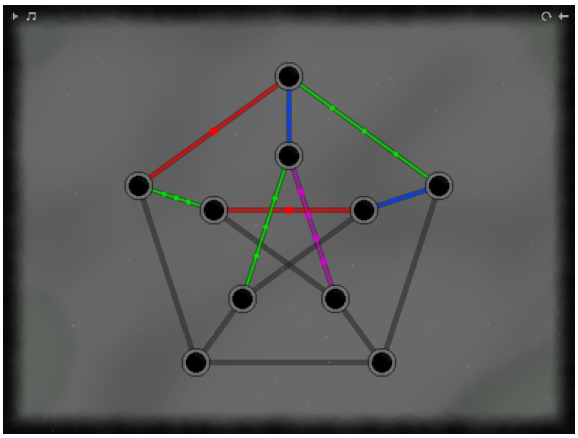


Figure 2.5: A screenshot from Chromagraph.

Chapter 3

Requirements

This chapter formalises the purpose and intended impact of the solution, and then devises some requirements to achieve the objectives.

3.1 Objectives

The project has four objectives. These remain unchanged since the specification.

1. Content The game should include a variety of interesting topics from combinatorics that are commonly taught at university.

2. Learning The game should improve learning outcomes for those studying combinatorics at university.

3. Fun The game should be engaging for those studying combinatorics. As a bonus, it should also be fun for people who are not studying combinatorics.

4. Accessibility The game should be as inclusive as possible.

3.2 Stakeholders & Impact

The project primarily aims to add value for students and module organisers. The game may also contribute value for a wider audience who are not studying combinatorics formally.

3.2.1 Students

The value added for the student is:

- Automatic validation, providing students with instant feedback.
- Automating the tedious aspects of solving problems on paper to allow students to focus on the mathematical aspects. In particular:

  - When finding bijections, writing out the small cases can be time consuming and some students attempt to skip this step and subsequently struggle to solve the problem.
  - Manipulating graphs on paper can involve a lot of rubbing out, or copying down subsequent steps. This often takes far longer than necessary, as lots of time is spent on non-mathematical aspects of the problem.
- Students are more likely to discover a solution themselves due to the constraints imposed. The bijections hint system also helps to lead players towards a solution without revealing it.
- Randomly generated problems give near-unlimited possibilities.
- Students can easily explore how the size and density of a graph affects its properties.
- The world design, graphics and narrative make it more engaging than a worksheet.
- All points from subsection 3.2.3 apply too.

3.2.2 The Module Organiser

The value added for the module organiser and teaching staff is:

- Saves time in creating worksheets for students.
- Students are more likely to discover solutions on their own, hence are less likely to require additional support from teaching staff.
- They may use the random generation to discover new and interesting problems.
- Students will have a better understanding of the content as a result of playing the game.

3.2.3 A Wider Audience

The value added for people who are not studying combinatorics is:

- A fun game to play in their free time.
- Increased interest in combinatorics.
- Increased problem solving skills.
- Increased ability at combinatorics, applying to many areas of pure and applied mathematics.

3.3 Using the Solution

It is not yet clear how the solution will fit into the existing pedagogy. There are two main approaches, which ultimately come from which stakeholder is the customer.

Module Organiser Most educational software is sold to educational institutions as a tool to help with teaching. These educational tools often feature a functional aesthetic and give a greater degree of control to the teaching staff, such as setting tailored worksheets and collecting in results. Students may be required to use a particular piece of software as part of their course. Educational

software often fixes learning objectives first, and neglects to make the software fun to use when it would conflict with the learning objectives.

Student Most video games are sold to players directly. This approach would focus primarily on the needs of the student, giving equal weight to the learning outcomes and entertainment value of the solution. This will be more beneficial to the student, although the learning objectives be adjusted to create a more fun game, and therefore differ from the original plan. This approach also provides value for a wider audience than those studying combinatorics formally.

This project will choose the latter approach. This is beneficial to the project as provides freedom to go beyond the confines of a particular course. Focusing on the needs of the students using the product allows for a greater impact, and presents opportunities to reach a wider audience that would not be available in a strictly educational tool. Most importantly, the impact of the project is not dependent on the endorsement of module organisers. Module organisers are often slow to incorporate new methods of teaching, so this approach will allow students to use the product without the teaching staff needing to change their methods of teaching. However, students may be more likely to play the game if the teaching staff were to endorse it.

As a result, the project is primarily a video game over an educational piece of software. It features playful visuals, dialogue, and creative world design to increase engagement. While the need for educational impact is obvious, it is also important the game is fun to play otherwise it will not be used. Therefore, we assign equal priority to the learning outcomes and fun of the solution. While module organisers may prefer a more functional aesthetic, students tend to prefer a more engaging feel, so this is the chosen direction for the project.

3.4 Learning Objectives

The learning objectives outline the intended educational impact of the game, to fulfil objective 2. They are not intended as an exhaustive list of what a player may learn when playing the game. The learning objectives were formed iteratively, in accordance with the agile development methodology.

1. Bijections and Catalan Number Zone.
   (a) Prove that two counting problems are equivalent by finding a bijection between them.
   (b) Recognise the presence of multiple solutions to the same problem.
   (c) Create new bijections that have not yet been discovered.
   (d) Write down bijections using code.
   (e) **[Level 1]** Find a bijection between subsets and binary strings.
   (f) **[Level 2]** Provide a bijective proof for the 'stars and bars' combinatorial technique.
   (g) **[Level 3]** Understand the Robinson-Schensted correspondence.
   (h) **[Levels 4 & 5]** Define what is meant by an integer partition.
   (i) **[Level 4]** Create a bijection between self-conjugate integer partitions and distinct odd integer partitions.

(j) **[Level 5]** Create a bijection between distinct integer partitions and odd integer partitions.

2. Planar Graphs.
   (a) Define what is meant by a planar graph and a planar drawing.
   (b) Distinguish between some planar and non-planar graphs by initial observation.
   (c) Prove that a graph is planar by rearranging its vertices.
   (d) Recall the Kuratowski-Wagner theorem.
   (e) Construct a minor of a graph, including understanding of edge contractions.
   (f) Prove that a variety of graphs are non-planar using the Kuratowski-Wagner theorem.
   (g) Recognise the differences in graphs that contain a K_5 and graphs that contain a $K_{3,3}$.
   (h) Prove planarity of larger graphs than would be feasible on paper.
   (i) Recognise how the density of a graph affects its planarity.
3. Graph Colouring.
   (a) Define what is meant by a graph colouring.
   (b) Find graph colouring on a variety of different graphs.
   (c) Use the maximum degree of a graph to obtain an upper bound on its chromatic number.
   (d) Use the four colour theorem by rearranging graphs into planar form.
   (e) Use the largest clique in a graph to obtain a lower bound on its chromatic number.
   (f) Find graph colourings in larger graphs than would be feasible on paper.
   (g) Recognise how the size and density of a graph affects its chromatic number.
4. Counting Problems.
   (a) **[Level 1]** Count the number of different orders for a small list of items.
   (b) **[Level 2]** Use stars and bars to determine how many ways to distribute items into buckets, where some items are fixed to some buckets.
   (c) **[Level 3]** Count the number of graphs on a given number of vertices.
   (d) **[Level 4]** Use the Inclusion-Exclusion principle.

3.5 Functional and Non-Functional Requirements

The requirements are designed so that a solution implementing these requirements should achieve the objectives in section 3.1 (content, learning, fun, accessibility).

The requirements were developed iteratively. New requirements were introduced as new features were added into the scope, although unmet requirements were not removed. Chapter 7 will evaluate against the unchanged objectives and not against the changing requirements.

We use the MoSCoW framework [23] (M = must, S = should, C = could, W = won't).

1. Requirements pertaining to Bijections and the Catalan Number Zone:
   (a) **[M]** Provide an interface whereby players can match up the elements for the first few values of n.
   (b) **[S]** Accept some alternate solutions.
   (c) **[S]** Validate the player's bijection with high accuracy.
   (d) **[S]** Be as easy as possible for a player to input their solution once they've thought of it.
   (e) **[S]** Define all key terms used.
   (f) **[S]** Contain a hints system to encourage the player to find a solution.
2. Requirements pertaining to Bijections only:
   (a) **[M]** Contain at least five levels.
   (b) **[S]** Contain at least eight levels.
   (c) **[S]** Contain integer partition problems.
   (d) **[S]** Draw a Young/Ferrer diagram for integer partitions.
   (e) **[C]** Display a proof for each level to the user.
3. Requirements pertaining to the Catalan Number Zone only:
   (a) **[M]** Choose two random Catalan problems and ask for a bijection between them.
   (b) **[M]** Contain at least five problems.
   (c) **[S]** Contain problems that look and feel different from each other.
   (d) **[S]** Contain at least ten problems.
   (e) **[C]** Have a difficulty setting to generate problems of a desired difficulty.
   (f) **[C]** For each pair of problems, display all known proofs, including transitively via other problems.
   (g) **[C]** Contain at least fifteen bijections.
   (h) **[C]** Store which bijections the user has already encountered and avoid showing the same pair of problems twice.
4. Requirements pertaining to Planar Graphs and Graph Colourings:
   (a) **[M]** Generate random graphs.
   (b) **[S]** Allow the user to customise the size and density of graphs.
   (c) **[S]** Provide a button to skip the current graph, even if not completed correctly.

(d) **[C]** Feature multiple ways of generating graphs.

5. Requirements pertaining to Planar Graphs only:
   (a) **[M]** Allow the user to drag vertices to rearrange them.
   (b) **[M]** Determine if the drawing is planar.
   (c) **[M]** Allow the user to advance to another graph when a planar drawing is reached.
   (d) **[M]** Allow the user to do vertex deletions, edge deletions and edge contraction.
   (e) **[M]** Allow the user to advance to another graph when a K_5 or a $K_{3,3}$ is reached.
   (f) **[S]** Highlight which edges are crossing.
   (g) **[S]** Preserve the progress of a planar proof and an non-planar proof when switching.
   (h) **[S]** In the planar side, provide tools to expand, shrink or redraw the whole graph.
   (i) **[C]** Add a setting to generate only planar or only non-planar graphs.
   (j) **[C]** Feature a time-limited challenge mode.
   (k) **[C]** Save the user's high scores.
6. Requirements pertaining to Graph Colourings only:
   (a) **[M]** Allow the user to change the colour of each vertex.
   (b) **[M]** Allow the user to advance to another graph if they obtain a valid graph colouring.
   (c) **[S]** Include a section to establish bounds on the chromatic number, with at least one upper bound and at least one lower bound.
   (d) **[S]** Allow the user to advance to another graph only if they obtain a valid colouring that could be optimal.
   (e) **[S]** Highlight which edges have both endpoints with the same colour.
   (f) **[C]** Include the four colour theorem by asking the user to obtain a planar drawing.
   (g) **[C]** Use the current best upper bound to inform how many colours are available for the user to choose between.
7. Requirements pertaining to the dialogue:
   (a) **[S]** Be light-hearted and funny, to offer a break between complex puzzles.
   (b) **[S]** Contain multiple different characters that speak in different ways.
   (c) **[S]** Form part of a cohesive narrative.
   (d) **[S]** Motivate player progression.
   (e) **[S]** Explain any key concepts in a conversational tone.
   (f) **[W]** Distract from the educational content.
8. Additional features the game should have:

(a) **[M]** A main menu and a menu to navigate between topics.

(b) **[C]** Some counting problems.

(c) **[C]** A soundtrack.

(d) **[C]** A fun menu to navigate between topics.

9. Non-functional requirements pertaining to the whole game (for objective 4, accessibility):

(a) **[M]** Not rely on colour alone to convey information.

(b) **[M]** Be easy-to-use.

(c) **[M]** Display all text with sufficient size, spacing, and contrast to the background.

(d) **[M]** Perform adequately on any modern processor with integrated graphics[1].

(e) **[M]** Work on small laptop screens[2].

(f) **[S]** Draw at least 60 frames per second on a mid-range computer[3].

(g) **[W]** Have any bugs or technical issues.

[1] An example of a low-end processor is the AMD 3015Ce.
[2] A 13 inch display rendering at 720p.
[3] Such as an AMD Ryzen 5700U with integrated graphics.

Chapter 4

Design

Before implementing the solution, it was important to design the user interface and the problems that will be presented.

4.1 Bijections

This topic practices the technique of finding bijections between different counting problems. There are two input methods supported for bijections, with the first being graphical input.

4.1.1 Graphical Input

When finding a bijection, a common first step is to write out all the elements for small values of n and then attempt to match them up in a logical manner. The graphical input will automatically display all the elements side-by-side and allow the player to match them up by drawing lines on the screen. This is quick for small values of n although can become tedious for larger n.

While this input method struggles to validate solutions, it helps to assist the player in gaining intuition quickly without needing to write out lots of cases themselves.

The first UI design can be found in figure 4.1. This presents the two problems side-by-side and can be matched up by dragging lines between elements. However, the initial design involved a single screen that would change for different values for n. This was prototyped, then playtesting showed that it was frustrating to be unable to switch between values of n freely and view multiple at once. Therefore, the different values of n are now laid out side-by-side, where the camera is moved between values of n. Figure 4.2 shows the final design, as implemented in the game.

Validating Solutions

When validating, we want to check that the player is using some logical process to match up the elements and is not just picking randomly. However, it turns out to be mathematically impossible to validate solutions perfectly with zero knowledge of the process the user is following to match up

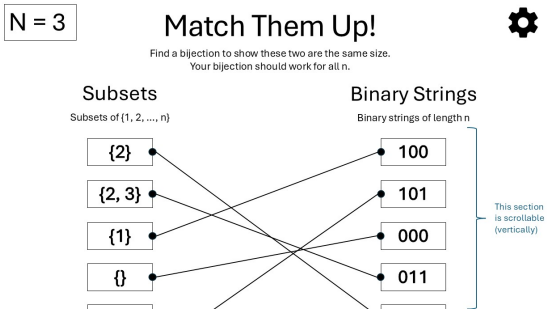


Figure 4.1: The first prototype UI design for bijections.

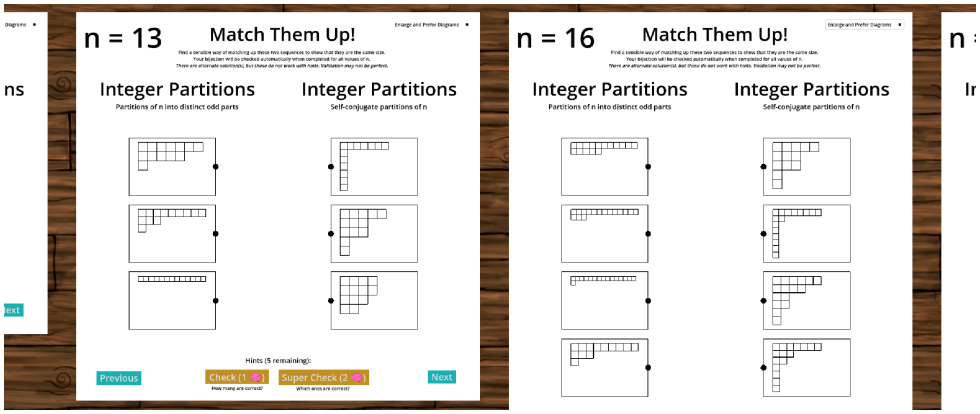


Figure 4.2: Implementation of the final UI design for bijections.

the elements. For any sequence of mappings input by the user, we can construct a logical process that they could have used to produce this solution.

Claim. *Let $N \subset \mathbb{N}$ and $(f_i : A_i \to B_i)_{i \in N}$ be mappings input by the player. Let $(g_i : A_i \to B_i)_{i \in N}$ be a 'sensible bijection'. Then, there exists a logical process $(h_i : A_i \to A_i)_{i \in N}$ whereby for every $i \in N$, $f = g \circ h$ (i.e. the player's input equals some sensible bijection and a logical process).*

Proof. Since g_i is a bijection, it has a unique inverse, namely g_i^{-1}. Then $h_i = g_i^{-1} \circ f_i$ for all i. □

This claim tells us that no matter what the user inputs, we should mark it as correct, as it is part of some valid scheme for all n. However, that misses the point of the topic. In reality, most of these bijections will not be sensible.

To store solutions, each problem is given one primary solution, and zero or more alternate solutions.

Observation 1. *Suppose $X = \{x_1, ..., x_n\}$ and $Y = \{y_1, ..., y_n\}$. Suppose $f : X \to Y$ is a sensible bijection where $f(x_i) = y_i$ for all $i \in \{1, ..., n\}$. Then $g : X \to Y$ where $g(x_i) = y_{n-i+1}$ is usually also a sensible bijection.*

While not a formal statement, this holds true for many of the levels that are implemented. Therefore, this solution generated by 'flipping' the bijection is implemented for every primary solution and every alternate solution. This gives every level at least two solutions.

As an example, consider three bijections between binary strings on $\{A, B\}$ and on $\{1, 2\}$.

Bijection 1. *Substitute A with 1 and B with 2.*

Bijection 2. *Substitute A with 2 and B with 1.*

Bijection 3. *For odd n, use bijection 1 and for even n, use bijection 2.*

Bijection 1 is the primary solution. Bijection 2 is the flipped solution. Bijection 3 is marked incorrect, as it is not a 'sensible' solution, despite following a logical construction.

Hints System

A key question was when the solutions should be validated. Instant validation for small values of n admits a brute-force approach and prevents the user from engaging with the problem fully, but waiting until all values of n are complete can lead to a frustrating experience. To combat this in a gamified manner, a hints system is introduced to guide the player towards a solution.

There are three types of hints, shown in table 4.1. Each level has 5 hints available.

Name	Per	Cost	Description
Level Hint	Level	1	A hand-written hint in words.
Check	Value of n	1	Find out how many elements are matched correctly.
Super Check	Value of n	2	Find out which elements are matched correctly.

Table 4.1: Types of hint available in the bijections topic.

The level hint is deliberately cheap for the value offered, to encourage the player to use it as a first resource. The remaining hints are easy to understand, nearly always yield useful information, and do not trivialise the problem.

A key limitation is that it does not interact with any alternate solutions. This is not easily fixed. The hints system could compare to every single alternate solution, but this gives away a lot of information and is confusing. It could just compare to one solution, but then it is not clear which solution to test against and the solution may change, causing further confusion. The best solution found was to make the hints system only interact with the primary solution.

4.1.2 Validation

While the graphical input helps in gaining intuition and coming up with a bijection, the validation of solutions is poor. There may be false negatives if the player inputs a bijection that hasn't been programmed in, and there may be false positives if the player happens to obtain a correct solution without actually following a methodical process. We will consider two alternate input methods with higher accuracy.

Quiz The quiz input method presents the player with a larger element, say one for $n = 10$, and ask them to input what their bijection would map that element to. This would be done about 3 times. This addresses the chance for false positives as it can test the player's understanding on larger versions of the problem. It also doesn't require matching up every element, which can be time-consuming. However, many of the elements are graphical in nature so getting the user to input these will be time-consuming to implement. Furthermore, this input method still relies on hardcoding the bijections, meaning it struggles with the same problem of false negatives, which is the more common issue.

Code The code input method will ask the user to input their bijection using code. Then, the computer will check its validity on some large values of n. This does not require hardcoding particular bijections, hence avoids false negatives and achieves near-perfect accuracy. However, it may be more challenging for players with no coding experience, and it relies on representing many graphical objects in code.

Criterion	Quiz	Graphical	Code
Accuracy	Medium	Low	High
Ease of Input	High	Medium	Low
Intuition	Low	High	Low
Ease of Implementation	Low	Medium	Low

Table 4.2: Advantages and disadvantages of bijection input methods.

While the focus is on presenting interesting problems over perfect validation, it is worthwhile to implement another input method to tackle the problem of validating solutions. Due to its higher accuracy, we will implement the code input method. Importantly, this will complement the existing input method rather than replacing it, as the code input does not help the user find a solution, only validate it.

4.1.3 Code Input

The code input offers near-perfect validation by asking the user to input their bijection in code, and then testing for injectivity and surjectivity on a large number of elements. This will not have any false negatives, although there is a very small chance of a false positive as it cannot check all n.

As this may be challenging for people without coding skills, it is not required to mark a level as complete. It is put at the end as a bonus challenge aimed primarily at individuals with coding experience. Most students taking combinatorics do have some experience with programming, so most students should be able to engage with this.

The user interface (figure 4.3) is designed to be easy-to-use, with most of the space dedicated to the code itself and some surrounding buttons to run the code or view documentation. The representation of the input and the output is given with the data types as this is crucial to write code. Buttons to insert blocks such as if-else and while remove the need to remember the syntax and make it more similar to block-based coding. A window is provided to test the code on particular inputs so that players do not have to run the bijection on every input to diagnose issues.

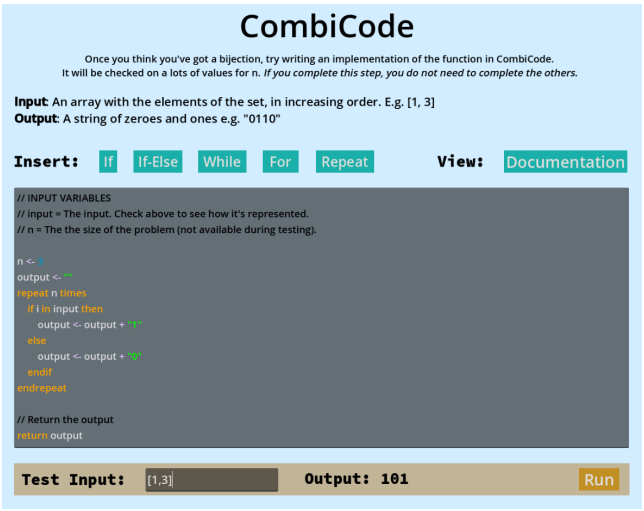


Figure 4.3: User interface for the code input method.

The Language

While the initial plan was to use a pre-existing language such as Python, this has poor compatibility across platforms and therefore is not suitable. Therefore, we will design a custom language. The language will parse control blocks such as `if` and `while` and handle assignments, but expressions will be parsed using Godot's built-in `Expression` class. This saves time during implementation and allows for the use of Godot's wide range of built-in data-types and methods. However, it can be challenging to design a parser that combines the two aspects effectively, as the parser lacks an understanding of what is in the expressions.

The language is designed to be simple and easy-to-understand, especially given a mathematical background but no background in programming. The syntax is designed to look like pseudocode.

A context-free grammar for the language in Backus-Naur Form (BNF) follows:

```
program ::= line_list
line_list ::= line_list line
           |  epsilon
line ::= stmt "\n"
stmt ::= assign
      |  expr
      |  if_stmt
      |  while_stmt
      |  for_stmt
      |  repeat_stmt
      |  return_stmt
      |  epsilon
assign ::= IDENTIFIER "<-" expr
if_stmt ::= "if" expr "then" "\n" line_list else "endif"
```

```
else ::= "else" "\n" line_list
      |  epsilon
while_stmt ::= "while" expr "do" "\n" line_list "endwhile"
for_stmt ::= "for" IDENTIFIER "in" expr "do" "\n" line_list "endfor"
repeat_stmt ::= "repeat" expr "times" "\n" line_list "endrepeat"
return_stmt ::= "return" expr
expr ::= EXPRESSION | IDENTIFIER
```

Where the terminal `IDENTIFIER` matches the regex `[_a-zA-Z][_a-zA-Z0-9]*` and `EXPRESSION` matches any string (provided that none others match).

Notable design choices for the language are:

- It does not contain functions. This is primarily to save development time. The bijections should be simple enough to implement without defining additional functions.
- The language is not sensitive to indentation. Many new programmers struggle with indentation issues, so we instead use keywords such as `endif` to end blocks (these are very common in pseudocode). Indentation is still encouraged in the interface.
- We use the symbol `<-` for assignment as it is more common in pseudocode. It also helps to identify a common error with the use of `=` as a comparison operator.
- `for` statements are included, as these are very common when defining bijections.
- We also include a `repeat` statement for count-controlled iteration that is easier to understand.
- We avoid curly braces, instead opting for words like `then` and `endif` to give a more pseudocode feel. This also makes it easier to catch common mistakes with mismatched braces.
- All variables have global scope. This is easier to understand.

Validation

Using this language, we have two steps to validate the bijection.

1. Generate all possible cases for a particular value of n.
2. Run the user's bijection on every value, and check for injectivity and closure (surjectivity follows automatically because we already know that these sets are the same size).

This requires the implementation of new methods for each level to generate the cases algorithmically for arbitrary n.

4.1.4 Level Design

The game features six levels (table 4.3). The well-structured codebase makes it easy to add more.

Level 3 introduces the Robinson-Schensted correspondence [18] [51]. This is an important bijection often taught during advanced combinatorics courses between two seemingly unrelated concepts. This level will help users to discover this bijection on their own. The $n = 2$ case is in figure 4.4.

Levels 4 and 5 cover an important topic of integer partitions and have toggleable Young diagrams. These Young diagrams are crucial to developing intuition but are time consuming to draw by hand,

Level	Description
0	Binary strings on different alphabets (tutorial)
1	Subsets and binary strings
2	Stars and bars
3	Robinson-Schensted correspondence
4	Self-conjugate vs distinct odd partitions
5	Distinct vs odd partitions

Table 4.3: Levels in the bijections topic.

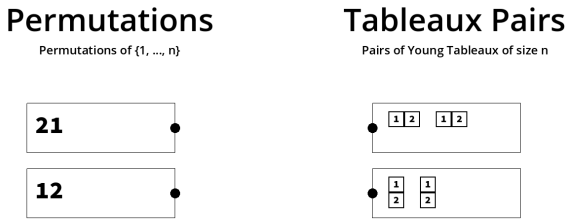


Figure 4.4: The Robinson-Schensted correspondence (level 3, n=2).

so these levels will help students significantly. Level 4 (figure 4.2) is a particularly good bijection as it easy to match some elements by observation but hard to devise a rigorous bijection.

4.2 Catalan Number Zone

The Catalan number zone extends the bijections topic by offering procedurally-generated problems counted by the Catalan numbers. It re-uses the interface from the bijections topic.

While it would be simple to fill the game with lots of very similar problems, problems are intentionally chosen that look and feel different from each other to provide more gameplay variety and deliver more educational value. Problems are chosen that are succinct to explain but offer complex and novel bijections to other problems.

Definition. *A Dyck path of length n is a walk starting at (0, 0) consisting of up steps, (1, 1), and down steps, (1, -1), finishing at (n, 0) and never going below the x-axis.*

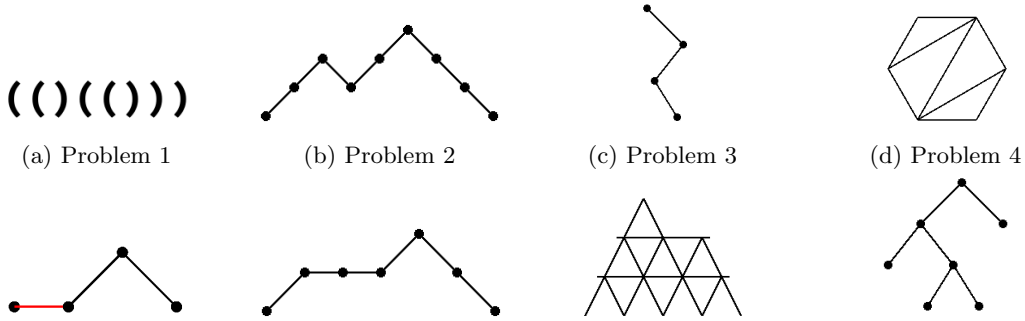
Definition. *A Motzkin path is similar to a Dyck path but it also allows flat steps, (1, 0).*

Definition. *A Schröder path is similar to a Motzkin path, but the flat steps have length 2, (2, 0) instead of (1, 0).*

Definition. *A string of parentheses is balanced if it is generated by the grammar* $S \to \underline{(}S\underline{)}S|\varepsilon$.

Definition. *A house of cards is a structure made of playing cards where the bottom level is filled with cards and each subsequent level is of length one less than the one below, arranged in a triangle pattern. Furthermore, cards may not be placed above empty spaces. See figure 4.5g.*

We will refer to a counting problem that is counted by the Catalan numbers as a Catalan problem.



(a) Problem 1 (b) Problem 2 (c) Problem 3 (d) Problem 4

(e) Problem 5 (f) Problem 6 (g) Problem 7 (h) Problem 8

Figure 4.5: Drawings of one element from each Catalan problem (drawn in-game).

A full list of the Catalan problems in the game follows, all chosen from *Catalan Numbers* by Richard P. Stanley [54]. See figure 4.5 for a drawing of each.

1. Strings of balanced parentheses of length $2n$.
2. Dyck paths of length $2n$.
3. Binary trees on n vertices, up to isomorphism.
4. Triangulations of a convex $(n+2)$-gon into n triangles, using $n-1$ diagonals that do not intersect in their interiors.
5. Motzkin paths of length $n-1$ with the flat steps coloured either red or blue.
6. Schröder paths of length $2n-2$ with no valleys.
7. Ways to form a house of cards with n pairs of cards on the bottom level.[1]
8. Complete binary trees with $2n+1$ vertices.

Figure 4.6 depicts a graph of the problems and the bijections between them. Since it is connected, it is possible to generate a bijection between any two problems. Currently, this graph is sparse. Ideally, it would contain more alternate solutions, which is an opportunity for future work.

Some original bijections were devised for this game.

Bijection 4 $(7 \to 6)$**.** *Draw a line across the top of the house of cards to form a Schröder path with no valleys.*

Proof. A house of cards differs vertically by at most one in each step. Horizontally, the paths are always of length $2n-2$, when counting flat steps as two. Furthermore, a house of cards must be filled on the bottom level, so the Schröder path does not go below the x axis. Valleys are not possible because horizontal cards are always drawn over the top, so a string of down steps will always be followed by a flat step and not an up step. □

[1] Adapted from problem 191, which instead poses this as a pile of coins

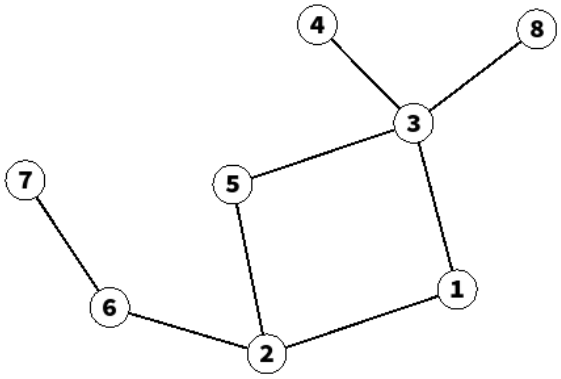


Figure 4.6: Bijections between the Catalan problems.

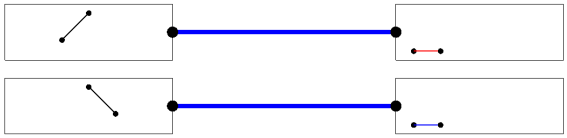


Figure 4.7: Bijection 5 for $n = 2$.

Bijection 5 ($5 \to 3$)**.** *First, we handle the flat sections. For each red flat step, add a left child and proceed from this new vertex. For each blue flat step, add a right child and proceed from this new vertex. This is a bijection between flat coloured motzkin paths and binary trees where each vertex has at most one child (see figure 4.7).*

However, we also have up and down steps. For each up step, add both a left and a right child. Identify the corresponding down step. The content before this down step defines the left child. The content afterwards defines the right child. See figure 4.8 for an example.

Proof. It is injective because each coloured step advances further down the tree, hence no conflicts to upper levels. The up steps are the only way to add both children, and the left and right child are completely separated so there is no way to generate the same tree from different paths.

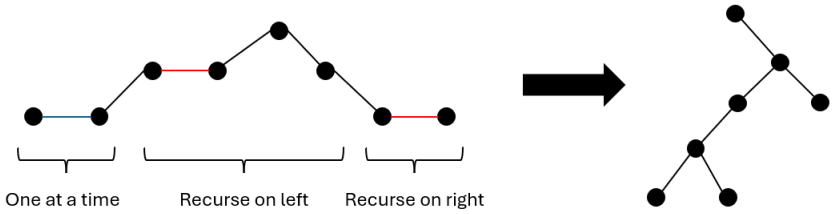


Figure 4.8: A coloured Motzkin path associated to a binary tree.

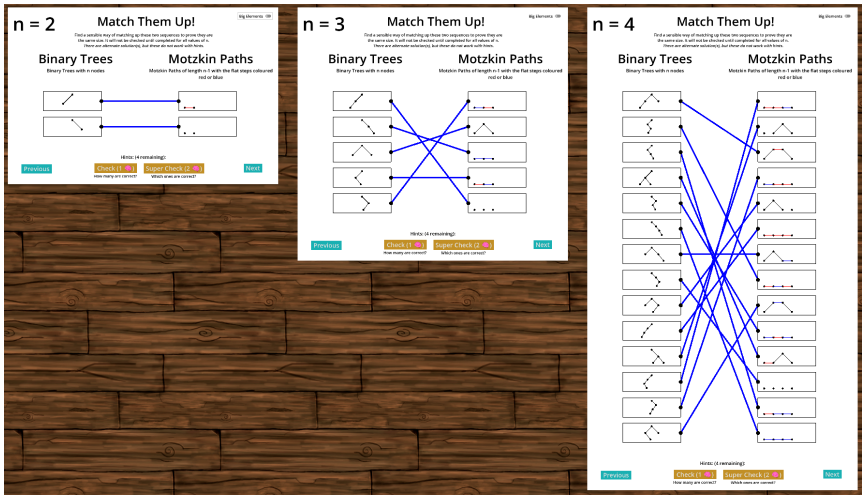


Figure 4.9: Bijection 5 done in-game.

Surjectivity can be proven by defining an inverse to map a binary tree to a Motzkin path. Upon reaching a node with a single child, add a blue or red path (depending on left/right node) and traverse to the child. Upon reaching a node with two children, add an up step. Recurse on the left child. Add a down step. Recurse on the right child. This forms a coloured Motzkin path of length $n-1$ (the Motzkin path has one edge for each edge in the binary tree). Therefore, every Motzkin path of length $n-1$ is in the image.

Therefore, this is a bijection for all n. □

This game proved very helpful in devising this bijection (see figure 4.9).

4.3 Planar Graphs

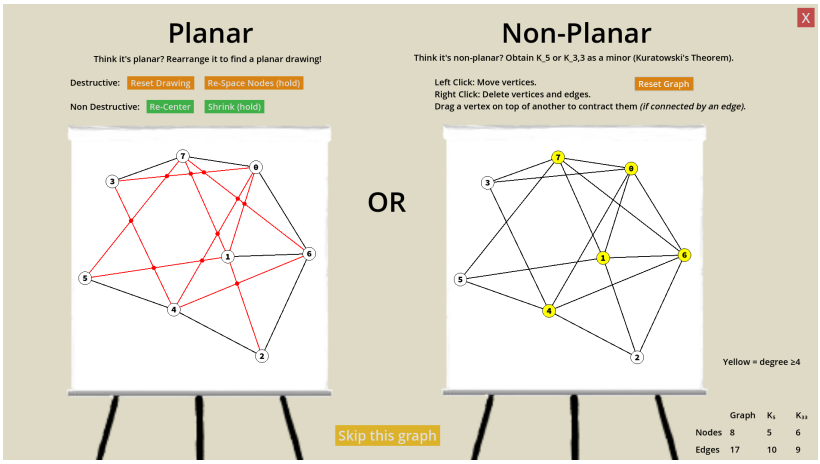
The user interface must include a part for the player to prove that a graph is planar, and a part to prove that a graph is non-planar. There are two approaches for arranging these (figure 4.10):

Side-by-Side Show both on the same screen. This is potentially confusing as only one of them needs to be completed, however it makes more efficient use of the screen space and makes it easy for the player to make an attempt at both directions. For example, when proving a graph is non-planar, it is sometimes useful to have a picture of the original graph rather than just the current minor.

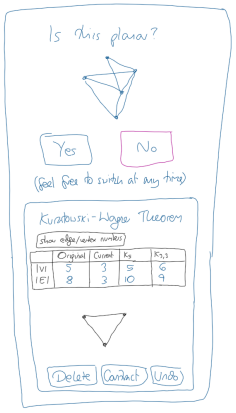
Click-to-Switch A click is required to switch between proving it is planar and non-planar, with only one of them displaying at once. This makes it clearer that only one direction is required (and possible), however it requires more clicks and does not make as efficient use of the screen space.

The final user interface is side-by-side due to the better utilisation of screen space.

On the planar side, the majority of the screen space is taken up with the graph. Vertices can be dragged by holding left click, and intersecting edges are highlighted in red. Intersection points are



(a) The final side-by-side user interface.



(b) The initial design for the click-to-switch interface.

Figure 4.10: Potential designs for the planar graphs interface.

represented with red circles to highlight them and avoid relying on colour alone to communicate the information. There are also some buttons at the top for graph manipulation, described in table 4.4. There is a clear indication of which operations are destructive to the player's progress.

Button	Description
Position Nodes Randomly	Sets all nodes to random positions.
Improve Node Spacing	While held, runs the iterative drawing algorithm.
Recentre Drawing	Expands the graph to fill its bounds.
Shrink Drawing	While held, the drawing shrinks towards the centre.

Table 4.4: Graph manipulation buttons for proving a graph is planar.

The non-planar side follows a similar design pattern to the planar side with an explanation of the controls and a reset graph button. There are two additional tools designed to help players: vertices of degree at least four are highlighted in yellow, and the number of nodes and edges in the graph is compared to that of K_5 and $K_{3,3}$. Both of these were added during playtesting as they help the player with the tedious aspects of solving the problem.

Left click is maintained to move vertices as it is consistent. Right clicking on a vertex or edge will delete it - it's important that both deletion operations use the same button to maintain consistency. Edge contractions are done by dragging neighbouring vertices on top of each other and then releasing the mouse button, as this is natural and easy to use without requiring an additional button. This was initially done when the circles were intersecting, however this was changed to be when the mouse cursor is over the target vertex based on results from playtesting.

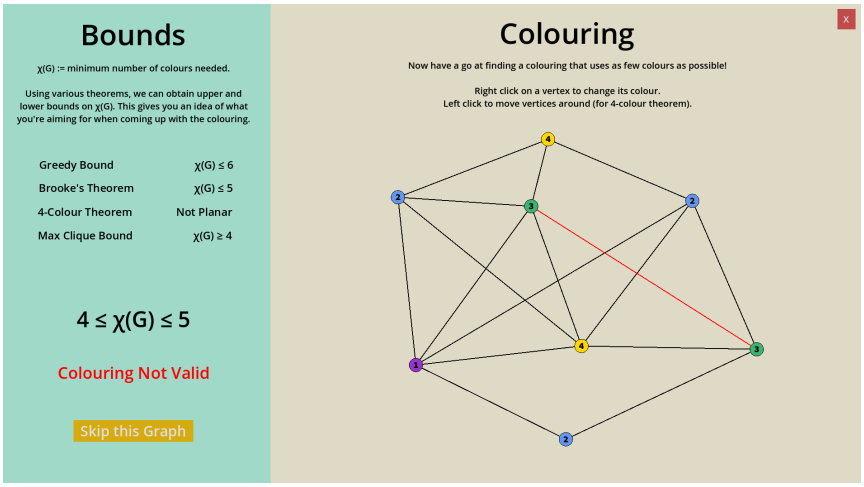


Figure 4.11: User interface for the graph colouring topic.

4.4 Graph Colourings

The user interface (figure 4.11) devotes the majority of the space to the graph, with the bounds forming a panel on the left. Changing the colour of a vertex is done by right clicking to cycle through all available colours. This could have been implemented where right clicking would bring up a separate menu to choose the colour, but it is rare that the player needs to choose a particular colour. To simply cycle through colours until none conflict, this menu is the quickest to use. In particular, it only cycles through enough colours as given by the current best upper bound. The controls are explained onscreen clearly.

Establishing bounds is done using four different theorems that are depicted in table 4.5. These were chosen as they are commonly used and easy to understand. Applying the four colour theorem requires the user to rearrange the graph into planar form before they can discover this bound, providing a useful link between the two topics. These theorems add significant value beyond merely finding a colouring.

Theorem	Bound	Description
Greedy Bound	$\chi(G) \leq \Delta + 1$	A greedy algorithm can trivially colour a graph.
Brooke's Theorem	$\chi(G) \leq \Delta$	Does not apply for complete graphs or odd cycles.
4-Colour Theorem	$\chi(G) \leq 4$	Only applies to planar graphs.
Max Clique Bound	$\chi(G) \geq k$	k = size of maximum clique.

Table 4.5: Theorems used to bound the chromatic number $\chi(G)$.

Figure 4.12: The main menu.

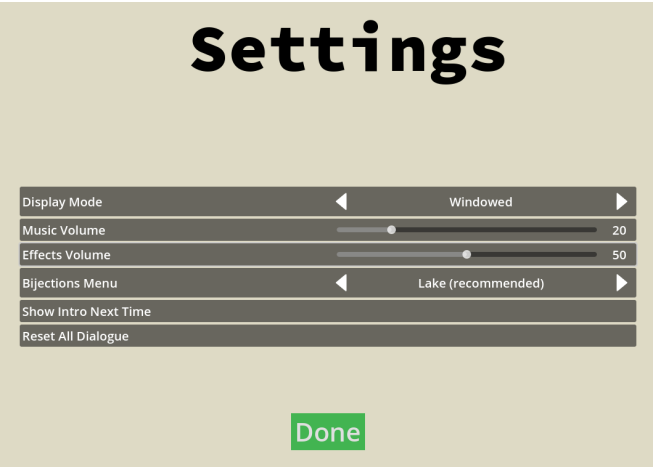


Figure 4.13: The settings page.

4.5 Navigation & Menus

The main menu (figure 4.12) has three buttons: Play, Settings and Quit. These are arranged to look like a traffic light, keeping in theme with the game.

The settings page (figure 4.13) has options for:

- A toggle between fullscreen and windowed mode.
- Separate volume sliders for music and sound effects (necessary for accessibility requirements).
- A toggle for an alternative bijections menu that is accessed using a mouse.
- A button to show the intro sequence again.
- A button to reset all encountered dialogue (requires another click to confirm).

These are all saved when the player quits and re-launches the game.

The world is the menu used to navigate between topics (figure 4.14). This features a hut for each of the topics and some little signs for each of the counting problems. This gives an interesting way to add lots of possible elements that isn't simply a list of buttons. The topics are connected with roads and the player may drive between them in a car. This is fun, offering a little break in between complex topics although doesn't take much time away from it. The background is hand-drawn and the car has no collision with the environment, reinforcing the idea that it is a simple menu designed to put the player in a particular mood, over anything complicated and serious.

While most players found the car increased their enjoyment of the game, some players found the car difficult or inconvenient and would've preferred to navigate the world in a simpler way. Therefore, a teleportation menu (figure 4.15) is provided as an accessibility option to skip the driving. This makes the entire game playable without any keyboard input. It also skips any dialogue upon entering the topics, skipping straight to the topics themselves.

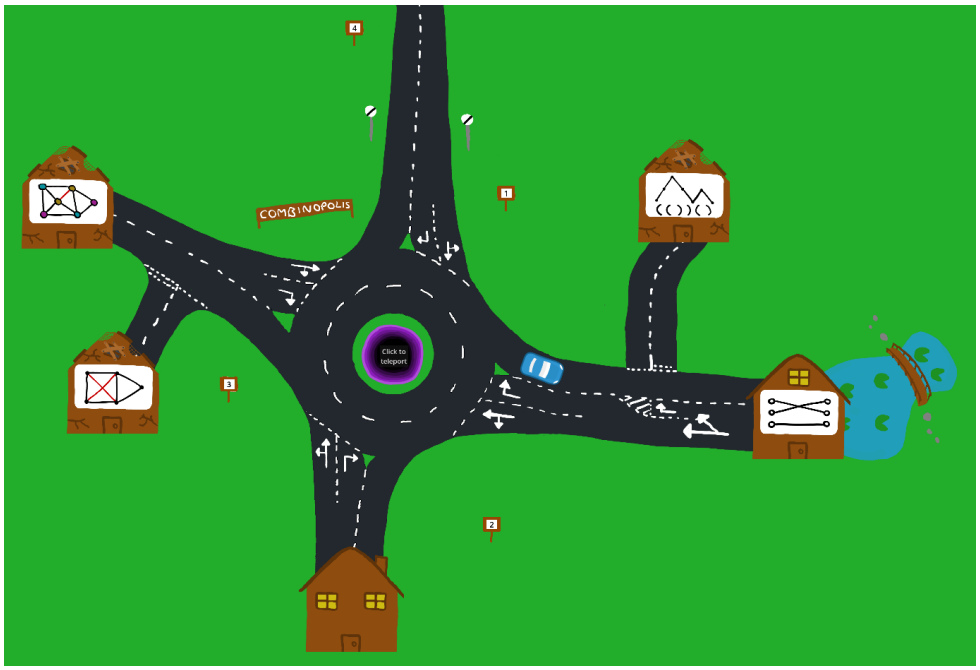


Figure 4.14: The world to navigate between topics.

Figure 4.15: Teleportation menu, as an alternative to the driving menu.

The world also facilitates the counting problems topic described in section 4.7.

4.6 Narrative & Dialogue

The game features dialogue to guide players through the game and explain key concepts in an immersive way. These come together to form a simple narrative will make the game more fun and motivate players to learn more.

4.6.1 Narrative

A summary of the narrative is as follows: *Combinopolis is a town set up a while ago devoted to the study of combinatorics that was once thriving. However, there haven't been any visitors recently and the professors have forgotten how to solve their puzzles. As a result, the town has fallen into disrepair, and the player needs to travel around the town to solve the puzzles and breathe new life into the town.*

A key advantage of this is the visual indication of progression offered by repairing huts (in figure 4.14, bijections has been completed and so the hut is repaired, but the other topics have not).

This narrative will benefit the game because:

- Players will be more motivated to progress, and therefore will learn more (objective 2).
- Players will have more fun playing (objective 3).

Studies show that distributed narrative, intrinsically integrated fantasies, empathetic characters, and adaptiveness or responsivity are four characteristics of game narratives found to be effective in educational games [46]. The narrative aims for all of these, however will not be devoted a significant portion of time as it is not the focus of the project.

4.6.2 Dialogue

The dialogue is written in a light-hearted tone that responds to the actions of the player, including empathetic characters and responsivity. Most scenes give the options for serious and fun responses, and the content and length of the following dialogue is adjusted based on the player's response.

The dialogue at the beginning of each topic is also responsible for explaining the premise of the topic and defining any important terms. This is often interactive, responding to how much the player already knows and keeping them engaged. For example, consider the Planar Graphs topic. The options given are "What's a graph?", "What's a planar graph?", "What's the Kuratowski-Wagner theorem?", "Let's jump straight in" and "Actually, I don't feel like planar graphs right now.". The explanation of the Kuratowski-Wagner theorem is long, but is broken up by giving the player an opportunity to interject with "I know this already" or asking them which explanation they'd like first, keeping them engaged so it doesn't feel like they are just reading text.

A key requirement is for the dialogue to avoid distracting from the educational value. This is done by keeping the dialogue short, so it doesn't take up very much time.

The dialogue was prototyped in Twine [7]. The opening scene can be found in figure 4.16.

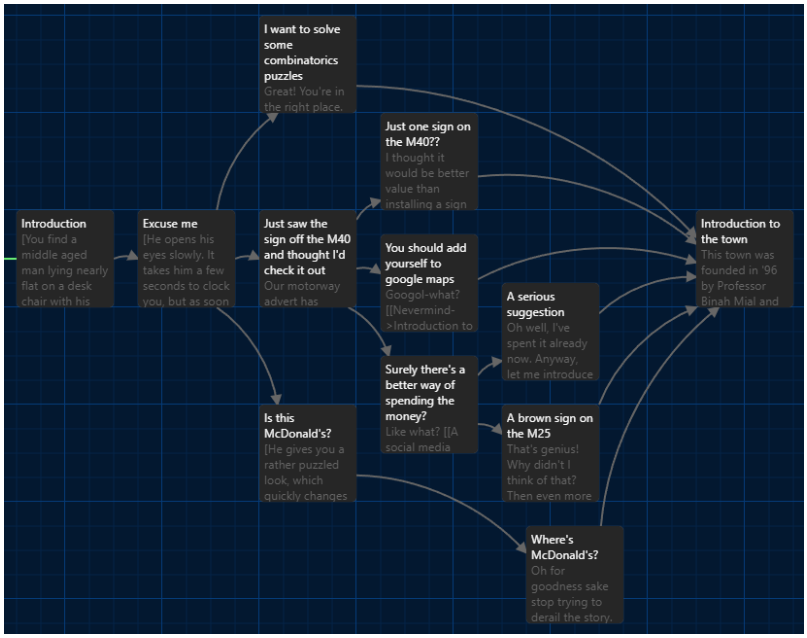


Figure 4.16: The opening scene prototyped in Twine.

4.7 Counting Problems

The goal of the topic is to give the player a space to engage with counting problems, preferably adding value beyond simply doing worksheets. This has gone through many different versions.

4.7.1 A Topic

The initial plan was for this to be a level-based topic, presenting players with one counting problem per level. The player would type in the solution and the game would be able to offer hints or point out common mistakes if the player types in certain numbers. However, the implementation would've largely consisted of hard-coding values, and the game doesn't offer very much value over a worksheet, so the topic was not implemented in this form.

4.7.2 A Meta-Topic

Upon implementing the bijections topic, it became clear that this could be combined with other combinatorial techniques to form a larger proof. Often, students need to practise synthesising different techniques rather than just each technique on its own. Therefore, this topic was going to be implemented as a topic that combines lots of other topics. This was intended to occupy a significant portion of term 2.

The overall approach is inspired by block coding such as Scratch [8] and ProofBlocks [4], whereby the player drags in different blocks corresponding to different techniques. Each technique would either transform the problem into a new problem, or generate an answer from the problem. Therefore,

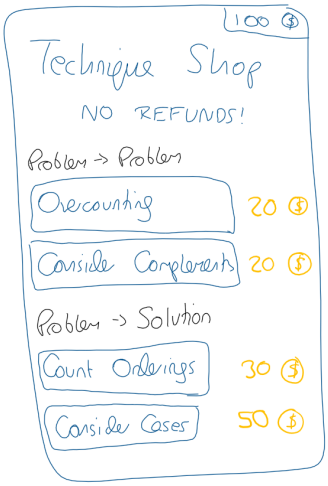


Figure 4.17: The technique shop, with coin costs for each technique.

each solution would consist of finitely many *problem to problem* blocks and then exactly one *problem to solution* block.

Akin to Super Mario 64 [9], each level will have multiple solutions and replaying the same level multiple times is encouraged to find all possible solutions. This will give students a more thorough understanding of the techniques, and solving the same problem in multiple ways is often a very satisfying part of mathematics. To avoid the player brute forcing a solution, techniques are given a coin cost which is spent as soon as they are dragged out of the shop, and cannot be recovered if the player later decides to not use that particular technique (see figure 4.17). Completing a level with more remaining coins is better.

The final step in the design process is to formalise the layout of a block. The initial designs did not distinguish sub-problems to be taken further in the proof (e.g. when considering multiple cases), required problems to demonstrate understanding of the technique (e.g. bijections). The final layout (figure 4.18) separates these clearly so the player knows exactly what each part is for in every technique, and to provide standardisation needed to simplify the implementation.

Unfortunately, there is no easy way to implement this topic without hard-coding lots of problems. Each counting problem is unique from each other, and there are rarely any ways to encode how a particular technique could work on a general problem. Therefore, the implementation would have taken significant time while contributing very little technical complexity, so the topic was not implemented in this manner. However, an implementation of this design would have significant value for students so is an opportunity for future work.

4.7.3 In-World

The final implementation is not as a topic, but instead as bonus exercises located across the world. When selected, a pop-up is opened with a counting problem (figure 4.19). This is most similar to the first approach, although as instead of being a topic, it serves as something to do when having a

break between other problems and encourages exploration around the world. This can gently nudge players to solve problems and come back to ones they've seen before without forcing them to. While the game will check that the answer is correct, there are no hints or block-based approaches so it is quick to implement.

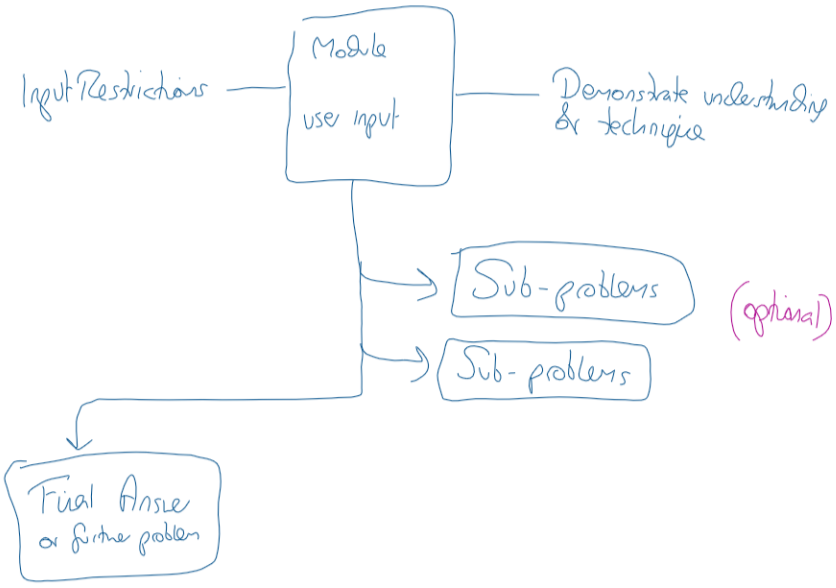


Figure 4.18: Block syntax for counting problems meta-topic.

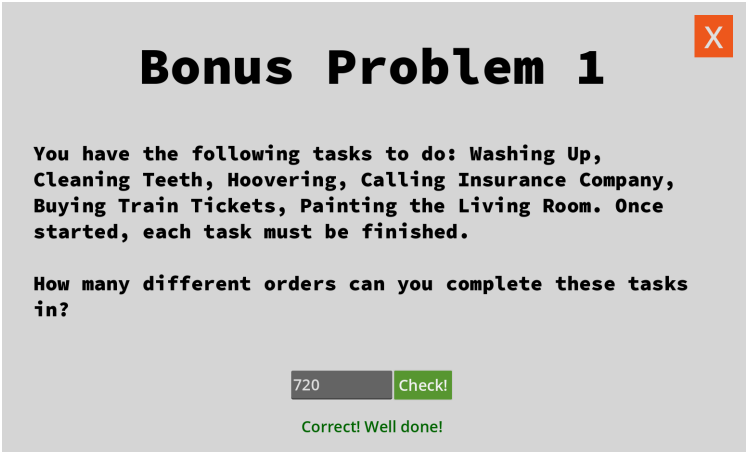


Figure 4.19: The user interface for counting problems, shown with counting problem 1.

Chapter 5

Implementation

Following from the design, this chapter outlines the implementation of the solution.

5.1 Methods and Tools

The game is developed using Godot [10], an open-source engine that is widely used for 2D game development. The game engine is notably lightweight, helping the game to run efficiently on low end hardware. The engine facilitates exporting to Windows, macOS, Linux and Web (plus Android and iOS, although these will not be used), so it contains all the export options that are needed. This makes Godot a suitable game engine for the project.

Godot uses a proprietary language called GDScript, which is object-oriented. This makes it well-suited to a game development project where many parts of the solution can naturally be represented with objects. The main limitation of object-oriented programming, space, is not of concern. As an example, the graphs system is made using classes which makes it very easy for the planar graphs and graph colouring topics to inherit from one graphs system.

Godot also features a robust drawing system whereby each node has a `_draw()` function, allowing easy writing of scripts to draw complex objects. This has been crucial in the development of bijections and the Catalan number zone.

5.2 Bijections

The implementation consists of finding a suitable data structure to store the levels, and then implementing the interface to use the levels.

5.2.1 Organisation & Data Structures

We first define a class for a level, e.g. Subsets and Binary Strings. We have fields for titles, descriptions, a proof and a level hint. The key field is `bijections` which stores a dictionary of type `{int: Bijection}`. A `Bijection` contains all of the information for one particular value of n. In

particular, `from` and `to` contain an array of `BijectionElement`s of the LHS and RHS respectively, and `problem_size` contains the value of n. Table 5.1 summarises the important classes.

Class	Description
BijectionLevel	Two counting problems, e.g. subsets and binary strings.
Bijection	A particular value of n within a level.
BijectionElement	An element in a bijection, e.g. 0110 or {2,3}.

Table 5.1: Classes used in the bijections topic.

A `BijectionElement` stores the data for an element but is also a game object that can be assigned to a position in the interface. This has many superclasses such as `IntegerPartitionElement` which can show diagrams by overriding `draw_contents_diagrams()`. All elements have a `text` field, which we insist upon having even if for elements that are only shown diagrammatically, as it is occasionally used[1].

We also need to decide how to store and validate the solutions. The approach that follows is simple, and there is no need for anything more complicated.

Primary Solution Each `BijectionElement` has an `id`. Every element on the LHS has a field `match` to store which element it is matched to. A bijection is correct if for every `element` on the LHS, `element.id == element.match.id`.

Alternate Solutions The `BijectionLevel` has a field `alternate_solutions` containing the number of alternate solutions. Each `BijectionElement` on the LHS has an array with this many elements, each index corresponds to an alternate solution. A bijection is correct if for every `element` on the LHS, there exists an `alternate_solution` such that `element[alternate_solution] == element.match.id`.

Reversed Solutions Observation 1 is implemented by also considering `reverse=true` for all of the above, replacing an id with `total_elements - OLD_ID + 1` if so.

Importantly, these alternate solutions must match up between all values of n. If the solution is correct using the primary solution for even n, and alternate solution 1 for odd n, then this must be marked incorrect for all n. The method `check(alternate_solution, reverse)` method takes in two parameters, and the level will consider each possible solution and then check it against all values of n.

All level state (except for that which is specific to one value of n) is managed in the level. For example, "big elements" is a toggle that has one state for the whole level. When toggling "big elements", each `Bijection` exposes a method to listen to changes from other values of n and has its own method to send updates to the `BijectionLevel`. This makes it clear which class is responsible for maintaining the state of which variable, avoiding pitfalls of object-oriented programming.

5.2.2 User Interface

The user interface was mostly simple to implement. There is some complexity in managing the large number of classes (each level, each value of n, the script containing the matching logic, the bijection

[1]For example, when displaying the results of hints in the Hints & Definitions section.

itself) and passing the data between them, being very careful about only having one particular class being responsible for each value.

Major re-structuring of the code was required when switching from a one-page design to showing all values for n in the scene at once. Furthermore, drawing diagrams required re-structuring. The elements used to be rendered in the same draw function as the matched lines, but this made drawing different objects difficult. With separate classes, it became easy to override the draw method to draw different diagrams. `IntegerPartitionElement` included an algorithm to calculate the maximum possible size of the squares (without leaving bounds) and then generate the position of the squares to draw a Young diagram [32].

Many changes were made based on playtesting. For example, linear interpolation (lerp) was added to the camera on keyboard input which significantly improved the experience, and the appearance of the hints system was made far clearer.

5.2.3 The Parser

The parser implementation was far more complex. The initial approach included a lexer, however this was quickly realised to be hindering the parsing rather than helping it. This is because expressions are not parsed by the program, so it would be redundant to divide it up into tokens and then re-create the original string afterwards. The final implementation uses a recursive descent parser to build an abstract syntax tree (AST) directly from the input string. This is a suitable choice as it is more natural to write and often facilitates more useful error messages, which is especially important to make the language easy-to-use.

The grammar is also relaxed here, to only use new lines when they are helpful and to ignore them when they are not. For example, the following piece of code is valid, despite lacking some new lines. If the parser is not expecting a new line, it will be ignored. Note that the line break is still required to separate the two expressions in the `else` block.

```
if var == 1 then var <- var + 1 else var <- var + 2
var <- var - 1 endif
```

We also sometimes permit keywords that are used incorrectly, and treat them as part of an expression instead of flagging an error. These relaxations make the language easier and more flexible for the player to write code in, while still encouraging them to write well formatted and indented code.

The parser has one class for each node in the AST. Each has a `parse` method taking in the full code as a String and the current position the parser has reached, and nodes have an `end_pos` which is used to update the position after parsing. The parse methods use the FIRST and FOLLOW sets in tables 5.2 and 5.3 to choose the right production. Much of the implementation relies on the built-in `find()` method to find the next occurrence in the string, and then parsing each part appropriately.

The mixture of expressions, which are not parsed, and keywords, which are, made it complex to implement. For example, when parsing a `repeat` loop, we use `find("times", pos)` and then parse the text in-between as an expression and the following text as a block. This block is given type `repeat` (so it knows to expect `endrepeat` and not `endfor`, for example). This approach needs to conform to the grammar and display useful error messages when it does not.

Errors are done with a special AST node. This is because Godot does not support error handling,

A	FIRST(A)
program	FIRST(stmt)
line_list	FIRST(stmt) $\cup$ {ε}
line	FIRST(stmt)
assign	{IDENTIFIER}
if_stmt	{"if"}
else	{"else", ε}
while_stmt	{"while"}
for_stmt	{"for"}
repeat_stmt	{"repeat"}
return_stmt	{"return"}
expr	{EXPRESSION, IDENTIFIER}
stmt	{"if", "while", "for", "repeat", "return", EXPRESSION, IDENTIFIER, ε}

Table 5.2: FIRST sets for all non-terminals.

A	FOLLOW(A)
line_list	{EOF, "endif", "else", "endwhile", "endfor", "endrepeat"}
else	{"endif"}
stmt	{"\n"}

Table 5.3: FOLLOW sets for nullable non-terminals.

and so return values must be used to track errors. Error ASTs also have a `start_pos`, so that when displaying an error, the relevant portion of code can be highlighted to the user in the input field.

Once the AST is formed, executing the code is easy. Each node has an `execute()` method which takes in a `Dictionary` of `variables` and outputs the new variables. The full list of variables is passed into every expression that is being executed by Godot. The special values in table 5.4 all use % signs so they are not valid identifiers and therefore cannot be accessed by the programmer.

Variable	Description
%return%	The return value (if present, execution must terminate immediately).
%result%	The result of the expression that was last executed.
%error%	An error message (if present, execution must terminate immediately).
%error_pos%	Start position (character in source code) of the error.
%error_pos_end%	End position (character in source code) of the error.

Table 5.4: Special variables used to transfer information during execution of the parser.

The user interface for the code input uses the built-in `CodeEdit` node which supports syntax highlighting, auto-indent and more features designed for editing code.

5.2.4 The Levels

After being designed, the implementation of the levels largely consisted of data entry. The most complex part was making a function that generates all the elements for a particular value of n. This was necessary for both sides of the bijection for every single level. As an example, we will consider self-conjugate integer partitions.

We first generate all integer partitions of n as in algorithm 1.

Algorithm 1 Algorithm to generate all integer partitions of n with no part greater than max_part.

```
function GENERATE_PARTITIONS(n, max_part)
    if n = 0 then
        return [[]]
    end if
    partitions ← []
    for largest_part = min(n, max_part) to 0 do
        for sub_partition in GENERATE_PARTITIONS(n - largest_part, largest_part) do
            partitions.append([largest_part] + sub_partition)
        end for
    end for
    return partitions
end function
```

Claim. *Algorithm 1 generates all partitions of n with no part greater than max_part.*

Proof. Let $P = [p_0, ..., p_k]$ be a partition of n in descending order with $p_0 \leq$ max_part. The first for loop ranges from 0 to max_part. Since $0 \leq p_0 \leq$ max_part, the loop runs with p_0 as the largest_part. We place the largest part at the beginning, and recurse on the rest which follows from induction. Therefore, P is generated by algorithm 1. □

After generating all partitions, we need to filter out those that are not self-conjugate. This is challenging because the notion of being self-conjugate is inherently graphical. Fortunately, some further research led to a paper that presents a method of checking which does not require any interaction with Young diagrams and is suitable for implementation [47]. Following the implementation of this algorithm, the partitions are filtered so that only the self-conjugate partitions remain.

5.3 Catalan Number Zone

For the Catalan number zone, we define a class `CatalanProblem` containing the information for one side of a level. Two `CatalanProblem`s may be combined together to create one `BijectionLevel`.

5.3.1 Storing Bijections

Bijections were planned to be stored similarly to the bijections topic, whereby all elements would have an `id` which would define a bijection. However, using the same validation approach is limiting for the Catalan number zone because:

- Catalan problems often use complex representations, meaning it is time consuming to write out all 23 cases for each problem.
- As more alternate solutions are introduced, the number of alternate solutions that need to be added to each problem grows considerably. This is both time consuming (as lots of alternate solutions have to be added manually) and reduces maintainability (as each problem will have a long list of integers after it).
- The list of integers does not tell us which alternate solutions make it up. In particular, the proofs are detached from the alternate solutions. This makes the code less maintainable.

Instead, a new approach was developed:

1. Arbitrarily choose a base problem, and arbitrarily assign `id`s to each element (same as before).
2. For each problem, add an adjacency list to each other problem with a known bijection. Write each bijection as a function to transform one into the other.
3. To generate the cases for a problem that isn't the base problem, use a shortest path algorithm to the base problem and then compute all the bijections along this path for each element in the base problem.
4. When validating solutions, use a breadth-first search to find all paths between the two problems - these constitute all the possible solutions.

This approach is better because:

- It is much quicker and easier to introduce new bijections.
- It is easy to display all possible proofs to the player
- It is much less prone to errors in the implementation - IDs could be mistyped, but writing a function to convert from one element to another is far more robust.
- It is easier to extend to other input methods, such as the quiz.
- It is easy to extend the solution to add a notion of difficulty to each proof, should this be desired in future.

We will choose balanced parentheses to be the base problem. This is because it is closely related to Dyck paths, arguably the most common Catalan problem, and offer a compact representation.

We will refer to Dyck paths and balanced parentheses interchangeably. Recall the trivial bijection between these two problems.

Bijection 6 (Dyck paths to balanced parentheses)**.** *Each up step in the Dyck path maps to an opening parenthesis and each down step maps to a closing parenthesis.*

We will refer to any other problem as a 'derived' Catalan problem.

5.3.2 Ordering Dyck Paths

While the orders may be assigned arbitrarily, writing them down in some sensible order may help us to spot patterns and bijections later on. This project contributes a new total order on Dyck paths, capturing some of the underlying structure of the problem.

The algorithm sorts on three criteria in this order:

1. By height of the tallest peak, which we will define as H.
2. Number of peaks of height H, then number of peaks of height $H-1$ and so on until the number of peaks of height 1.
3. Position of the leftmost peak of height H, then the position of the leftmost peak of height $H-1$ and so on until the leftmost peak of height 1 (further left = larger).

Claim. *This algorithm forms a total order on Dyck paths of length n.*

Proof. Reflexivity, transitivity and comparability are trivial. To prove it is antisymmetric, notice that the heights and positions of the peaks uniquely define a Dyck path. □

The final ordering up to $n = 4$ is as follows (the height of the tallest peak is indicated in blue):

n	Sequences in order (descending)	C_n
1	1: ()	1
2	2: (()), 1: ()	2
3	3: ((())), 2: (()()), (())(), ()(()), 1: ()()()	5
4	4: (((()))), 3: ((()())), ((())()), (()(())), ((()))(), ()((())), 2: (()()()), (())()(), ()(()()), (())(()), (())()(), ()(())(), ()()(()), 1: ()()()()	14

Table 5.5: Ordering for Dyck paths.

5.3.3 Representing and Drawing Elements

As many of the elements in the Catalan number zone are graphical, it can be challenging to devise suitable representations. As these are user-facing during the code input method, they should be concise, easy to understand, and close to the drawing (e.g. we could represent all of them with parentheses and merely adjust the drawing algorithm, but this does not support code input).

Motzkin paths and Schröder paths both inherit from Dyck paths, so they can use one drawing algorithm. A Dyck path stores a list of integers 1 or -1, and the superclasses have an additional field which is translated into Dyck path steps during initialisation. The drawing algorithm is found in algorithm 2. This algorithm determines the step length that would fill the width and the step length that would fill the height, then takes the minimum to ensure it draws within bounds. We then iterate over the steps and draw the lines and circles representing the path. `get_colour()` is overridden for drawing coloured Motzkin paths, but always returns black for Dyck paths.

For binary trees, each node could be represented as an array of length two, where each value is either a 0 or another array. A 0 indicates that this child does not exist, whereas an array represents the child. For example, figure 4.5c is represented as [0, [[0, [0, 0]], 0]]. This is compact, and ensures the left/right children are not mixed up when only one child is present. Drawing these follows a similar approach to Dyck paths to get the length of each edge, and then draws the nodes and edges recursively, using a horizontal shrink factor to ensure that nodes in different branches do not overlap further down the tree.

Algorithm 2 Algorithm to draw a Dyck path (also used for Motzkin and Schröder paths).

```
available ← size - padding * 2
step_length ← min(available.x / length, available.y / tallest_peak, max_size)
x ← size.y - padding.y
y ← padding.x
draw_circle((x, y))
for step in steps do
    new_x = x + step_length
    new_y = y + step_length * step * -1
    draw_line((x, y), (new_x, new_y), get_colour())
    x ← new_x
    y ← new_y
    draw_circle((x, y))
end for
```

Houses of cards are represented as an array of layers, with each layer being an array of 0s and 1s corresponding to whether that position is filled. As an example, figure 4.5g is represented as [[1,1,1,1], [1,1,1], [1,0], [0]]. This was chosen to be as close as possible to the drawing, although there is redundancy in the representation. After determining an appropriate square_size using a similar approach to before, they are drawn as in algorithm 3. This algorithm goes through each position and draws two slanted lines representing the vertical cards and then a horizontal line representing the cards laid on top. However, these horizontal cards are drawn between two adjacent peaks on the same level, so it requires checking that the position to the left is filled.

Algorithm 3 Algorithm to draw a house of cards (following calculation of square_size).

```
for i = 1 to levels.length do
    for j = 1 to levels[i].length do
        if levels[i][j] = 1 then
            x ← padding.x + j * square_size + i * (square_size / 2)
            y ← padding.y + available.y - i * square_size
            draw_line((x, y + square_size), (x + square_size / 2, y))
            draw_line((x + square_size, y + square_size), (x + square_size / 2, y))
            if j > 0 and levels[i][j-1] = 1 then
                draw_line((x + square_size * (1/2 + extra), y), (x - square_size * (1/2 + extra), y))
            end if
        end if
    end for
end for
```

5.3.4 Implementing the Bijections

All Catalan problems have a field `bijections` storing any known bijections to other problems in the form `{id: bijection}`. Each bijection is a `CatalanBijection`. The most important field is the mapping, which is a lambda function transforming one element into another. Regarding code structure, the representation shown to the player in the code input is decoupled from the

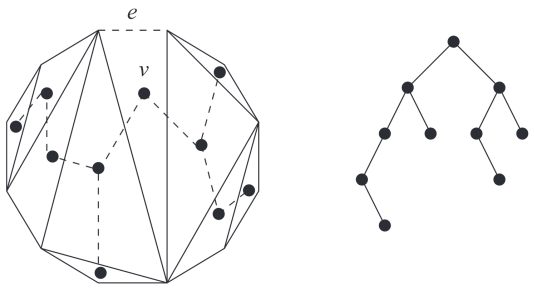


Figure 5.1: A triangulation matched to a tree [54].

representation used to implement the bijections internally so that the user-displayed representation can be easily changed later without affecting existing bijections.

The implementation of the bijections has been more complex than expected. This is because while many are present in literature, they are often described in plain English rather than with an algorithmic implementation. The project contributes some implementations for these bijections.

The most complex was the bijection from problem 3 to problem 4, best explained in figure 5.1.

Bijection 7 ($3 \to 4$ [54])**.** *Fix an edge e. Add a node inside each triangle, and let the one in the triangle containing e be the root. Traverse adjacent triangles, starting at the root, adding a left edge when adjacent to the left edge of the triangle and right edge when adjacent to the right edge of the triangle (when viewed with the edge we entered into at the bottom). This forms a binary tree.*

It is not immediately obvious how to implement such an approach from a binary tree to a triangulation. As a starting point, we need to determine the opposite vertex so that we can draw one triangle and then recurse on the two halves. This vertex depends on how many vertices are in the left subtree. Since we will need this fact for subsequent recursions, we begin by computing an in-order traversal of the vertices. Then, we can draw the triangle and recurse on two halves.

Algorithm 4 Bijection from binary trees to triangulations.

```
φ(v) ← the order of the vertices in an in-order traversal.
function INNER(root, base_edge)
    if root = null then
        return [base_edge]
    end if
    left ← INNER(root.left, [base_edge[0], φ(root)+2])
    right ← INNER(root.right, [φ(root)+2, base_edge[1]])
    return left + right + [base_edge]
end function
return INNER(tree, [2, 1])
```

There is a lot of complexity with regard to the order of the edges. It's very important that the left and right and considered carefully, which involved flipping the first edge to be $[2, 1]$ instead of $[1, 2]$.

The bijection from Schröder paths to house of cards was also complex, requiring traversing through all of the steps in the path and filling in the path and all of the space below it.

5.4 Planar Graphs

Implementing the topic can be divided into two stages: building a graph system, and using this to build the topic itself.

5.4.1 Data Structures

The first task is to store graphs. An adjacency list will be used as it gives the best balance of space complexity, time complexity, and compactness of implementation.

The first approach uses classes for vertices, storing the positions of vertices with the vertices themselves.

```
class Vertex:
    int id
    Vertex[] neighbours

class Graph:
    Vertex[] vertices
```

```
class PositionedVertex(Vertex):
    // inherits from Vertex
    Vector2 position

class GraphDrawing:
    PositionedVertex[] vertices
```

However, it has two main drawbacks.

Compactness of Representation Due to the references between vertices, it is difficult to write down a graph in a compact form. Instead, it requires multiple lines of code to initialise the vertices and set up the references.

Inheritance for Graph Drawings It is not possible to have `GraphDrawing` inherit from `Graph`, which is important so we can use the graph methods (such as `delete_vertex`) on a graph drawing.

To solve this, we instead represent vertices using integers, and store the neighbours of each vertex in a dictionary.

```
class Graph:
    Dictionary vertex_neighbours
```

```
class GraphDrawing(Graph):
    Dictionary vertex_positions
```

Example. `vertex_neighbours` *of* $K_{2,2} = \{0 : [2, 3], 1 : [2, 3], 2 : [0, 1], 3 : [0, 1]\}$.

This representation is more compact and allows for the inheritance, although the neighbours and positions are separate from each other. For example, the data structure cannot enforce that every vertex has a position. Note that positions are stored as vectors between $(0, 0)$ and $(1, 1)$, which are later converted into world co-ordinates.

We also add a `RearrangeableGraphDrawing` class inheriting from `GraphDrawing` with methods such as `is_hovering_over_vertex`, `mouse_moved`, `left_mouse_clicked`, and `draw_rearrangeable` to allow the player to rearrange the graph.

5.4.2 Generating Graphs

We need to generate random graphs on a fixed set of vertices by introducing edges at random.

Observe that graphs should be connected. If graphs aren't connected, the player may simply treat each of the connected components independently, and they have a much easier problem. In particular, for this topic they may ignore any connected component with fewer than 5 vertices (by the Kuratowski-Wagner theorem). Therefore, we will only generate connected graphs.

To do this, we will first generate a random spanning tree. This is easily done with algorithm 5, which repeatedly connects a random disconnected vertex to a random vertex in the tree.

Algorithm 5 An algorithm to randomly generate a spanning tree on a graph $(V, V \times V)$.

Let $v_0 \in V$. Let $S \leftarrow \{v_0\}$.
for $v \in V - \{v_0\}$ **do**
    Let u be a random vertex in S.
    Let $E \leftarrow E \cup (u, v)$. Let $S \leftarrow S \cup \{v\}$.
end for
return (V, E)

Finally, we fill in each additional edge with the user-customisable probability `edge_chance`.

5.4.3 Drawing Graphs

After generating a graph, it is important to generate a good drawing of it. A drawing is simply an assignment of positions to vertices. Note that we will always draw edges as straight lines (this does not affect the notion of planarity - see Fáry's theorem [27]). However, the question of what makes a good drawing varies upon the use case.

Observation 2. *An algorithm that tends to minimise the number of edge crossings is not suitable.*

Proof. This would trivialise the task of determining the planarity by observation, as well as the task of proving planarity (since planar graphs would always be drawn in a planar form). □

We have to be careful of this when choosing an algorithm. With this in mind, the algorithm used is a variant of the Fruchterman-Reingold algorithm [29]. The algorithm is iterative, repeatedly applying two forces until convergence:

- Edges pull their endpoints together (proportional to the square of the distance).
- Nodes repel each other (proportional to the distance).

When the parameters are configured well, this naturally leads to good graph drawings that are well spaced but have neighbours close together. Furthermore, it rarely leads to planar drawings, due to the tendency for cliques to clump together.

However, a third stage was added for this game:

- The boundary repels nearby nodes, and has a harsh cut-off at the boundary.

In a game development setting, it is crucial that the graph always draws within the bounds allocated to it. This third criteria guarantees that this will always happen.

The implementation consisted of applying these stages and adjusting the constants.

5.4.4 Detecting Planar Drawings

To obtain the intersection point between two edges (u, v) and (w, z), we:

1. Represent the edge (u, v) as a parametric equation in t, where $t \in [0, 1]$ constitutes points in the edge. Do likewise with (w, z) in r.
2. Solve these simultaneously to get:
$$t = \frac{(u_x - w_x)(w_y - z_y) - (u_y - w_y)(w_x - z_x)}{(u_x - v_x)(w_y - z_y) - (u_y - v_y)(w_x - z_x)}$$
and similar for r.
3. Check that $t, r \in (0, 1)$[2]. If so, the edges intersect, and their intersection point is at $u+t(v-u)$. If t falls out of this range, then their intersection point lies outside the line segment (u, v). Likewise, if r falls out of this range, then their intersection point lies outside of the line segment (w, z). Either way, the edges do not intersect.

We can detect if a drawing is planar by executing this for every pair of edges and checking that no edges intersect. If it is non-planar, the function outputs the intersection point to be displayed.

5.4.5 Finding Minors

We implement a `MinorRearrangeableGraphDrawing` class with some additional methods to check if a graph is a K_5 or a $K_{3,3}$, detect when vertices are dragged over each other, and perform deletions and contractions. To detect a K_5, the vertex and edge count is sufficient. For a $K_{3,3}$, after checking the vertex and edge counts, we choose a vertex arbitrarily and then its three neighbours become one of the two parts. We can then ensure that all of the vertices in the part are connected to one in the other part.

5.4.6 The User Interface

The user interface required creating the appropriate nodes and attaching the relevant scripts. The `Control` nodes need to be anchored to the correct positions to allow for resizing the screen.

5.5 Graph Colouring

The graph colouring topic was added through a `ColourableGraphDrawing` class inheriting from `RearrangeableGraphDrawing`. Then, the user interface could be made similar to before.

The new class has a `vertex_colours` field containing the colour of each vertex. We insist that every vertex has a colour (they are set to 1 during initialisation). Then, `get_colours_used`, `is_valid_colouring`, `get_colour_conflicts`, and `draw_rearrangeable`, were implemented.

[2]In practice, we check if $t \in [0.0001, 0.99999]$ to account for floating point errors.

The lower bound relies on obtaining the size of the maximum clique. This is done by considering every subset of vertices (by considering every binary string), and then checking if that subset forms a clique. While more efficient algorithms exist, no performance issues were encountered.

5.6 Navigation & Menus

The menus use buttons and spin buttons. A spin button is a common alternative to drop-downs in games, where the player selects options with left and right buttons (see figure 4.13). We use the library Godot Spin Button [65] as it provides a good interface and is easy to customise the spin buttons. The library also supports sliders in a similar style for inputting numerical values.

Implementing the world required drawing the roads, huts and other assets and placing them in the scene. The huts and counting problems are given collision boxes of a class `TopicHover`.

```
class TopicHover:
    String name // displayed on the screen
    function go // executed when the player confirms
```

This allowed handling much of the logic in one single class and avoiding re-writing the same code. The `go` method could be set to launch a counting problem pop-up instead of changing the scene.

Navigating the world is done in a car. The following forces apply to the car:

Acceleration There are separate constants for forwards acceleration and for reversing.

Braking This is applied when the player presses the opposite key to their direction of travel. Following a complete stop, there is a 0.2s delay before acceleration begins in the other direction.

Air Resistance Air resistance is proportional to the square of the speed, against the direction of travel. This constant was very carefully adjusted to limit the max speed to the desired level.

Damping Air resistance alone will never stop the vehicle as it is proportional to the velocity, leading to the vehicle continuing forever. Therefore, a small constant force is applied against the direction of travel to ensure the car stops in a sensible distance.

Bounds It stays within bounds with a force proportional to the distance outside of bounds.

The car script is re-used for the boat to navigate between levels in bijections. The constants are adjusted to make the boat slower and turn slower, giving it a more realistic feel but not being too slow to be frustrating. The lilypads display a flower if a level is completed (see figure 5.2).

Music is played through a global `MusicPlayer` script that has a `change_track` method. This allows music to be preserved between some scene changes, but still changed to suit the current scene.

5.7 Dialogue

Dialogue is implemented using the Dialogue Manager [37] library. Dialogue is quick to write as it is in text files. Randomness is supported to give variety in commonly displayed dialogue choices.

To display the dialogue, the default bubble was copied and modified to fit with the colour scheme. Adjustments to the layout were also made. The result is shown in figure 5.3.

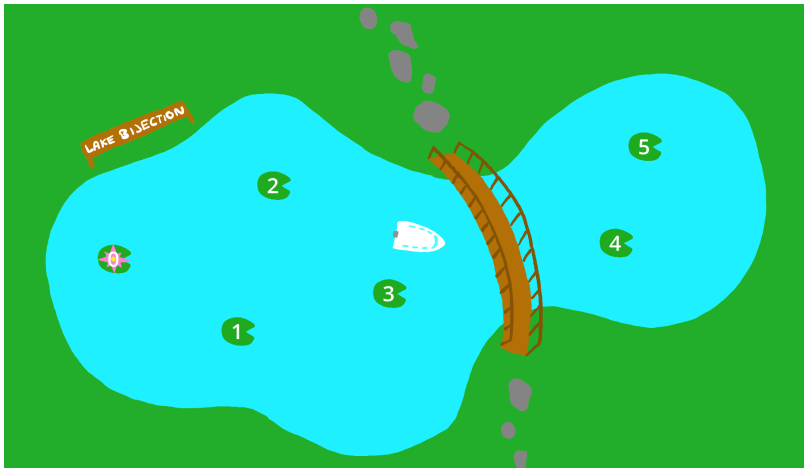


Figure 5.2: The level select screen for bijections, with level 0 completed.

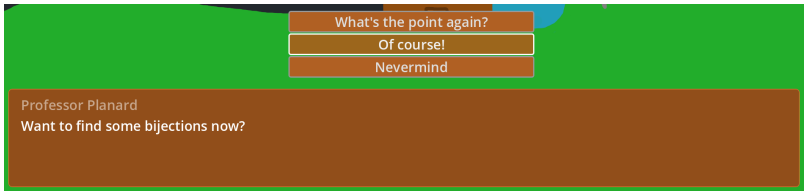


Figure 5.3: Dialogue bubble for the opening to the bijections topic.

Finally, much of the dialogue varies if an NPC has been visited before already. This requires a save system. This is done with a `SaveData` global class that uses Godot's config file system. This was chosen as it allows for storing simple types easily. The seen dialogue is stored in an array, so for example we can test if `"planar"` is in the array to decide which dialogue to show for planar graphs. The save data is also used to vary visual elements in the world such as the type of huts drawn.

Chapter 6

Project Management

This chapter will cover the development methodology, risk assessment, a chronological account of the progress made, and any relevant legal and ethical considerations.

6.1 Development Methodology

Game development projects tend to work best with highly iterative approaches, with more of an emphasis on playtesting and incorporating this into future designs [48]. Educational software can be less iterative. For example, the learning objectives are often defined beforehand and not changed throughout development, whereas in a game the learning objectives may be changed to make the game more fun. Agile methodologies have been found to be more effective in delivering more value to the students [34], so the project uses an agile methodology.

In particular, significant time is allocated to playtesting. It is far more useful to develop a working prototype and test it to figure out what is fun over frontloading the design effort. Nearly half of the development time is allocated to playtesting which is used to continue development in whichever direction is deemed the most fruitful at the time. Prototypes are produced as early as possible, switching the development focus to prioritise playtesting.

Topic	Time Allocated (weeks)	Time to Prototype (weeks)
Bijections	2	2
Catalan Number Zone	2	2
Planar Graphs	2	2
Counting Problems	3	2[1]
Graph Colouring	2	1

Table 6.1: Allocation of time to different topics.

[1]The majority of this time was spent designing the block-based meta-topic which was not pursued beyond the design phase.

The short allocation of development time to the individual topics (table 6.1) forced a prototype early and kept development focused. The time allocated to playtesting was necessary, as topics often required another few weeks to reach their final stages. Since this was allocated to playtesting, it could be used where it was most valuable rather than being allocated to one particular topic.

6.2 Risk Assessment

Table 6.2 shows the risks associated with the project using a 5x5 matrix approach [35].

	Before Mitigation			After Mitigation		
Risk	Prob.	Impact	Risk Level	Prob.	Impact	Risk Level
Data Loss	2	5	10 (High)	2	2	4 (Low)
Unavailability of Computer	2	4	8 (Medium)	2	1	2 (Very Low)
Unavailability of Assets	3	2	6 (Medium)	1	2	2 (Very Low)
Lack of Interest in Testing	3	3	9 (Medium)	2	3	6 (Medium)
Development Overruns	4	3	12 (High)	2	3	6 (Medium)

Table 6.2: Risks associated with the project.

The mitigations are as follows:

Data Loss GitHub is used to store the code and manage versions, and this is synced daily at a minimum. This means that even if data loss occurs, at most one day of progress will be lost.

Unavailability of Computer Should the personal desktop become unavailable, a laptop is available. Should these both become unavailable, the department computers may be used.

Unavailability of Assets Assets may instead be drawn manually, as they do not have to be professional quality.

Lack of Interest in Testing A breadth of opinions may be gained at evaluation day and Warwick Game Design Society [62]. In-depth opinions may be gained by low-risk personal connections.

Development Overruns Significant slack time is allocated in the schedule.

6.3 Progress

This section gives a chronological account of the progress made during the project, summarised in table 6.3. The initial plan can be found in appendix B.

After the specification, development began with the bijections topic, with a prototype created in two weeks. In parallel, some design work was done for the Catalan number zone, namely which problems to include and how to implement it. Following some playtesting, the supervisor noticed that this topic could be combined with other topics to form part of a larger proof. The next two weeks comprised of designing the counting problems module as a way to combine different topics together, however this was ultimately not pursued any further as it would have been too time-consuming for the technical contribution offered. A meeting with the module organiser for

Week	Progress
1	Project Specification
2-3	Bijections (prototype)
4-5	Counting Problems (design)
6-7	Planar Graphs
8	Progress Report
9-10	Testing & Polishing

(a) Term 1

Week	Progress
1	*No progress due to illness*
2-3	Bijections
4-5	Catalan Number Zone
6-7	Polishing & Evaluation
8	Graph Colouring & Polishing
9-10	Presentation & Polishing

(b) Term 2

Table 6.3: Weekly progress in the project.

MA241 [41] helped to confirm the impact of the solution and make changes. The planar graphs topic was implemented over the next two weeks in time for the progress report, which was the focus for the following week. The final two weeks were focused on introducing menus, conducting (solo) playtesting and various improvements were made during this time. By the end of term 1, the game was a prototype 'vertical slice' containing two topics.

Term 2 began by revisiting the bijections topic and restructuring the scene and some of the code to support different values for n side-by-side as well as drawing diagrams. The hints system was also added at this point. These changes were necessary to facilitate the Catalan number zone, which was developed over the following two weeks. Weeks 6-7 contained polishing and improvements in advance for evaluation day, the first evaluation point for the project. Week 8 began with the graph colouring topic, and fixing various usability issues discovered in evaluation day. The term ended with the presentation and some further polishing.

The Easter holidays & early term 3 consisted of writing up this report and doing further development such as bijections code input, a narrative, and many small improvements discovered in playtesting. While a fifth topic was planned to follow the presentation, this was de-prioritised in favour of improvements to existing topics such as the custom parser in bijections.

The modular nature of the project was a real strength which allowed development to proceed in a different order to the original plan. This allowed prototypes to be obtained earlier and for more playtesting to be done. While the order of completion was changed, the actual topics that we completed remained largely unchanged (see table 6.4).

Planned Topic	Result	Comments
Basic Counting Problems	Done	In-world (see section 4.7)
Finding Bijections	Done	As planned
Catalan Number Zone	Done	As planned
Matchings / Colourings	Done	Colourings
Planar Graphs	Done	As planned
Trees & Cycles	Not Done	Not needed[2]

Table 6.4: Completed topics vs initial plan.

[2]Adding this would make very little technical contribution to the project.

Overall, the progress was as expected and the strong initial plan allowed the project to achieve a successful outcome without risking its impact.

6.4 Issues Encountered

As a result of the initial planning done, progress largely proceeded as expected with no major unforeseen issues encountered. Illness removed one week of development time, which was not a major issue due to the slack time allocated. The counting problems meta-topic was found to be unsuitable for the project so this was quickly de-prioritised before any development time was allocated and subsequently wasted. Despite not being developed, the plans for the topic remain a useful avenue for future work. Beyond this, there were some minor issues of things taking longer than expecting, but no major issues.

6.5 Legal, Social, Ethical and Professional Considerations

Ethically, playtesting was limited to providing the game and collecting feedback through interviews and questionnaires. Data from questionnaires is anonymised. All data recorded is strictly limited to the experiences with the game, and will not include any personal information. Interviews may distinguish based on factors such as level of experience with combinatorics or familiarity with games, however this will not include information on demographics such as gender or ethnicity. Therefore, ethical approval is not required.

Legally, if assets are used from other sources, the license will always be checked carefully to ensure it is legal to use in the game.

Chapter 7

Evaluation

Evaluation is primarily done through evaluation day and extended playtesting. After considering the methods of evaluation, each objective will be considered in turn.

7.1 Evaluation Day

Evaluation day took place on Wednesday 19th February. At this point, the project had bijections, Catalan number zone, and planar graphs, but did not feature graph colouring.

The feedback was obtained through a standardised form. The form was kept short to increase the number of responses. In total, there were 8 responses.

The questions were:

1. How much fun did you have with each topic? (1-5 scale for each topic)
2. How useful did you find each topic for learning about combinatorics? (1-5 scale for each topic)
3. How easy-to-use was it? (1-5 scale)
4. Please rate the theme of the game (1-5 scale)
5. Do you have any concerns about the accessibility of the game? (free text)
6. If you have any more feedback, feel free to leave it here (free text)

The data will be referred to when evaluating the objectives. The full data is given in appendix A.

The form was done on Google Forms [11], shared using a QR code. The link was not shared anywhere else, so we can be reasonably sure that all of the responses came from people who did play the game. One issue was with the number of people entering N/A differing between questions. N/A is intended for people who have not played the topic, so these should match up between question 1 and 2. However, they are often off by one. This means that some people may have treated N/A as an "I don't know" option, or they may have skipped through the form without

enough care. Although we cannot entirely guarantee the validity of the responses, there are no major issues that warrant the exclusion of any data.

7.2 Extended Playtesting & Interviews

While evaluation day sought to gather a broad range of opinions, this method seeks to understand the full user experience in more depth. The interviews were conducted on various dates following April 10^{th}, at which point all topics were present and the project was in a near-final state.

Playtesters are first provided with the game to play, while observations and feedback were recorded. Unlike evaluation day, no explanations were given to the playtesters so it relied entirely on the in-game explanations. This was important so that the user experience could be properly evaluated. However, playtesters were sometimes prompted to switch topics to ensure that they all experience all topics within the time constraints. The playtest is followed by an interview to fill in any gaps in understanding. To ensure the interviews remain focused, some desired outcomes (questions) were formalised in advance. Overall, these lasted a minimum of one hour.

The playtesters can be found in table 7.1. All playtesters have been anonymised for this report.

No.	Date	Degree	Prior Experience
1	10/04	Discrete Mathematics (3rd)	Taken MA241 [41]
2	12/04	Aerospace Engineering (1st)	None
3	16/04	Discrete Mathematics (Grad. 2024)	Taken MA241 [41]
4	27/04	Computer Science (3rd)	Basic

Table 7.1: Playtesters for extended playtesting.

7.3 Requirements

The requirements were designed in 3.5 to achieve the objectives. While the evaluation will be against the objectives and not the requirements, it is useful to consider which requirements have been met. It can be found in table 7.2.

No.	Achieved	Not Achieved
1	(a), (b), (c), (d), (e), (f)	
2	(a), (c), (d)	(b), (e)
3	(a), (b), (c), (h)	(d), (e), (f), (g)
4	(a), (b), (c)	(d)
5	(a), (b), (c), (d), (e), (f), (g), (h)	(i), (j), (k)
6	(a), (b), (c), (d), (e), (f), (g)	
7	(a), (b), (c), (d), (e), (f)	
8	(a), (b), (c), (d)	
9	Covered during objective 4 (accessibility)	

Table 7.2: Achievement of requirements.

Most requirements that were not achieved were de-prioritised as the time required would not add significant technical contribution to the project. For example, 2b and 3d required a certain amount of content, and these were narrowly missed. The project is focused on giving a vertical slice of the game rather than containing large amounts of content, as further content is easy to add in future work. 2e and 3f would require copying proofs from established literature which would not have added any technical contribution. Other features may have had a small impact on the enjoyment of the game, but were not prioritised as they were lacking in technical contribution.

7.4 Objective 1: Content

The original aim was to include a variety of relevant and interesting topics that are commonly taught at university, such as MA241 at the University of Warwick [41] and MATH43920 at Durham University [42]. This has been achieved with the four topics present in the final game.

All the topics are commonly taught at university. At the University of Warwick, MA241 [41] covers all of the topics in the game. At Durham University, MATH43920 [42] covers the topics not involving graphs and MATH1031 [12] covers the topics that involve graphs. At the University of Oxford, all topics are covered in Discrete Mathematics [13].

Each topic is also relevant to combinatorics and adjacent fields. Finding bijections is a very common technique, which is practised by two topics. The topic also provides a way to explore other techniques, such as stars and bars. Finding bijections is also useful across mathematics, so these topics have a wide impact. Planar graphs have applications to most areas of graph theory, and familiarity with the Kuratowski-Wagner theorem will help with other forbidden minor theorems. Graph colouring is a fundamental topic in combinatorics and is notable for reducing to lots of constraint-satisfaction problems.

The topics are varied, so as to cover different parts of the field. While the project could have focused on entirely graph theory or entirely enumerative combinatorics, the project combines both areas. While some similarity is inevitable and more variety was planned, this was ultimately decided against to allow for more focus on a smaller range of topics.

Finally, the topics are not already present in a video game in the same way. This is important to ensure that this project is a new contribution. In particular, many of the potential topics in graph theory were not implemented as they are already present in a game and this project could not contribute significant value beyond what already exists.

Overall, this objective is achieved. While it is always possible to add more topics, the present number of topics is sufficient to have a significant impact and cover enough technical complexity for a third year project.

7.5 Objective 2: Learning

The project aims to improve learning outcomes for those studying combinatorics at university. This section first considers the perceived learning from students and then the particular learning objectives in section 3.4.

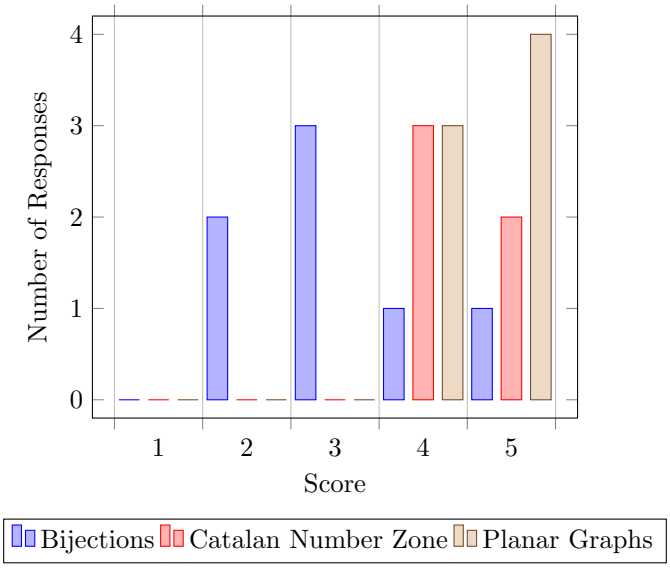


Figure 7.1: (Evaluation day) "How useful did you find each topic for learning about combinatorics?".

Bijections received mixed feedback at evaluation day, with a median and mode of 3 out of 5 and a mean of 3.1. Unfortunately, evaluation day was not a suitable environment to test this topic. The topic is designed to take 5-15 minutes per level, and there was not the time to do this in evaluation day. Most testers missed the point of the topic, thinking that it was about entering in a bijection that was already known rather than devising a new bijection. One put "binary tree and paranthesis bijection may not be that easy to understand for some beginners, so some tutorial should be provided".

The Catalan number zone, surprisingly, scored higher, with a mean of 4.4 out of 5. While the Catalan number zone is even harder than the bijections topic, the people that reached this topic were a smaller sample of more invested people. In the bijections topic, most people attempted the tutorial level or simple levels to understand the premise of the topic, so it did not appear useful. However, the Catalan number zone consistently presented challenging problems and many people commented on the utility of randomly generating problems.

During extended playtesting, playtester 1 found these two topics extremely useful as it is very similar to the approach they would use for revision. They commented that the integer partitions was especially useful, and the visualisations offered by both topics added a lot of value. Playtesters 2-4 also found it useful but tended to prefer the graph topics, potentially due to having less experience in the subject area. Playtester 2 found level 3 very satisfying to work out, and was then able to apply this to one of the bonus counting problems in the world. Playtesters 1 and 2 both commented that the overall value depends heavily on the levels present, as the first few levels were too simple to be useful. This is as expected, as these levels are designed to serve as an introduction to the topic. The definitions were found to be useful.

The code input varied upon the level of experience with coding. In particular, playtester 2 struggled to implement a very simple bijection using code, but would have found it useful if they were more

experienced with coding. Playtester 1 found it useful for simpler problems, but would not always use it due to the implementation time. All playtesters commented on a lack of sufficient explanations and examples to help them write the bijection, as it takes some time to learn the intricacies of a new language. Playtester 4 was the quickest to write a simple bijection, and was impressed at the infinite loop detection and versatility of the language. A block-based instead of a text-based language would solve some of these problems, and is an opportunity for future work. However, code input will always be more difficult than graphical input so is left as an optional input method that is not required to complete a level.

Opinions differed on which topic was better out of bijections and the Catalan number zone. Playtester 1 found them equivalently useful, playtester 2 found bijections more useful, whereas playtester 3 found the Catalan number zone more useful. Playtester 4 found the bijections topic the most useful initially, and then the Catalan number zone more useful to generate more problems. All agreed that both topics are useful to practise the technique of finding bijections.

Planar graphs received very positive feedback on evaluation day, with a mode of 5 and a mean of 4.6. This topic naturally proved more suitable for evaluation day, as the concept is easy to understand in a few minutes. It was easy for respondents to see the value offered by this topic, so all rated this either a 4 or a 5.

The feedback from playtester 1 was less positive. They often preferred to use other techniques such as Euler's formula, which the game does not cover. They did find it helpful for gaining intuition, especially with edge contractions, but would not have used it as part of their revision. The other playtesters were more positive, all commenting this is their favourite topic. After a short period of time, all playtesters found themselves quicker at solving the problems presented to them and with a better understanding of the definitions and theorems used.

All the additional tools were found to be very useful. The graph manipulation buttons were useful, including the ability to re-draw a graph in a different way. The highlighting of high degree vertices was also found to be useful, as well as the edge counters compared to K_5 and $K_{3,3}$. These tools make the game more effective in education.

Graph colouring was not present during evaluation day, so results are only from extended playtesting. Playtester 1 found it to be useful, especially the bounds and the 4-colour theorem. They commented how it could be useful to practise running algorithms to colour graphs, although they would not use it to practise finding graph colourings as they do not need more practise in this area. Playtesters 2 and 4 both understood how to colour a graph, but didn't have any prior knowledge of the theorems used to establish bounds and so did not understand this area. A suggestion was to display the statement of the theorems upon hovering over the name. The graph generation was found to consistently pose interesting and varied problems.

Following the playtest, playtesters were asked to rate how well the game achieves each learning objective from 1 to 5, where a 5 means fully achieved. N/A was given where the playtester did not experience this part of the game enough, or was unsure. The results are in table 7.3.

Overall, all the learning objectives have been achieved. The counting problems are not the focus, although when they were tested, feedback was positive. More experienced players tended to find the first two topics more helpful, whereas less experienced players tended to find the latter two topics more helpful. All topics were found to be very useful in achieving their educational objectives.

Obj.	Ratings from Playtester			
N	1	2	3	4
1a	5	5	5	5
1b	5	5	5	5
1c	5	5	4	5
1d	5	4	5	5
1e	5	5	5	5
1f	5	5	5	N/A
1g	N/A	N/A	N/A	5
1h	4	4	5	5
1i	N/A	N/A	5	N/A
1j	N/A	N/A	5	N/A
2a	5	5	5	5
2b	5	5	4	5
2c	5	5	5	5
2d	5	4	5	N/A
2e	5	5	5	N/A

(a) Learning objectives 1a-2e

Obj.	Ratings from Playtester			
N	1	2	3	4
2f	5	5	5	N/A
2g	5	5	4	5
2h	5	5	5	5
2i	5	4	5	5
3a	5	5	5	5
3b	5	5	5	5
3c	5	5	5	5
3d	5	5	5	5
3e	5	5	5	5
3f	3	5	5	5
3g	5	5	5	N/A
4a	5	4	5	5
4b	N/A	4	N/A	N/A
4c	N/A	N/A	N/A	N/A
4d	N/A	N/A	N/A	N/A

(b) Learning objectives 2f-4d

Table 7.3: Ratings from each playtester on how well each learning objective was achieved.

7.6 Objective 3: Fun

The game aims to be engaging for people that are studying combinatorics at university. As a bonus, the game could also be fun for people who are not studying combinatorics at university.

The feedback for bijections and the Catalan number zone was mixed, with means of 3.3 and 3.5 respectively. This will partly be due to a misunderstanding of the purpose of the topic, as discussed in the previous section. There was not the time for anyone to deduce a complex bijection on their own, which is where most of the fun is expected to lie.

During extended playtesting, playtester 1 found it very enjoyable but only to the extent that they enjoy solving maths problems. If they didn't enjoy solving maths problems in their free time, they would not have found it as enjoyable. None of the other playtesters especially enjoy solving maths problems in their free time, and therefore all ranked this topic as their least favourite for how much fun is offered. However, all commented that this topic was more fun than merely solving the problems on paper.

As with learning, planar graphs performed extremely well in evaluation day, with a mean score of 4.8 out of 5. All playtesters also found planar graphs to be the most fun topic, due to the simple nature of it but the complexity of problems offered. Many puzzle games are designed to lead the player towards a solution that is nearly correct, before they have to change everything to produce a valid solution [22]. Planar graphs often does this, and it leads to a very fun puzzle. Proving graphs are non-planar is also fun, although can sometimes be tedious to spot high degree vertices. This was later improved by highlighting high degree vertices in yellow, which was found to be very useful by all the playtesters.

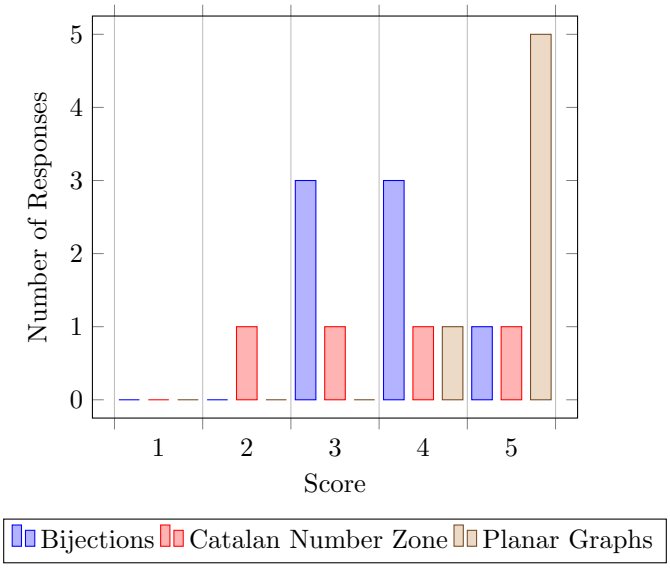


Figure 7.2: (Evaluation day) "How much fun did you have with each topic?".

Playtester 1 found graph colouring too simplistic and ranked it their least favourite topic for fun. Other playtesters appreciated the simplicity of the topic and found the topic fun to play. The topic doesn't offer as much depth as planar graphs, so it didn't tend to retain interest for as long, although players had fun while they were playing it.

As of evaluation day, the world and driving was implemented although most of the narrative was not yet present. Most people found the world menu to increase their enjoyment of the game, although some people found it difficult to control the car. This was later addressed with various usability improvements and the teleportation menu. The mean score for the theme of the game was 4.3, showing that people appreciated the visuals and world design. We can also notice that most players experienced bijections before the Catalan number zone, due to the layout of the world.

For extended playtesting, the narrative was also present which improved the enjoyment of the game even further. All playtesters enjoyed the writing and the narrative, and found this made the game more fun and motivated them to progress further. It was also found to be very useful in explaining key concepts from the game to people who did not have the required prior knowledge. The teleportation menu was appreciated to skip past the driving and the dialogue when undesired.

Overall, those with a strong interest in combinatorics enjoyed the complexity offered by bijections and the Catalan number zone, whereas the graph topics were found to be more enjoyable to a wider audience. All playtesters enjoyed their time playing, especially due to the world design and narrative in the game.

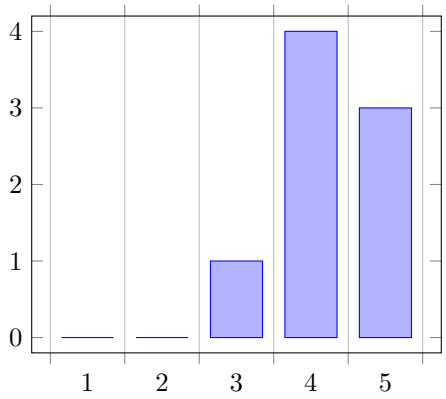


Figure 7.3: (Evaluation day) "Please rate the theme of the game". Description: "This includes the quality of visuals, world design and how coherent everything felt as a video game".

7.7 Objective 4: Accessibility

The games industry is known for often having substandard accessibility options which make games unplayable for certain groups of people. This is especially problematic if applied to the education sector, where the inability to use a certain teaching resource could lead to worse learning outcomes, and therefore worse job prospects. Universities such as Warwick are committed to making their courses more inclusive [58] [61]. Therefore, accessibility is a core goal for the project.

The requirements are sorted into gaming, education, ease-of-use and hardware requirements.

7.7.1 Video Game Guidelines

The requirement is for all basic guidelines on Game Accessibility Guidelines [33]. The relevant guidelines have been summarised in table 7.4.

Some of these required changes to the interface. Requirement 2 requires the controls to be as simple as possible, and that the user interface can be accessed in the same way as the gameplay. This was achieved by ensuring the entire game can be controlled with a mouse only. This required some alternative menus such as the bijections level select screen, which can be toggled in settings. Since everything is accessible with a mouse, there was no need to implement remapping of controls.

Interactive tutorials were somewhat provided in the dialogue on the introduction to each topic. There is also often text within each topic that explains the task more briefly, however there are no interactive tutorials within each topic. While this would help the solution a bit, it would require more development time and was not prioritised.

7.7.2 Education Guidelines

Some issues apply especially to educational contexts. In particular:

1. Text should be displayed in a clear format, with sufficient size, spacing and contrast to the

N	Description	Status
1	Buttons: size and spacing	Passed
2	Input methods	Passed
3	Simple controls	Passed
4	Remapping controls	Failed
5	Avoid flickering images	Passed
6	Read dialogue at own pace	Passed
7	Interactive tutorials	Partial
8	Simple clear language	Passed
9	Start without many menus	Passed
10	Text formatting	Passed
11	Text size	Passed
12	Text contrast to background	Passed
13	No essential info by colour alone	Passed
14	No essential info by sound alone	Passed
15	Separate volume sliders	Passed
16	Save settings to disk	Passed
17	Choice of difficulty	Passed

Table 7.4: Basic requirements from Game Accessibility Guidelines [33].

background [57].

2. Amount of text should be reduced as much as possible. If there are large amounts of text, it may be useful to facilitate having it read aloud [26].

Many of the initial user interface designs used a font size of 16. This was deemed too small, so was subsequently increased to 20 to pass this requirement. The fonts used are clear and there is always enough contrast to the background.

The amount of text is reduced by ensuring all the explanations are brief. The dialogue system is also very successful for this requirement, as it allows for breaking up large portions of text into manageable chunks that are digestible as a conversation.

7.7.3 Ease-of-Use

On evaluation day, the mean score for ease-of-use was 4 out of 5. While this is passable, it is lower than desired. Many of the issues leading to this score were identified on the day, and many issues were fixed immediately (see appendix A for the issues that were fixed). Extended playtesting did show a noticeable improvement in ease-of-use, with the dialogue in particular proving helpful to explain the premise of each topic. While further usability issues were raised, the project's ease-of-use was deemed satisfactory following the playtesting. Continual improvement of the user interface to make it more intuitive is an opportunity for future work.

Some suggestions for future improvements to usability include:

- Add interactive tutorials in each topic.
- Add interactive explanations for the controls.

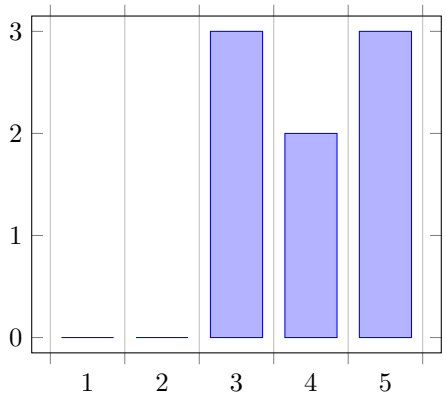


Figure 7.4: (Evaluation day) "How easy-to-use was it?".

- Add a message in bijections that pops up when completing a value of n to explain when validation occurs.
- Only enable one side of planar graphs at a time, requiring a button press to switch to that side. This will make it clearer that only one side is possible to complete.
- When completing a planar graphs level, display a pop-up, like with bijections. Don't allow the user to attempt the other side upon completing one of the sides.
- More documentation and examples for code input in bijections.

7.7.4 Hardware

Finally, we need the game to run on a wide variety of hardware. The original requirements stipulate:

- Any modern processor with integrated graphics.
- A web browser.
- Small screen sizes.

The game offered a 'good experience' on Google Chrome [14] on a low-end computer[1]. The game also managed to consistently draw 60 frames per second on a mid-range computer[2]. Performance was just as good as a native build on a variety of web browsers. While no native 720p display was available for testing, the game performed well when the window was re-sized to 720p, and all the UI elements were still readable at 13". Therefore, these requirements are all passed.

[1] Lenovo Ideapad 3 ChromeOS (14AP06), with an AMD 3015Ce.
[2] AMD Ryzen 7 5700U with 16 GB RAM and integrated graphics.

Chapter 8

Conclusion

Overall, the project has reached a successful outcome. This chapter will consider the achievements, limitations and future work.

8.1 Achievements

The final product covers a variety of topics in combinatorics, is fun and useful in education, and is accessible to nearly everyone. The game is found to improve learning outcomes for those studying combinatorics, and provide a fun game even for those who are not. Over 80 [15] universities across the UK (and even more worldwide) teach mathematics at undergraduate level. While the exact impact varies between courses, all mathematics courses would benefit from students playing this game. Furthermore, the game is fun and accessible to those who are not studying combinatorics, widening the appeal of combinatorics and developing mathematical ability in a fun way. This gives the project a huge impact.

The project also makes technical contributions to combinatorics and game development. The bijections topic contributes a custom language with a recursive descent parser that can be used in other Godot projects. This applies a novel approach to validating bijections by generating all of the cases on either side and checking for injectivity and closure. The Catalan number zone contributes new bijections and implementations for some well-known bijections. Many different elements are drawn in Godot, and the topic is easily extendable to add more elements. Planar graphs and graph colouring contribute a well-structured graphs library for Godot, which is easily usable in other Godot projects.

Overall, the project demonstrates the results of using video games in university-level mathematics, an area that is under-explored. The strong initial plan and appropriate methodology allowed the project to proceed towards a successful outcome with minimal risk. The final solution demonstrates significant impact and technical achievement, and is therefore a successful third year project.

8.2 Limitations

The project does have some limitations. The bijections topic contains two input methods, neither of which are perfect. It would be ideal if there was a single input method that had high accuracy of validation and helped to develop intuition, although this does not appear to be possible. The Catalan number zone is lacking in the number of bijections implemented, which would form more alternate solutions. The code input becomes especially challenging for the complex representations used in the Catalan number zone. The planar graphs topic uses only one method to prove if a graph is planar and non-planar, however many other results exist which may be able to prove these faster. The graph colouring topic is limited in the number of bounds offered, and the lack of assessing if a colouring is indeed optimal.

As a whole, the game is limited in its application towards module organisers. Module organisers are not given control over which problems are presented, and are not able to check on progress by students. Additionally, some may be put off by the playful aesthetic offered by the game. This was an intentional choice to prioritise the needs of the student over the module organiser, however it may limit the project's reach if module organisers do not officially endorse the tool.

8.3 Future Work

Future work can be to the technical content or to the theme of the game. Future improvements to the theme could include improvements to graphical fidelity, expanding the narrative and doing even more playtesting to motivate player progression even further. Future improvements to the technical content could involve improving existing topics or adding new topics. Improvements to existing topics could include:

Bijections More levels could be added, including for problems often posed in other areas such as algorithmic reductions. Another opportunity for improvement is with validation, potentially marking solutions as correct sooner and capturing more information from the user during the input process. Block-based coding would also be more suitable for certain users.

Catalan Number Zone Primarily, this topic needs more problems and more bijections. A difficulty selection would be useful and can easily be implemented by assigning a difficulty to each bijection and using Dijkstra's algorithm to find appropriate bijections. Furthermore, if this was used by lots of people it could become a database containing lots of new bijections. It could contain a server where users can upload their own bijections and proofs and other people can review them. This would contribute massively to the field, but relies on a large user base. The topic could also be expanded into other sequences like Schröder numbers or Motzkin numbers which are also counted by a large variety of problems.

Planar Graphs More sophisticated graph generation could generate problems of particular difficulty and with particular properties. The topic could also be extended to work with other forbidden-minor properties, such as ladder graphs or linklessly embeddable graphs.

Graph Colourings The topic could be used to demonstrate certain proofs in colouring. Graph generation could incorporate particular properties to explore how these affect the colourability of the graph, and there could be a mode in which graphs may be modified by the user. It would also be useful if the game identified an optimal colouring.

New topics could include:

- Minimum spanning trees [52].
- Matchings [40].
- Hamiltonian cycles and Ore's theorem [49].
- Isomorphic graphs (listing all possible graphs) [43].
- Flows (such as the Ford-Fulkerson algorithm) [38, pp. 338–357].
- Chip-firing game [20].
- Kasteleyn's theorem for planar graphs [63].
- Entropy and Shearer's lemma [59].
- Error-correcting codes [50].
- Partially ordered sets [53].
- Extremal graph theory (constructing certain graphs, Turan's theorem) [21].
- Random graphs (finding threshold functions) [28].
- Finite projective planes and latin squares (could link to bijections) [25].
- Cryptography (such as Shannon's coding theorem, writing hash functions) [55].
- Quantum computing [45] [24].

A final topic that would be useful is the counting problems meta-topic as discussed in section 4.7. The topic would present problems that require the use of multiple techniques to solve them, and require players to drag-and-drop in the techniques they need. This would elegantly combine together different topics to solve exam-style problems, however it would require lots of hard-coding of solutions to implement.

8.4 Author's Assessment of the Project

What is the (technical) contribution of this project? The project applies new computer techniques to a mathematical field to create a fun and educational game that improves learning outcomes and engagement when used alongside lectures and seminars to teach combinatorics at university. The project contributes new results in combinatorics, and implementations for existing results in Godot.

Why should this contribution be considered relevant and important for the subject of your degree? This project allows me to apply the programming skills I have gained through the computer science parts of my degree to a mathematical field, incorporating both aspects of Discrete Mathematics.

How can others make use of the work in this project? The project may be used by any student that is learning combinatorics, and any student with a mathematical background that enjoys fun puzzles. Others may also incorporate the systems developed for the game in their own similar games or interactive worksheets. For example, snippets of the game could be incorporated directly into lecture notes. The graphs system and custom language could also be used for similar games made in Godot.

Why should this project be considered an achievement? I have applied various important and commonly-taught topics from combinatorics to a video game for the first time, to achieve a game that is both fun and educational. This required a thorough understanding of the underlying techniques to design a video game to teach it, and programming expertise to implement this.

What are the limitations of this project? Level-based topics currently lack a wide range of levels and problems, and graph topics could have more extensive graph generation. However, the main limitation is that the game only covers four topics, which presents many exciting opportunities for future work.

Appendix A

Evaluation Day Survey

The full results of the survey are given here. There were 8 responses in total.

A.1 Questions 1-4

Questions 1-4 are on a scale from 1 to 5 with 1 being the worst and 5 being the best.

Question 1 How much fun did you have with each topic?

Results See table A.1.

Question 2 How useful did you find each topic for learning about combinatorics?

Results See table A.1.

	Question 1: Fun						Question 2: Learning					
Topic	1	2	3	4	5	N/A	1	2	3	4	5	N/A
Bijections	0	0	3	3	1	1	0	2	1	3	1	1
Catalan Number Zone	0	1	1	1	1	4	0	0	0	3	2	3
Planar Graphs	0	0	0	1	5	2	0	0	0	3	4	1

Table A.1: Results for questions 1 and 2.

Question 3 How easy-to-use was it?

This includes the quality of user interfaces, and how well you understood the problems being presented to you.

Results See table A.2.

Question 4 Please rate the theme of the game.

This includes the quality of visuals, world design and how coherent everything felt as a video game.

Results See table A.2.

Question	1	2	3	4	5
Q3: Ease of Use	0	0	3	2	3
Q4: Theme	0	0	1	4	3

Table A.2: Results for questions 3 and 4.

A.2 Questions 5-6

Questions 5 and 6 are text input. Both are optional. All responses are left verbatim here.

Question 5 Do you have any concerns about the accessibility of the game?

If not, please leave this blank

- Would be good to have some indicators how to drive the car (and play the game, e.g. binary tree and paranthesis bijection may not be that easy to understand for some beginners, so some tutorial should be provided)
- Colourblindness? Can you customise the controls too (for e.g. one-handed people).
- Could have instructions on what you're meant to do (e.g. Match bijections, prove if same graph is planar/non planar)
- If people don't play many computer games they may not intuitively know the controls
- No

Question 6 If you have any more feedback, feel free to leave it here.

- I found it hard to use the car in the menu, also I think for the bijection n game you should have a button that says how to move onto the next game
- I don't know if there's a story but that would be cool if there was. Otherwise the visuals were really cute and I liked the game!
- Great interface
- No

A.3 Observations

The following issues were observed while watching people play:

- The dots on the end of elements in bijections are not a touch target, but many people were dragging from the dots instead of from the elements as was intended. These have since been made clickable, with a larger radius than the visual indicator.
- It wasn't immediately clear how to switch the value of n in bijections - buttons have now been added to make this even easier.
- The hints system was not clear as a hint system - many people clicked this without realising it was costing them a hint.

- Empty cases in bijections were confusing.
- Mouse controls were unclear in bijections.
- Keyboard controls were unclear in bijections.
- Keyboard controls for driving the car need to be displayed.
- Kuratowski-Wagner theorem needs to be explained more.
- An alternate menu to driving.
- Flip the world so that the car starts pointing upwards (prevents confusion as W key is expected to go up).
- Automatically validate in bijections when there's only one case.
- Add the word "OR" between planar side and non-planar side to make it clearer.

All have since been fixed.

Appendix B

Planned Timeline

Figures B.1 and B.2 contain the original plan from the project specification. Figure B.3 contains the updated Gantt chart from the progress report.

Section 6.3 discusses the actual order of progress made.

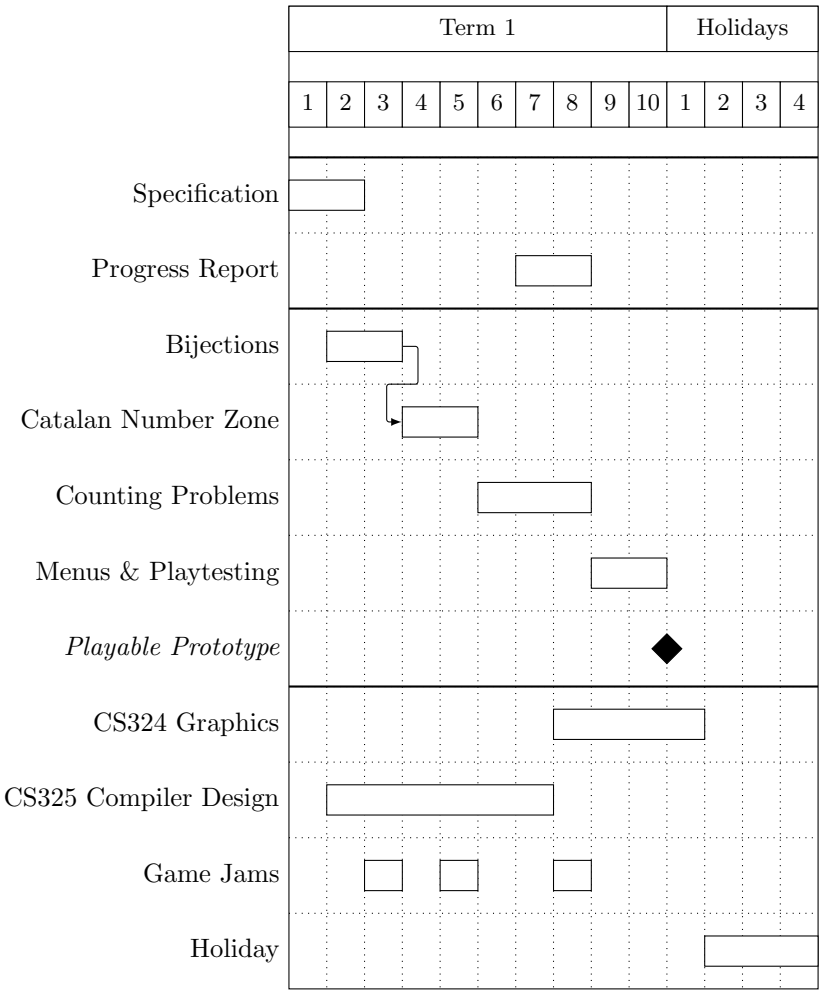


Figure B.1: Original plan for term 1.

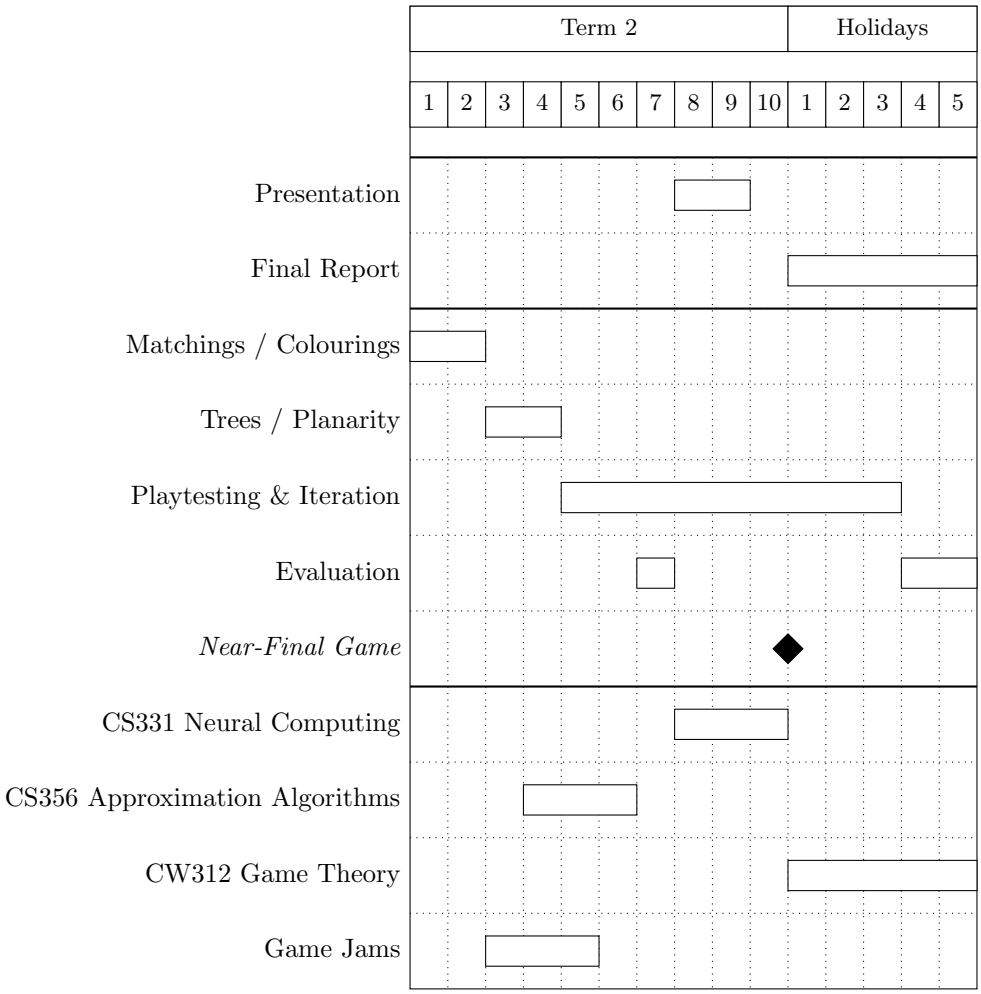


Figure B.2: Original plan for term 2 from the specification.

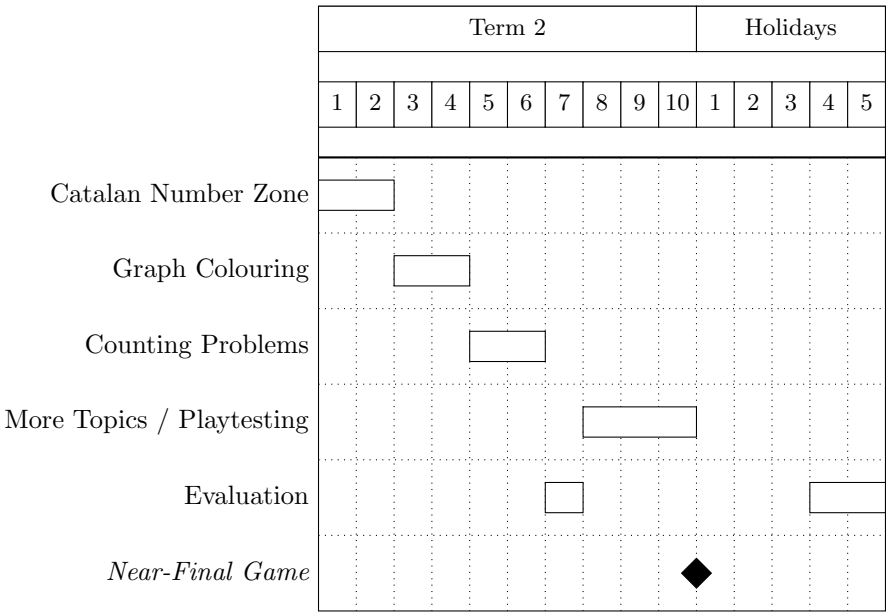


Figure B.3: Updated plan for term 2 from the progress report (deliverables and other commitments have been removed as they are unchanged).

Glossary

acyclic Containing no cycles (paths with the same start and end vertex). 84

adjacent Two vertices u and v are adjacent in $G = (V, E)$ if $(u, v) \in E$ or $(v, u) \in E$. 83

balanced parentheses A sequence of parentheses is balanced if it is generated by the grammar with production rule $S \to \underline{(}S\underline{)}|\varepsilon$. 32, 48, 82

bijection A function $f : X \to Y$ that is injective and surjective. 7, 12, 20–22, 33, 82

bijective proof A proof that two sets are the same size by finding a bijection between them. 20

binary tree A rooted tree where each node has at most two children, and the left and right children are labelled. 32, 82

catalan number A sequence $(C_n)_{n\in\mathbb{N}}$ where C_n = the number of sequences of n opening and n closing parentheses that are balanced parentheses. Note $C_n = \frac{1}{n+1}\binom{2n}{n}$. 10, 13, 31

catalan problem A counting problem A_n where $|A_n| = C_n$ ($C_n = n^{th}$ Catalan number). 7, 31–33, 48, 50

chromatic number The minimum number of colours required to colour a graph, $\chi(G)$. 13, 21, 23

clique A complete graph. 21

complete A binary tree is complete if every node has 0 or 2 children. 32

complete bipartite graph A complete bipartite graph on $2t$ vertices is $K_{t,t} = (A \cup B, \{(a, b)|\, a \in A, b \in B\})$ with $|A| = |B| = t$ and A, B disjoint. 83

complete graph A complete graph on t vertices is $K_t = (V, V \times V)$ with $|V| = t$. 82, 83

connected A graph is connected if there exists a path between every pair of vertices. 84

counting problem A problem to do with counting. With regard to bijections, this means more specifically a sequence $(A_n)_{n\in\mathbb{N}}$ where each A_i is a finite set (the problem is to count the size of each A_i). 10, 12, 13, 25, 82, 83

degree Size of the neighbourhood of a vertex. 21

dyck path A walk consisting of up steps, (1, 1), and down steps, (1, 0), ending at (L, 0) for some length L and never going below the x-axis. 32

edge contraction An edge contraction of $(u, v) \in E$ in $G = (V, E)$ produces a graph $G' = (V', E')$ with $V' = V \cup \{v'\} - \{u, v\}$ where $N(v') = N(u) \cup N(v)$. 13, 21, 23, 35, 83

equivalent counting problems Two counting problems A_n and B_n are equivalent if for every $n \in \mathbb{N}$, $|A_n| = |B_n|$. 10, 12

four colour theorem If a graph G is planar, then G admits a 4-colouring, i.e. $\chi(G) \leq 4$. 10, 13, 21, 23, 36

fáry's theorem Any planar graph admits a planar drawing that uses only straight lines. 53

graph A graph, or a *simple undirected* graph, is a 2-tuple (V, E) where V is a finite set (called the 'vertices' or 'nodes') and $E \subseteq V \times V$ (called the 'edges'). 6, 7, 13, 21, 34–36, 43, 52, 54, 65, 67, 72, 82–84

graph colouring A colouring of a graph G is a proper vertex colouring of G which is a k-colouring of G for some $k \in \mathbb{N}$. 13, 21, 23

graph drawing A drawing of a graph $G = (V, E)$ is a function $f : V \to \mathbb{R}^2$. *Note: we assume that all edges are drawn as straight lines. This assumption does not affect the notion of planarity - see Fáry's theorem.* 52, 53, 84

injective A function $f : X \to Y$ is injective if for every $x_1, x_2 \in X$, $f(x_1) = f(x_2)$ implies $x_1 = x_2$. 82

integer partition An integer partition, or simply a partition, of n is an ordered list of positive integers adding up to n. We usually write this in descending order separated by plus signs, e.g. 4+3+3+1. We may also shorten this representation by using a product, e.g. 4+2(3)+1. 12, 20–22, 47, 84

k-colouring A k-colouring of a graph $G = (V, E)$ is a function $f : V \to \{1, ..., k\}$ such that for every $(u, v) \in E$, $f(u) \neq f(v)$. 13, 83

kuratowski-wagner theorem A graph is non-planar if and only if it has a K_5 (complete graph on 5 vertices) or a $K_{3,3}$ (complete bipartite graph on 6 vertices) as a minor. 10, 13, 21, 39, 53, 63, 77

lhs Left-hand side. 44

minor A graph K is a minor of G if it can be obtained by a sequence of edge deletions, vertex deletions and edge contractions. 13, 21, 34, 83

motzkin path A walk consisting of up steps, (1, 1), down steps, (1, 0), and flat steps, (1, 0), ending at (L, 0) for some length L and never going below the x-axis. 32

neighbourhood The neighbourhood of a vertex is the set of vertices that are adjacent to it. 13, 82

non-planar Opposite of planar. 10, 13, 21, 23, 34, 54, 72

planar drawing A graph drawing is planar if no two edges intersect. 13, 21, 23, 84

planar graph A graph is planar if there exists a planar drawing of it. 10, 13, 15, 21, 23, 34, 36, 72, 83, 84

robinson-schensted correspondence A bijective correspondence between pairs of Young tableaux of the same shape and permutations of size n [18] [51]. 7, 20, 30, 31

rooted tree A tree where one vertex is fixed as the root vertex. 82

schröder path A walk consisting of up steps, (1, 1), down steps, (1, 0), and flat steps, (2, 0), ending at (L, 0) for some length L and never going below the x-axis. 32

self-conjugate An integer partition whose Young diagram is symmetrical over $y = -x$. 20, 47

surjective A function $f : X \to Y$ is surjective if for every $y \in Y$, there exists an $x \in X$ such that $f(x) = y$. 82

tree A connected acyclic graph. 84

Bibliography

[1] URL: https://www.metacritic.com/game/the-witness/.

[2] URL: https://gaming.stackexchange.com/questions/313273/a-witness-marsh-puzzle-having-more-than-one-correct-answer-spoilers.

[3] URL: https://roomescapeartist.com/2018/05/18/the-witness-review/.

[4] URL: https://www.proofblocks.com/.

[5] URL: https://www.jasondavies.com/planarity/.

[6] URL: https://adam-rumpf.github.io/games/chromagraph.html.

[7] URL: https://twinery.org/.

[8] URL: https://scratch.mit.edu/.

[9] URL: https://www.mariowiki.com/Super_Mario_64.

[10] URL: https://godotengine.org/.

[11] URL: https://docs.google.com/forms.

[12] URL: https://apps.dur.ac.uk/faculty.handbook/2005/UG/module/MATH1031.

[13] URL: https://www.cs.ox.ac.uk/teaching/courses/2024-2025/discretemaths/.

[14] URL: https://www.google.com/intl/en_uk/chrome/.

[15] URL: https://www.thecompleteuniversityguide.co.uk/courses/search/undergraduate/mathematics.

[16] Z. Abel et al. "Who witnesses The Witness?" In: *International Conference on Fun with Algorithms* (2019). URL: http://arxiv.org/pdf/1804.10193. (accessed: 24/11/24).

[17] Matthew Barr. "Video Games in Higher Education". In: (2017). URL: https://www.gla.ac.uk/media/Media_276228_smxx.pdf.

[18] Gilbert de Beauregard Robinson. "On the Representations of the Symmetric Group". In: *American Journal of Mathematics* (1938). URL: https://doi.org/10.2307%2F2371609.

[19] Edward E. Bender and S. Gill Williamson. *Foundations of Combinatorics with Applications*. Dover Publications, 2006. URL: http://www.math.ucsd.edu/~ebender/CombText/.

[20] Anders Björner, László Lovász, and Peter W. Shor. "Chip-firing Games on Graphs". In: *European Journal of Combinatorics* 12.4 (1991), pp. 283–291. ISSN: 0195-6698. DOI: https://doi.org/10.1016/S0195-6698(13)80111-4. URL: https://www.sciencedirect.com/science/article/pii/S0195669813801114.

[21] Bela Bollobas. *Extremal Graph Theory*. Dover Publications, 1978.

[22] Mark Brown. "What Makes a Good Puzzle?" In: *Game Maker's Toolkit* (2018). URL: https://youtu.be/zsjC6fa_YBg?t=157.

[23] Dai Clegg and Richard Barker. *Case Method Fast-Track: A RAD Approach*. Addison Wesley, 1994.

[24] *Combinatorial Optimisation*. URL: https://www.quera.com/glossary/combinatorial-optimization. (accessed: 26/04/25).

[25] Bob Connelly. "Classical Geometries". In: *Department of Mathematics, Cornell University* (2008). URL: https://pi.math.cornell.edu/~web452/chapter10.pdf.

[26] Martyn Cooper. "Making online maths accessible to disabled students – issues and lessons from the Open University's experience". In: (Feb. 2011). URL: https://martyncooper.wordpress.com/2011/02/20/making-online-maths-accessible-to-disabled-students-%E2%80%93-issues-and-lessons-from-the-open-university%E2%80%99s-experience/. (accessed: 04/10/24).

[27] István Fáry. "On straight line representations of planar graphs". In: *Acta Univ. Szeged. Sect. Sci. Math* (1948).

[28] Alan Frieze and Michal Karonski. *Introduction to Random Graphs*. Cambridge University Press, 2025.

[29] Thomas M. J. Fruchterman and Edward M. Reingold. "Graph Drawing by Force-Directed Placement". In: *Department of Computer Science, University of Illinois* (1990). URL: https://dcc.fceia.unr.edu.ar/sites/default/files/uploads/materias/fruchterman.pdf.

[30] Joshua Fullard. "Does using games to teach maths at university work?" In: *Warwick Business School* (2024). URL: https://www.wbs.ac.uk/news/games-to-teach-maths-university/. (accessed: 03/10/24).

[31] Joshua Fullard. *Using Games to Improve Students' Engagement and Understanding of Statistics in Higher Education*. Warwick Business School, 2024. URL: https://libjournals.mtsu.edu/index.php/jfee/article/view/2475/1459. (accessed: 24/11/24).

[32] William Fulton. *Young Tableaux, with Applications to Representation Theory and Geometry*. Cambridge University Press, 1997.

[33] *Game Accessibility Guidelines*. URL: https://gameaccessibilityguidelines.com/. (accessed: 03/10/24).

[34] Carina S. GonzÃ¡lez-GonzÃ¡lez, Pedro Toledo-Delgado, and Vanesa MuÃ-Cruz. "Agile human centered methodologies to develop educational software". en. In: *DYNA* 82 (Oct. 2015), pp. 187–194. ISSN: 0012-7353. URL: http://www.scielo.org.co/scielo.php?script=sci_arttext&pid=S0012-73532015000500024&nrm=iso.

[35] Patricia Guevara. "A Guide to Understanding 5x5 Risk Assessment Matrix". In: *Safety Culture* (2024). URL: https://safetyculture.com/topics/risk-assessment/5x5-risk-matrix/.

[36] John M. Harris, Jeffry L. Hirst, and Michael J. Mossinghoff. *Combinatorics and Graph Theory*. Springer-Verlag, 2000.

[37] Nathan Hoad. URL: https://github.com/nathanhoad/godot_dialogue_manager.

[38] Jon Kleinberg and Eva Tardos. *Algorithm Design*. Pearson Education, 2006.

[39] Kazimierz Kuratowski. "Sur le problème des courbes gauches en topologie". In: *Fund. Math.* (1930).

[40] Claudio L. Lucchesi and U.S.R Murty. *Perfect Matchings: A Theory of Matching Covered Graphs*. Springer, 2024.

[41] *MA241: Combinatorics at the University of Warwick*. URL: https://warwick.ac.uk/fac/sci/maths/currentstudents/modules/ma241/. (accessed: 03/10/24).

[42] *MATH43920: Topics in Combinatorics at Durham University*. URL: https://apps.dur.ac.uk/faculty.handbook/2023/PG/module/MATH43920. (accessed: 05/10/24).

[43] Jiri Matousek. *Thirty-three Miniatures*. American Mathematical Society, 2010.

[44] Stephen Melczer. "Bijections and Combinatorial Proofs". In: *An Invitation to Enumeration*. University of Waterloo, 2024. URL: https://enumeration.ca/toolbox/bijections/.

[45] Ashley Montanaro. "Combinatorics in quantum computation, and vice versa". In: *Department of Computer Science, University of Bristol* (2014). URL: https://people.maths.bris.ac.uk/~csxam/presentations/combtalk.pdf.

[46] Emily Naul and Min Liu. "Why Story Matters: A Review of Narrative in Serious Games". In: *Journal of Educational Computing Research* 58.3 (2020), pp. 687–707. DOI: 10.1177/0735633119859904. eprint: https://doi.org/10.1177/0735633119859904. URL: https://doi.org/10.1177/0735633119859904.

[47] Rebecca Odom. *Identifying Self-Conjugate Partitions*. 2022. arXiv: 2208.13729 [math.HO]. URL: https://arxiv.org/abs/2208.13729.

[48] Ann Osborne O'Hagan, Gerry Coleman, and Rory V. O'Connor. "Software Development Processes for Games: A Systematic Literature Review". In: *Systems, Software and Services Process Improvement*. Ed. by Béatrix Barafort et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 182–193. ISBN: 978-3-662-43896-1.

[49] E. M. Palmer. "The hidden algorithm of Ore's theorem on Hamiltonian cycles". In: *Computers Math. Applic.* 34.11 (1997), pp. 113–119. URL: https://mathscinet.ams.org/mathscinet/relay-station?mr=1486890.

[50] Peterson and E. J Weldon. *Error Correcting Codes*. 2nd ed. MIT Press, 1972.

[51] Craige Schensted. "Longest increasing and decreasing subsequences". In: *Canadian Journal of Mathematics* (1961). URL: https://doi.org/10.4153%2FCJM-1961-015-3.

[52] Robert Sedgewick and Kevin Wayne. *Algorithms, 4th Edition (Section 4.3)*. Addison Wesley, 2011. URL: https://algs4.cs.princeton.edu/43mst/.

[53] Dan A. Simovici and Chaabane Djeraba. "Partially Ordered Sets". In: *Mathematical Tools for Data Mining: Set Theory*. Springer, 2008, pp. 129–156.

[54] Richard P. Stanley. *Catalan Numbers*. Cambridge University Press, 2015.

[55] Douglas R. Stinson. "Combinatorial Designs and Cryptography, Revisited". In: *University of Waterloo* (2019).

[56] *The Witness*. URL: http://the-witness.net/. (accessed: 04/10/24).

[57] Clare Trott. "Mathematics, dyslexia, and accessibility". In: (Jan. 2011). URL: https://repository.lboro.ac.uk/articles/conference_contribution/Mathematics_dyslexia_and_accessibility/9373703. (accessed: 04/10/24).

[58] *University of Warwick Inclusion Strategy*. URL: https://warwick.ac.uk/about/strategy/inclusion. (accessed: 04/10/24).

[59] Jan Vondrak. "Lecture 5. Shearer's Lemma". In: *Stanford University* (2016). URL: https://theory.stanford.edu/~jvondrak/MATH233-2016/Math233-lec05.pdf.

[60] Klaus Wagner. "Über eine Eigenschaft der ebenen Komplexe". In: *Mathematische Annalen* (1937).

[61] University of Warwick. "Code of Practice for Disabled Students". In: (June 2023). URL: https://warwick.ac.uk/services/aro/dar/quality/categories/goodpractice/codeofpracticedisabledstudents/disabled_student_code_of_practice_updated.pdf. (accessed: 04/10/24).

[62] *Warwick Game Design Society*. URL: https://www.warwicksu.com/societies-sports/societies/warwickgamedesign/.

[63] Robin Whitty. *Kasteleyn's Theorem*. URL: https://www.theoremoftheday.org/MathPhysics/Kasteleyn/TotDKasteleyn.pdf.

[64] Su-Ting Yong et al. "Exploring the Feasibility of Computer Games in Mathematics Education". In: *2019 IEEE International Symposium on Haptic, Audio and Visual Environments and Games (HAVE)*. 2019, pp. 1–6. DOI: 10.1109/HAVE.2019.8921018.

[65] Nikita Yudin. URL: https://github.com/yudinikita/godot-spin-button.