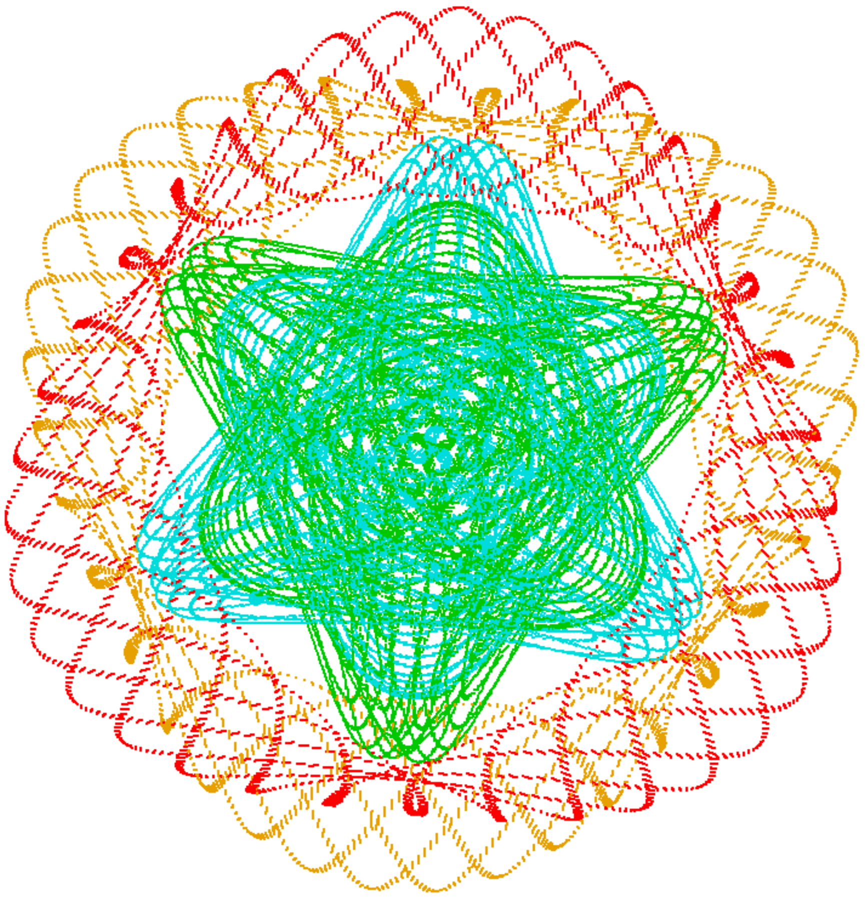
Ninety Degree Edge Rotations in Graph Drawings

Game and Media Technology Master thesis
Utrecht University, Department of Information and Computing Science



Casper van Dommelen, BSc
ICA-3896188

Supervised by
prof. dr. M. J. van Kreveld
Jérôme Urhausen, MSc

February 1, 2019

Abstract

We discuss a rotation-based variant of a geometric puzzle game presented in previous research. While the goal of the aforementioned puzzle game was to make a graph drawing planar by repeatedly swapping the endpoints of any selected edge with each other, the rotation variant presented here rotates an edge ninety degrees in either clockwise, or counter-clockwise direction. Fundamental properties are shown through proofs and a construction. Further analysis is done through random rotations on a large scale, as well as through a breadth-first approach. Unexpected behaviour, where certain patterns arise, was found by performing the operation sequentially in a non-random manner. Additionally, three other move variants are touched upon in less detail and used along with the rotation variant in a new geometric puzzle game. A user study was conducted to research preferred control schemes for two of the move variants.

Contents

1 Introduction **3**
1.1 Related work . . . 3
1.2 Structure . . . 4

2 Fundamental properties of the rotation operation **5**
2.1 The position of the centroid is constant . . . 6
2.2 Sums of squared distances . . . 7
2.3 Potential limit . . . 9
2.4 Quickest growth . . . 10

3 Experimental analysis **15**
3.1 Experiment: random rotations . . . 15
3.2 Experiment: average planar solvability . . . 19
3.3 Experiment: very good instances . . . 22
3.4 Exploratory analysis on rotations in a certain order . . . 23

4 The game **33**
4.1 Basic properties of other moves . . . 33
4.1.1 Collapse . . . 33
4.1.2 Slide . . . 33
4.1.3 Addition . . . 34
4.1.4 Interaction between move types . . . 34
4.2 The game: overview . . . 35
4.3 User study on control schemes . . . 36
4.3.1 The experiment . . . 37
4.3.2 Results . . . 38
4.3.3 Discussion . . . 43

5 Conclusions and future research **45**
5.1 Theoretical . . . 45
5.2 Experimentation . . . 45
5.3 Design . . . 46

A Results of exploratory experiments **49**
A.1 Vertex coordinates of previous graphs . . . 49
A.2 Many more results . . . 50

B Manual for the program used for Section 3 **61**

C User study levels and Questionnaire **63**

D Level generation pilot test **81**

1 Introduction

In a recent paper by Kraaijer et al. [7] a graph based game was introduced where the goal is to make a nonplanar graph drawing of a planar graph planar by swapping endpoints of edges. In their paper they also discuss a number of possible variations on this game. One of these variations uses a stronger version of their swap move, namely one in which the player can only rotate two endpoints of an edge 90 degrees around their middle point. This variant will be analysed in greater detail in this thesis. Although this analysis is far from extensive, some key properties will be proven and some will be approximated through experiments, discussed and speculated upon.

Another goal of this research was to create a game and explore more variations and how they interact with each other. The game immediately provides a practical application of the research conducted within this thesis. This is especially useful since this kind of research is aimed towards games, so an actual game shows what use the research *has*, and gives interested researchers within the field something concrete to research *on*.

1.1 Related work

This research falls into the large body of research dedicated to better understanding games, particularly puzzle games. Its application exceeds defining properties of games, as it provides powerful modelling tools for artificial intelligence and pushes technology [11]. Hearn et al. discuss constraint logic, defining a generic graph based game to formalise different puzzle games, showing that games are a powerful tool for expressing complexity [5].

A more specific topic within game theory includes that of Voronoi games [15] [3]. A concrete example of a commercial game is the mobile game Lines. An instance of Lines presents the player with connected edges, a number of coloured dots on those edges, and a number of different moves available to the player such as adding an edge, or removing a dot. When all moves are done the dots fill up the edges with their own colour with a regular speed. The goal is to have the player's colour cover the largest area.

Graph-based games are the most related to the research presented here. An example is the game often referred to as Brussels Sprouts, or just Sprouts, where the begin state is a selection of vertices and no edges. It is a competitive two-player game and a move consists of drawing a line between two vertices. The rules here are that no lines may intersect, no vertex may have a degree higher than three, and a new vertex should be placed anywhere on the new line, as long as it does not overlap with an existing vertex. The first player that cannot draw any line loses. For this game it has been proven that the number of moves is between $2n$ and $3n - 1$ where n is the number of vertices [9].

Another graph based game is John Tantalo's Planarity, in which the player is presented a nonplanar straight-line embedding of a planar graph [14]. The goal is to drag the vertices around until the graph is planar. Planarity has been the motivation for research into the topic of untangling nonplanar embeddings of planar graphs, specifically straight-line graph drawings. Goaoc et al. [4] prove that finding the minimum number of moves (a move consists of moving a vertex anywhere) is NP-hard and specify further bounds. Verbitsky [17] defines a metric for obfuscation, expressed by the maximum number of edge crossings in a straight line graph drawing. Spillner et al. [12] make advancements on the question whether a constant number of vertices can remain at their start positions when untangling a nonplanar graph. The aforementioned paper by Kraaijer et al. [7], of which this thesis can be seen as a direct continuation of, also falls in the category of research motivated by the game Planarity. One thing to keep in mind is that the swap operation from Kraaijer et al. essentially only changes the edges around in a graph drawing, while leaving the vertices stationary. The implication of this is that not every nonplanar drawing of a planar graph can be made planar through swaps, since not all vertex placements allow this. However, since the rotation operation discussed in this thesis does change the vertex positions, Fary's theorem, which states that every planar graph has a straight line representation [16], might suddenly become relevant. While interesting, this thesis does not seek to answer the question if every nonplanar drawing of a planar graph can be made planar using only 90 degree edge rotations, but does yield more insight on how to answer this question.

Another related topic is that of graph drawing. Especially, literature pertaining to making graph drawings at least readable and optimising their aesthetic features. Purchase defines metrics for these aesthetics [10]. Tamassia et al. define, among other things, a comprehensive list of detailed criteria for good graph drawings [13]. A paper of van Kreveld discusses bold graph drawings [8], specifically describing metrics for which we can define good instances of such graphs.

1.2 Structure

The main body of this thesis consists of three parts: a "theoretical", a "experimental" and a "design" part. Each has its own section. Section 2 will handle all theoretical parts as well as any proofs and a certain construction found for the rotation operation. Section 3 shows and discusses experimental analyses performed during this research. This yields more properties of the rotation operation to showcase. Also included in this section is an exploratory analysis yielding unexpected and, as of now, unexplained results. Section 4 details a practical use for this research as rotation is used within a novel graph based puzzle game. For this game, more operations are introduced, as well as a new goal state alongside planarity. These are also briefly discussed. Furthermore, a user study was performed involving different control schemes. Section 5 summarises the results and discusses possible future research.

2 Fundamental properties of the rotation operation

The rotation operation in this context means rotating an edge. To be more specific, after an edge e is selected, both its endpoints p and q will rotate 90 degrees around middle point m of e. All adjacent edged will change as well, because their endpoints will have changed after a rotation, as long as the rotated edge has a length greater than zero. This can be done either in clockwise direction or counter-clockwise direction. Let e' be the resulting edge after rotating e and p' and q' be the endpoints of e'. Figure 1 illustrates a clockwise rotation on a single edge, Figure 2 shows rotations in both directions in a graph.

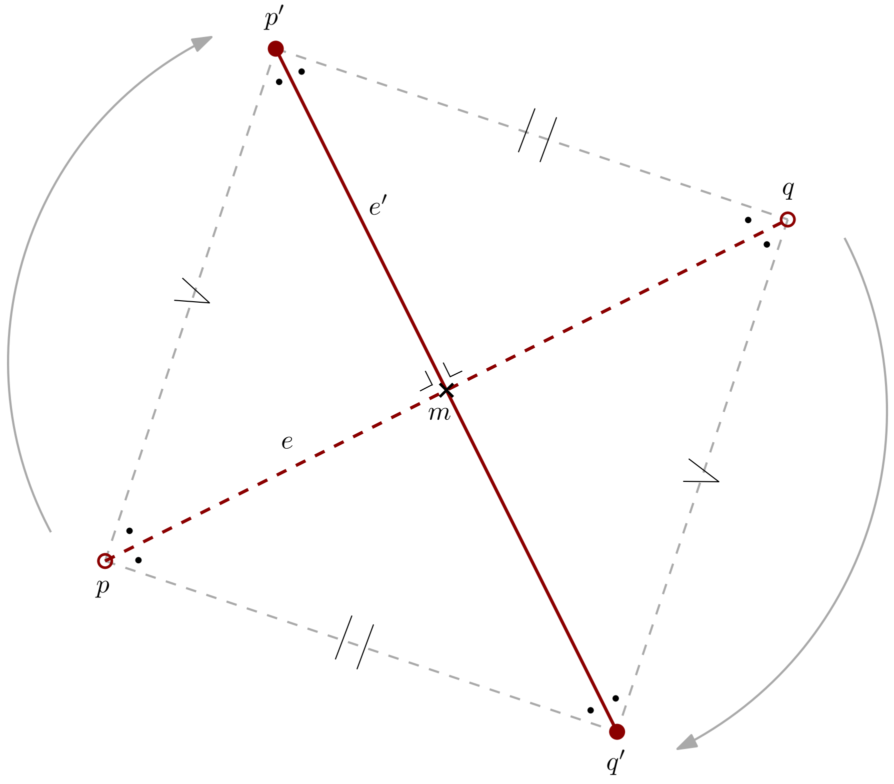


Figure 1: A rotation operation in clockwise direction. Endpoints p and q of edge e are rotated 90 degrees around m so that the resulting edge e' is perpendicular to e, and the endpoints become p' and q' respectively.

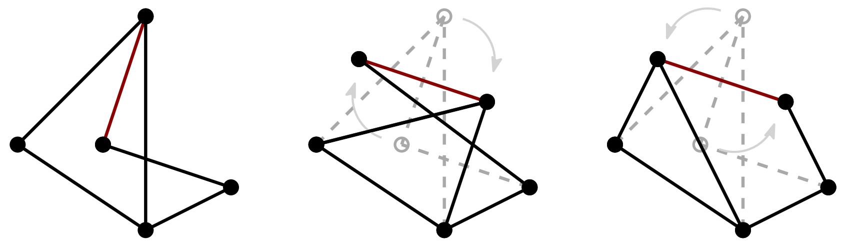


Figure 2: The rotation operation in both directions on an edge in a connected graph, where the left image shows the start graph with the edge to be rotated highlighted in red. The middle image shows the result of a clockwise rotation, the right image the result form a counter-clockwise rotation. Note that the graph became planar after rotating counter-clockwise, but not after a clockwise rotation.

In mathematical terms, after a rotation of points $p = (p_x, p_y)$ and $q = (q_x, q_y)$ around m to get p' and q' respectively, the new coordinates can be expressed as follows:

$$(p'_x, p'_y) = (\frac{p_x - p_y + q_x + q_y}{2}, \frac{p_x + p_y - q_x + q_y}{2})$$

and

$$(q'_x, q'_y) = (\frac{p_x + p_y + q_x - q_y}{2}, \frac{-p_x + p_y + q_x + q_y}{2})$$

This is also an efficient way to calculate a 90 degree rotation, as it does not involve any sine or cosine, which is why we used it for our implementations. To swap directions, simply swap p' and q'. Furthermore, the following holds true for this operation:

$$dist(p, p') = dist(p', q) = dist(q, q') = dist(q', p)$$

Another important observation is that, whenever an edge is rotated, only that edge and all adjacent edges are changed. In fact, as long as at least the rotating edge has a length greater than 0, the rotating edge and all adjacent edges are *always* changed after rotation. A logical consequence is that the order in which we do the rotations does not matter if and only if the edges that are rotated are not adjacent and each rotating edge has a length greater than zero.

The rotation operation can be done in two directions, both will yield the same set of positions that are occupied by vertices, but the edges will be different in most cases. There is one scenario where the choice between directions is irrelevant for edges as well. This is when both endpoints of the rotating edge are connected to the same vertices. In this case, rotating the edge either clockwise or counter-clockwise will result in the exact same graph drawing.

Since a swap is a subset of a 90-degree rotation, all of the properties proven in Kraaijer et al. [7] are relevant for this operation as well, although using twice the number of actions. Importantly, any graph that can be made planar using swaps, can also be made planar using rotations. This provides an upper bound for the number of rotations necessary to make any non-planar graph drawing planar that could also have been made planar using swaps only. Furthermore, using swaps we can move around edges in a connected graph without changing the vertex positions. This implies that we can achieve the same vertex positions with a path as with a complete graph.

2.1 The position of the centroid is constant

One characteristic of the rotation operation is that, no matter how many times it is used on the same connected graph, the graph cannot "move away". In other words, there is no translation possible of said graph. This is because the centroid remains unchanged after a rotation.

Theorem 1. *If two points in any point set with $n \geq 2$ points with centroid c are rotated around their middle point, c will not change its position.*

Proof. Suppose we have any point set with $n \geq 2$ points and the two points within this set to be rotated around their middle point are denoted by $p = (p_x, p_y)$ and $q = (q_x, q_y)$. Let $a = (a_x, a_y)$ be the sum of the coordinates of all other points, then before rotation the coordinates of the centroid can be written as:

$$(\frac{a_x + p_x + q_x}{n}, \frac{a_y + p_y + q_y}{n})$$

Assume that when rotating, the points are moved distance u horizontally, and v vertically. Then after rotation, the centroid can be expressed as follows:

$$(\frac{a_x + (p_x + u) + (q_x - u)}{n}, \frac{a_y + (p_y + v) + (q_y - v)}{n}) = (\frac{a_x + p_x + p_y}{n}, \frac{a_y + q_x + q_y}{n})$$

Which is the same as before rotation, meaning that the centroid remains at the same position. □

2.2 Sums of squared distances

Not only is it not possible to translate a graph using only rotations, a graph can also not be "scaled" infinitely. Where we measure "scaled" as the furthest vertex from the centroid m of the graph. This is because both the sum of squared distances between all pairs of vertices and the sum of all squares of the distances between each vertex and the centroid remain constant. These two properties are proved below, starting with the latter.

Theorem 2. *If two endpoints of an edge in any graph where $n \geq 2$ are rotated around the middle point of that edge, the sum of squared distances between each vertex of the graph and the centroid remains the same.*

Proof. It should be clear that the theorem holds true for any graph where the number of vertices is $n = 2$. Suppose we have a graph with $n \geq 3$ vertices (henceforth called points, as the same applies to any point set). Name the points to be rotated $p = (p_x, p_y)$ and $q = (q_x, q_y)$. After rotation, they are denoted as $p' = (p'_x, p'_y)$ and $q' = (q'_x, q'_y)$. Since the coordinate system can be placed however we want, the scenario can be rotated such that these four points are aligned with the x and y axes, as pictured in Figure 3. In this case the statements $p_x = p'_x$, $q_x = q'_x$, $p_y = q'_y$ and $p'_y = q_y$ hold true.

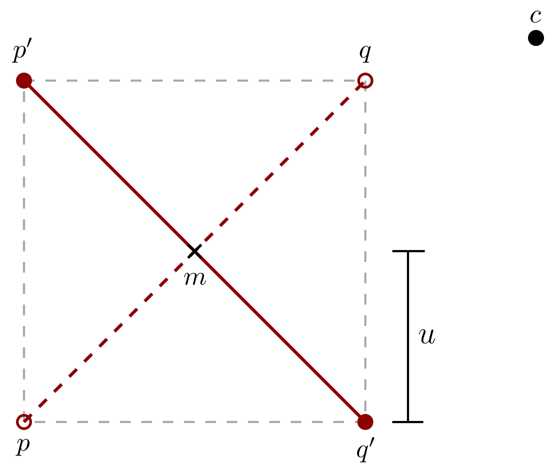


Figure 3: A rotation of two points, illustrating the main point of the proof of Theorem 2. Points p and q are endpoints of the original edge e, p' and q' respectively are the same points after rotation. Note that centroid c could have been placed anywhere, this does not matter for the proof. Keeping that in mind, since we have full range over the orientation, a rotation operation can always be drawn like this.

Now, denote the centroid of the point set as $c = (c_x, c_y)$, and the centroid of p, p', q and q' as $m(m_x, m_y)$. Let u be the distance between a point and its rotated counterpart, *divided by two.* The sum of distances of p and q to c before rotation can be written as:

$$\sqrt{(m_x - u - c_x)^2 + (m_y - u - c_y)^2} + \sqrt{(m_x + u - c_x)^2 + (m_y + u - c_y)^2}$$

Now let a be the sum of squared distances to c of all other points, then the total sum of all squared distances to c before rotation is the following:

$$a + (m_x - u - c_x)^2 + (m_y - u - c_y)^2 + (m_x + u - c_x)^2 + (m_y + u - c_y)^2$$

As shown in Theorem 1, the centroid remains at the same position after rotation. The sum of all squared distances to c after rotation is thus the following:

$$a + (m_x - u - c_x)^2 + (m_y + u - c_y)^2 + (m_x + u - c_x)^2 + (m_y - u - c_y)^2$$

This is exactly the same as the expression before rotation, meaning the total sum of squared distances between all points and the centroid is constant. □

Theorem 3. *If two endpoints of an edge in any graph where $n \geq 2$ are rotated around the middle point of that edge, the sum of squared distances between all pairs of vertices remains the same.*

Proof. It should be clear that this property holds true for point sets where $n = 2$. A known property of triangles is [2]:

$$dist(p_1, p_2)^2 + dist(p_2, p_3)^2 + dist(p_3, p_1)^2 = 3(dist(c, p_1)^2 + dist(c, p_2)^2 + dist(c, p_3)^2)$$

Where p_1, p_2, and p_3 are the corners of the triangle, and c is the centroid of the triangle. Since Theorem 2 shows that the right side of the equation above remains constant after rotation, the conclusion can be drawn that the left side also remains constant. This means that the current theorem holds true for all point sets where $n = 3$. Notice that for any point set where $n > 3$

can be seen as a collection of triangles (i.e. point sets where $n = 3$). The two endpoints of the edge that is rotated form triangles with all other points, to put it more precisely: they form $n - 2$ triangles. For all these triangles the right side of the equation remain constant, meaning that the left side remains constant as well. As implied by the fact that only adjacent edges of the rotating edge are changed, all edges that are not part of these triangles remain unchanged. From that we conclude that the sum of all squared distances between all pairs of vertices for any graph where $n \geq 2$ stays unchanged after rotation. □

2.3 Potential limit

Let us again assume a connected graph of n vertices. When performing the rotation operation randomly on edges and drawing all vertices of all instances with 2x2 pixel squares in their respective colours, a noticeable circle is approximated around the centroid. The maximum radius of this circle is the *limit*, in other words: the maximum distance from the centroid a vertex could potentially go. Because it is unproven whether every point within this circle will be reached by its single connected graph, we cannot say that the limit can always be reached (although it certainly seems this way, see Figure 4).

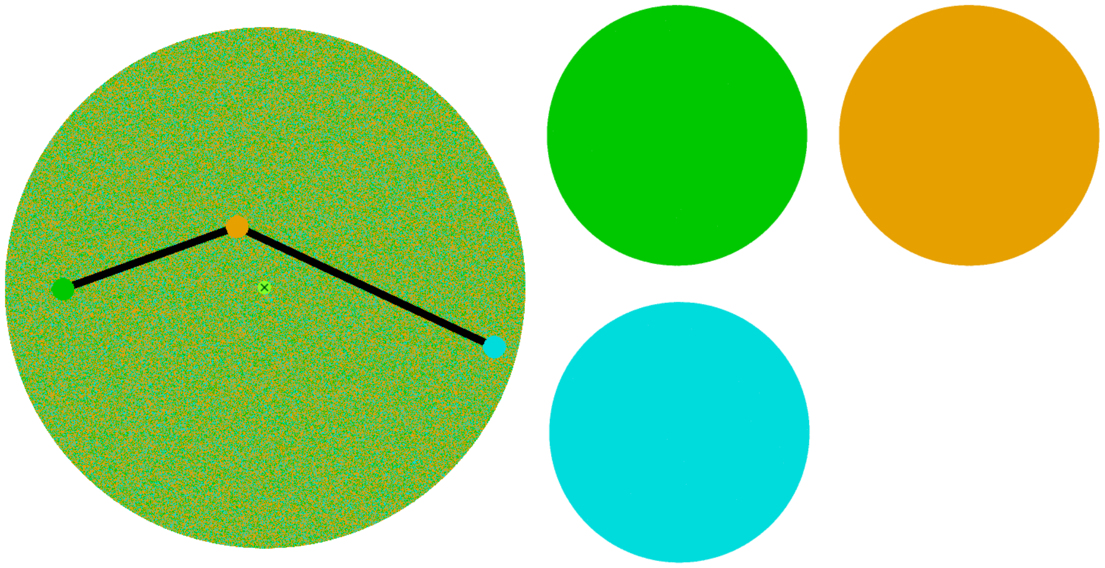


Figure 4: The results when doing clockwise rotations where the rotating edge is random each time, and we draw each vertex with a 2x2 pixel square with its respective colour each time. The black graph is the beginning state, the bright green circle in the middle with the cross is the centroid. The three circles on the right show what is drawn with only one vertex. Number of rotations: 9510191.

Since the sum of squared distances always stays the same, the absolute edge case that actually reaches this limit is where $n-1$ points are located at the exact same coordinates. In this special case the nth point lies on the limit. Furthermore, say that each vertex i and j both have x coordinates i_x and j_x respectively and y coordinates i_y and j_y respectively. Knowing this, the limit can be calculated of any graph with the following formula, using the sum of squared distances for all vertex pairs:

$$\frac{\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \sqrt{(i.x - j.x)^2 + (i.y - j.y)^2}}{n-1} * \frac{n-1}{n}$$

The formula above the line of the first fraction determines the total distance between all pairs of vertices. The part to the right of the multiplier is the offset for the centroid.

The same can be calculated in a similar fashion using the sum of squared distances between each point and the centroid:

$$\frac{\sum_{i=1}^{n} \sqrt{(i.x - m.x)^2 + (i.y - m.y)^2}}{n-1} * \frac{n-1}{n}$$

Since we do not know if the edge case can always be constructed from any graph drawing, using only rotation operations, we refer to this limit as the *potential* limit.

2.4 Quickest growth

Assume a circle C with middle point m and radius r. The question is how fast a graph can "grow" away from m through rotations, with the restriction that we can put vertices only within C. There is no limit on the number of vertices, and the resulting graph is a complete one. How "far away" the graph is, is measured by simply taking the distance between m and the vertex that is furthest away from m after rotation.

For one rotation it can be proven where the edge should be placed so that we get a vertex as far as possible from m.

Lemma 1. *Assume any circle C with radius r in which there is no limit on where and how many vertices can be placed. All vertices are connected with all others with edges. The upper bound of the distance between the furthest vertex from m and m after one rotation is $\sqrt{2}r$.*

Proof. Since this proof involves one rotation, we only have to focus on placing one edge anywhere in the circle. The situation is rotation-invariant, so we may restrict ourselves to horizontal edges, and we can also assume that the sought edge lies in the upper semi-circle. This proof will first show that we only have to look at edges that their endpoints on the boundary of C (i.e. a chord), then we show that a chord covering an arc of 90 degrees reaches the upper bound after rotation.

Assume any edge e_1 with both its endpoints on the boundary of C, then u is the distance between m and the furthest endpoint of e_1 after rotation. Let v be the distance between the furthest endpoint of any edge e_2 and m after rotation, where e_2 has at most one endpoint on the boundary of C and both edges are at the same height. Firstly, for any edge that has none of its endpoints on the boundary, there exists an edge that has the exact same middle point with at least one endpoint on the boundary of the circle that has a greater length, and will thus reach a greater distance from m after rotation. It is for this reason satisfactory to let e_2 be any edge with at least one endpoint on the boundary of C. Now let w be the distance between the two middle points of e_1 and e_2. Since e_1 has maximum length at this height (endpoints cannot go out of C), e_2 has $2w$ smaller length than e_1, meaning that the difference in distance of the middle point of e_2 and any of its endpoints is w. The situation thus far is shown in Figure 5. Next, we show that $v \le u$. Observe that v can be written as $\sqrt{(u-w)^2 + w^2}$. Note that:

$$\begin{aligned}
\sqrt{(u-w)^2 + w^2} &\le u \\
\Leftrightarrow (u-w)^2 + w^2 &\le u^2 \\
\Leftrightarrow u^2 - 2uw + 2w^2 &\le u^2 \\
\Leftrightarrow 2w^2 &\le 2uw \\
\Leftrightarrow w &\le u
\end{aligned}$$

Since $w \le u$ holds true in all cases, it follows that any chord will always reach the furthest distance from m after rotation compared to all other horizontal edges at the same height.

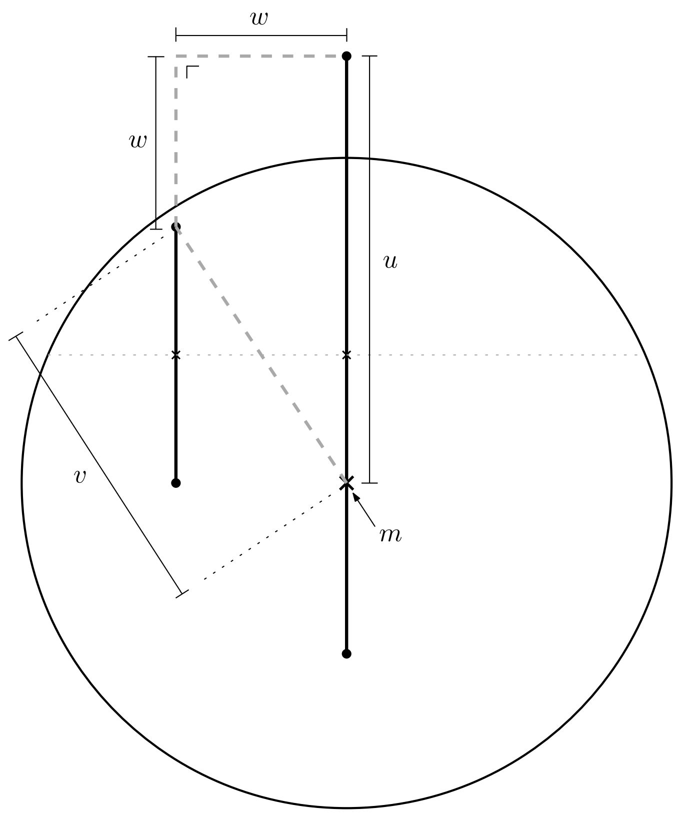


Figure 5: A visual of the first claim of the proof of Lemma 1 that shows that we only have to look at chords when proving this theorem.

Next we show that a chord with middle point m_e covering an arc of 90 degrees on C, yields the greatest distance from m after rotation. As the first part of this proof has shown, we only have to look at chords, meaning that m_e is at the same horizontal position as m. Let the length of a chord be $2b$, and the distance between m_e and m be a, as shown in Figure 6. Then the distance between the furthest endpoint after rotation and m is $a + b$. Let C be a unit circle and α be the angle, then a can be expressed as $\sin\alpha$ and b can be expressed as $\cos\alpha$. To achieve greatest distance after rotation, we want to maximise $a + b$ or rather, $max(\sin\alpha + \cos\alpha)$. This can be done through the following expressions:

$\frac{d}{d\alpha}(\sin\alpha + \cos\alpha) = 0$

$\Leftrightarrow \cos\alpha - \sin\alpha = 0$

This is true for $\alpha = \frac{\pi}{4}$, so 45 degrees. Since this only covers b, which is half of the chord, the actual arc is 90 degrees.

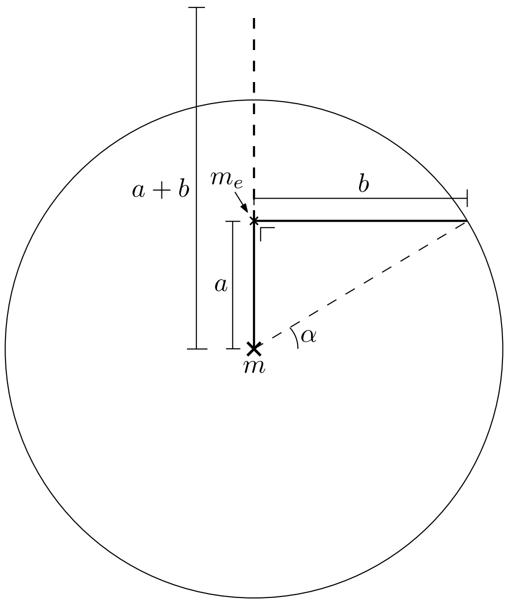


Figure 6: When moving an edge along a 90 degree triangle, the edge moves the same distance (u) as that its length decreases on one side (v).

After rotation, one endpoint will end up exactly at m, meaning that the furthest distance is equal to the length of the chord. This length is $2r * \sin \frac{0.5\pi}{2}$, which is $\sqrt{2}r$. □

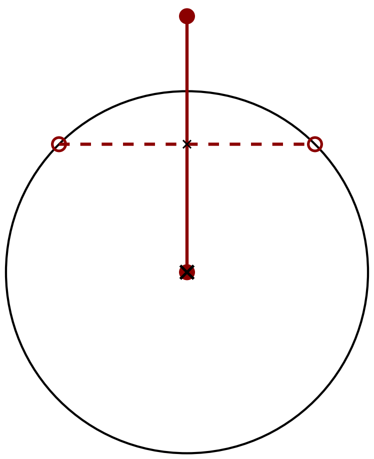


Figure 7: A construction showing where to place two connected vertices (dashed red line) so that the distance to the middle point of the circle is greatest after rotation (normal red line).

Lemma 1 outlines the first step of a construction and is shown in Figure 7. Notice that in order to gain maximum distance, the vertices are as far away possible (i.e. on the circle) and the distance of one of the endpoints to m of the rotated edge is minimised. Assume that the construction starts with two vertices and their edge (for this construction, always assume all vertices are connected with each other) as specified in Lemma 1. Let r be $\sqrt{0.5}$, then the distance of the furthest point p from m after rotation is 1. Next, a rotation should be done so that the distance covered by the third rotation is maximised. Imagine a circle C' after step 1 that goes through p and shares middle point m. Lemma 1 has already proven that a rotation with an edge covering a 90 degree arc is optimal. Since p is already on C', the second endpoint should be as pictured in Figure 8 (or on the exact opposite end of the circle). C' has radius 1, so using this construction, the third rotation reaches $1 * \sqrt{2} = \sqrt{2}$ distance. Figure 9 shows the first three rotations that together make steps 1 and 2. A step here refers to an increase in distance from m of the graph. Required were four vertices (two per start edge, note that two vertices overlap each other in the upper right), and three rotations (two for the same "first step" moves, one to get the final result, i.e. the second step).

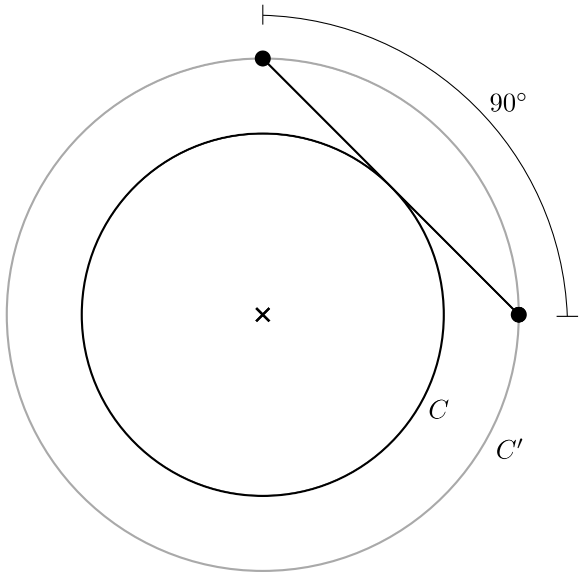


Figure 8: To expand from the second circle, we use the same setup as in Lemma 1, meaning we have two vertices with the same distance to m, but 90 degrees apart from each other.

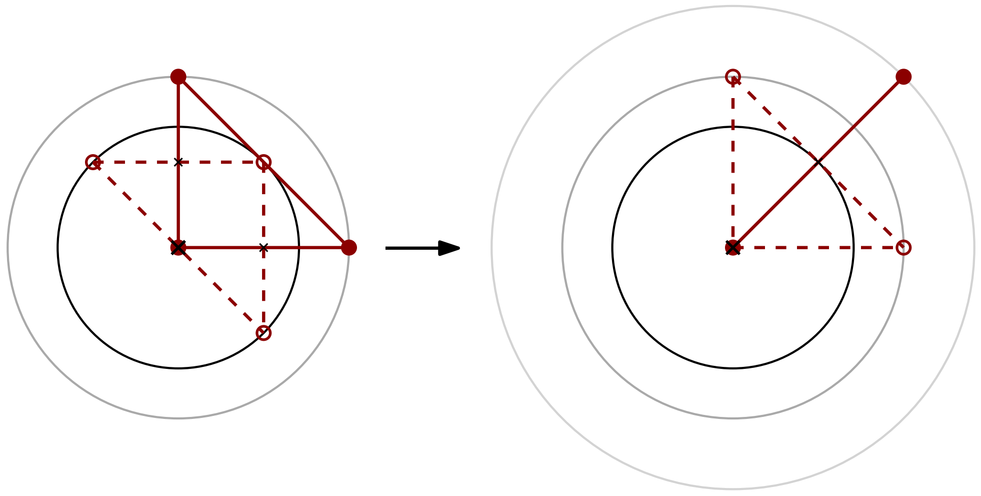


Figure 9: The first three rotations of the construction discussed in this section, where dashed red lines are before rotations, and normal red lines after. The left shows the result after doing the first step twice. The right shows the result of the second step (third rotation).

The exact same can be repeated, making this process recursive; repeating the first three moves, but 90 degrees away will once again yield the same construction as in Lemma 1, thus allowing another optimal gain in distance. This third step is illustrated in Figure 10. At this point, proving that a graph can grow arbitrarily far is mostly a formality.

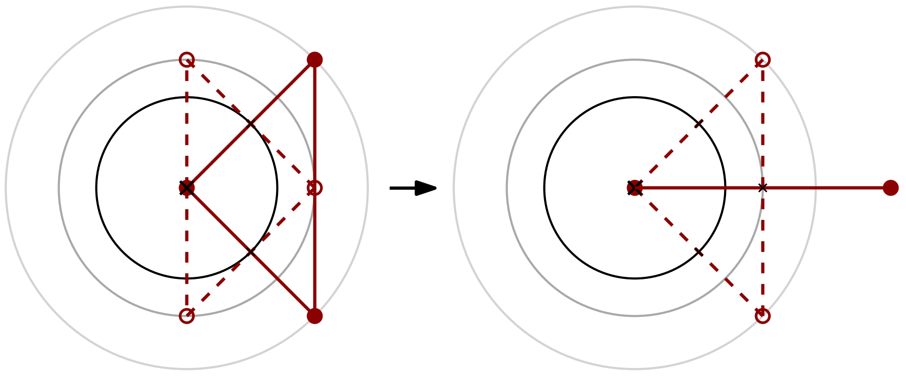


Figure 10: The process of Figure 9 is repeated to get the left image, although 90 degrees further in clockwise direction. A final rotation is performed on the rightmost edge to get the right result.

Theorem 4. *Assume that vertices can only be created within a determined circle C with middle point m. There is no limit on the number of vertices that can be placed and all vertices are connected to each other. Using only the 90 degree rotation operation, there is no bound on the distance between m and the furthest vertex from m.*

Proof. Now assume that through rotations we have some furthest point p located arbitrarily far away from m, outside of C. Since the beginning vertices are within C, it is always possible to do exactly the same as we did to get p, but 90 degrees further along the circle through p with middle point m. Then the step in Lemma 1 is once again possible, and the furthest distance can be multiplied by $\sqrt{2}$. Therefore, the furthest distance can always be increased using this construction. To be more precise: any furthest distance $dist(m, p)$ achieved through i steps, can always be multiplied by $\sqrt{2}$ by doing another $i + 1$ steps. □

This recursive process can be repeated infinitely, thus with this construction the graph can get infinitely far away, using an infinite number of vertices. For each step of the construction the number of vertices needs to be doubled, and twice the number of rotations plus one for the final step are required. To put this in equations, the number of vertices for each step is 2^{steps}, the number of rotations per step is $2^{steps} - 1$. The distance can be calculated as follows:

$$dist(m, p) = r * \sqrt{2}^{steps}$$

We believe that this construction is in fact, as the title of this subsection suggests, the fastest way to "grow" out of C. It is however hard to see how this should be proven, as this would imply showing that no other combination of rotations does not yield a more efficient method of growth.

3 Experimental analysis

To better understand how a graph of which the edges rotate 90 degrees behaves, several experiments were performed. The first experiments perform a large number of rotations randomly. A breadth first approach is introduced as well in Section 3.2 where we show how graphs behave regarding planarity. The section after that looks at behaviour related to “good” instances. Section 3.4 documents unexplained behaviour when doing the rotations in a predefined order.

3.1 Experiment: random rotations

In this experiment, clockwise 90 degree rotations are performed on random edges on randomly generated graphs. To generate a random graph, the first step of our method is to place the vertices randomly within our defined space. This is done as follows: n vertices are placed within a circle C with a radius r of 200 and middle point m, forming a point set P. The centroid c of P is calculated, after which all n vertices are shifted equally so that $c = m$. Afterwards, if any vertex is more than r away from m (i.e. the vertex is located outside of C), we start the process again. This is done until P is fully contained within C, while $c = m$.

For the experiments in this subsection we only look at paths and complete graphs. These two were chosen as they are the two extremes of a connected graph, meaning that with these we could potentially approximate both a lower bound as well as an upper bound. For paths the vertex that was placed first is simply connected to the second, the second to the third, and so on until the nth vertex is connected to vertex $n-1$. Note that complete graphs, while seemingly more complicated as they contain more edges, are actually quite easily optimised for rotations: since *every* vertex pair is connected with an edge, we do not have to keep track of any edges. An example of a successful generation of a path where $n = 3$ is shown in Figure 11.

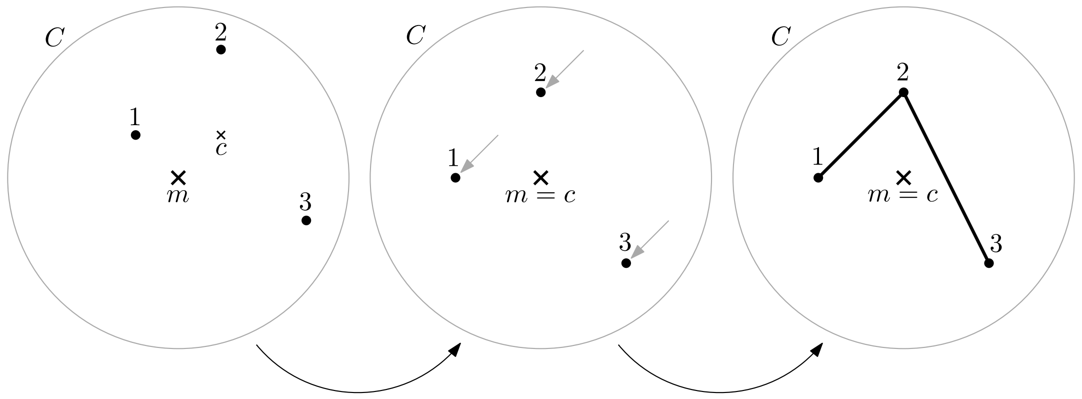


Figure 11: Generation of a path where $n = 3$, using the same process as used for the experimental analyses. Since all vertices are contained in C after shifting, the generation is successful and a path is created. The numbers of the vertices represent the order in which they happen to be placed in this case.

A graph is generated as explained above, and then for each rotation a random edge is selected and rotated clockwise. We decided on rotations in only one direction as this reduces repetitions (since a clockwise rotation is simply reverted by a counter-clockwise one directly after, and vice-versa). 1,000,000 rotations are executed on a graph, after which the next graph is generated and the same is done. This process is executed for graphs where $n = 3$ up to $n = 10$, where 100 graphs are generated per n. In other words, a total of 800 graphs were generated, and a total of 800,000,000 clockwise rotations were performed for only paths, then the same was done for complete graphs. We measure the largest distance between a vertex and m for each instance (the "furthest distance") and register when a new furthest distance is achieved, the average distance between vertices and m, and the average furthest distance. One final remark before continuing: despite using double floats, after enough rotations precision errors will emerge. Since these became quite apparent in the results, we decided to only count a furthest distance as greater if they are at least 0.02 greater than the previous. While precision errors are a hindrance for these kinds of experiments, they can also be regarded as some result, since they reveal some kind of lower bound for an average precision error one could expect on this scale. Unfortunately, this error could only be approximated on paths where $n = 3$, since only graphs of this small size seemed to achieve a further distance than possible (i.e. a further distance than the "potential" limit, as discussed in Section 2.3). On average, the amount of distance the furthest distance got further than the limit for both paths as well as complete graphs was 0.02 over limit, with a highest observed case of going 0.04 over the limit.

We refer to the average distance of all vertices of all instances and m as AD. The average *furthest* distance (i.e. the average distance to m of all vertices that were farthest away from m from each instance) as AFD. The results for each n are plotted in the graph in Figure 12.

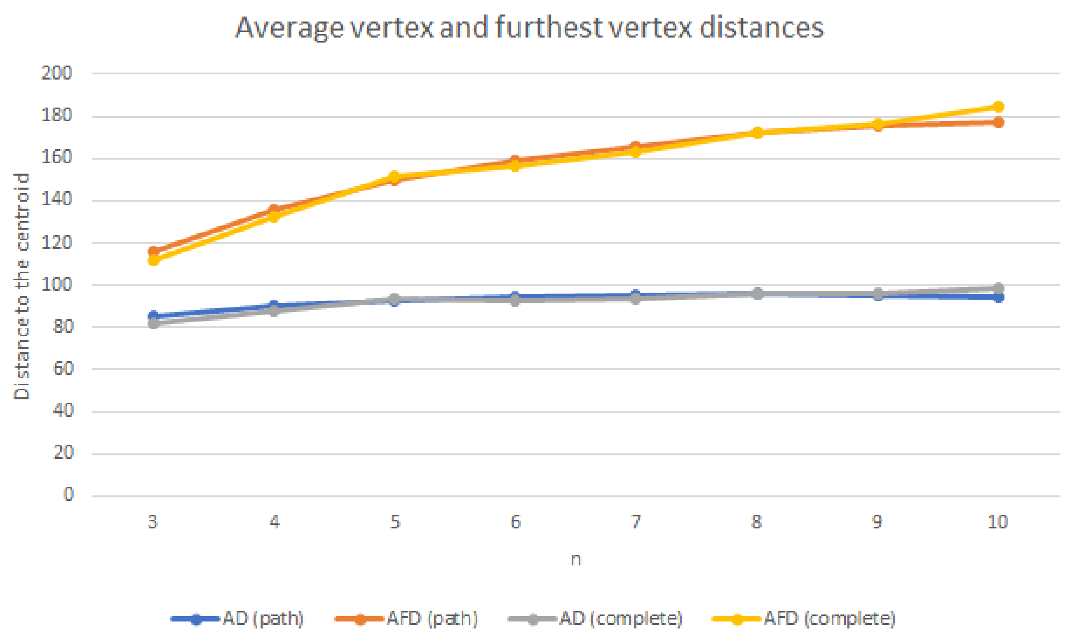


Figure 12: The average distances from m of all vertices (AD) and the average distances of the furthest vertex from each instance (AFD) are plotted for 100 randomly generated paths and 100 randomly generated complete graphs.

Perhaps more insightful are the attained average distances relative to the limit of the graph. In Table 1 the same results are shown, however for each graph we divided the distances by their limit. For completeness, we also included the average limit (AL) per n. The average distances and average furthest distances are plotted in the graph in Figure 13.

Average distances						
n	AD (path)	AFD (path)	AL (path)	AD (complete)	AFD (complete)	AL (complete)
3	0.67	0.91	127.77	0.67	0.91	123.36
4	0.53	0.80	168.58	0.53	0.80	164.24
5	0.46	0.74	203.20	0.46	0.74	205.35
6	0.41	0.69	231.40	0.41	0.69	228.06
7	0.37	0.65	256.84	0.37	0.65	252.49
8	0.34	0.61	280.38	0.34	0.61	280.31
9	0.32	0.59	299.87	0.32	0.59	301.29
10	0.30	0.56	314.50	0.30	0.56	327.46

Table 1: The average distances of all vertices, divided by the limit of each graph. The averages of these limits (AL) per n are also included.

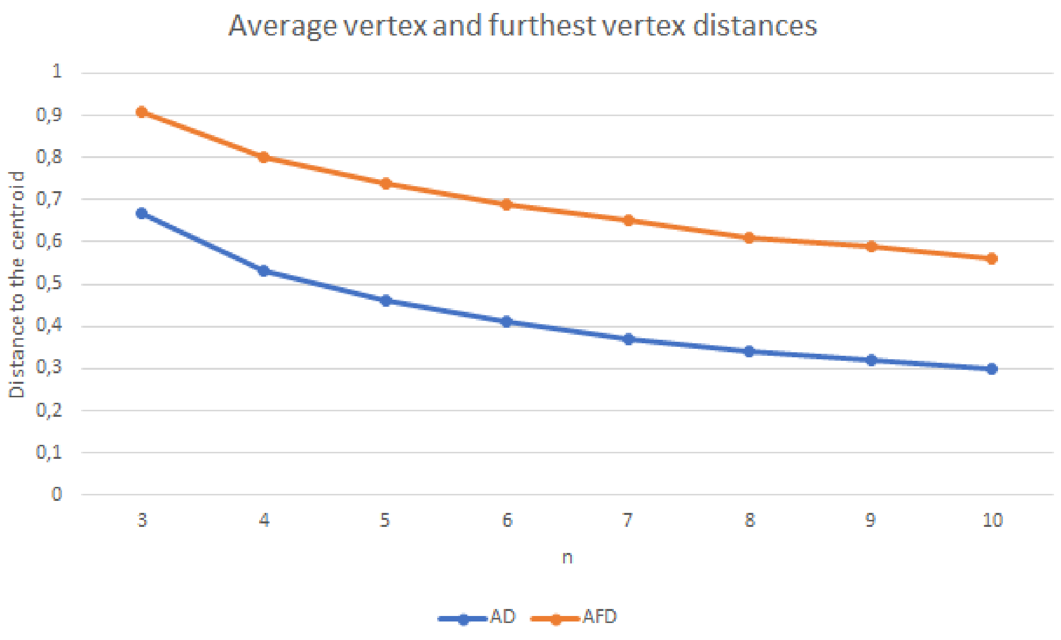


Figure 13: The same results as in 12, but divided by the limit of each graph. Since there was no difference between paths and complete graphs, we did not plot them separately.

As illustrated, the difference between paths and complete graphs are extremely minimal. In fact, while there is a difference shown in Figure 12, it is so small that it is more likely that this is due to the randomness introduced in this experiment. This is backed by the fact that the relative distances for paths and complete graphs are exactly the same in our experiment, as can be seen in the corresponding table. Interestingly, the absolute AD does not seem to increase significantly for larger n, while this does seem to be the trend for AFD. This is logical, as our randomly generated graphs are always within the same circle, so the AD should at least be the same for the starting instance and there is no reason to believe the starting position is not an "average" instance of that graph. The AFD increases, because introducing more vertices also means we increase the potential limit. The plots heading downwards for the relative distances might also be explainable, as for larger n it is less likely to get close to the limit. This is most likely because there are more edges, decreasing the chance that we do moves that get us closer to the limit. From Figure 12 it appears that, on average, the graphs do not move partly out of our starting area since the AFD stays below 200. The average limit however, already exceeds 200 for $n > 4$.

The claim that it is less likely to get close to the limit for higher n is backed in the following table and figure. The absolute furthest distance any vertex of any instance of a graph got was also tracked. This is once again shown in a table (Table 2) where the distances are divided by the limit of the corresponding graph, multiplied by a hundred. So if the proximity and the limit are the same, the result is 100. It is plotted in Figure 14.

Relative distances to limit		
n	Average proximity (path)	Average proximity (complete)
3	100.01	100.02
4	99.96	99.97
5	99.59	99.59
6	98.61	98.70
7	97.18	97.48
8	95.36	95.72
9	93.66	93.70
10	91.77	92.09

Table 2: The average closest relative distance all graphs of each n got after 1,000,000 rotations, where 100 is on the limit and zero is on m.

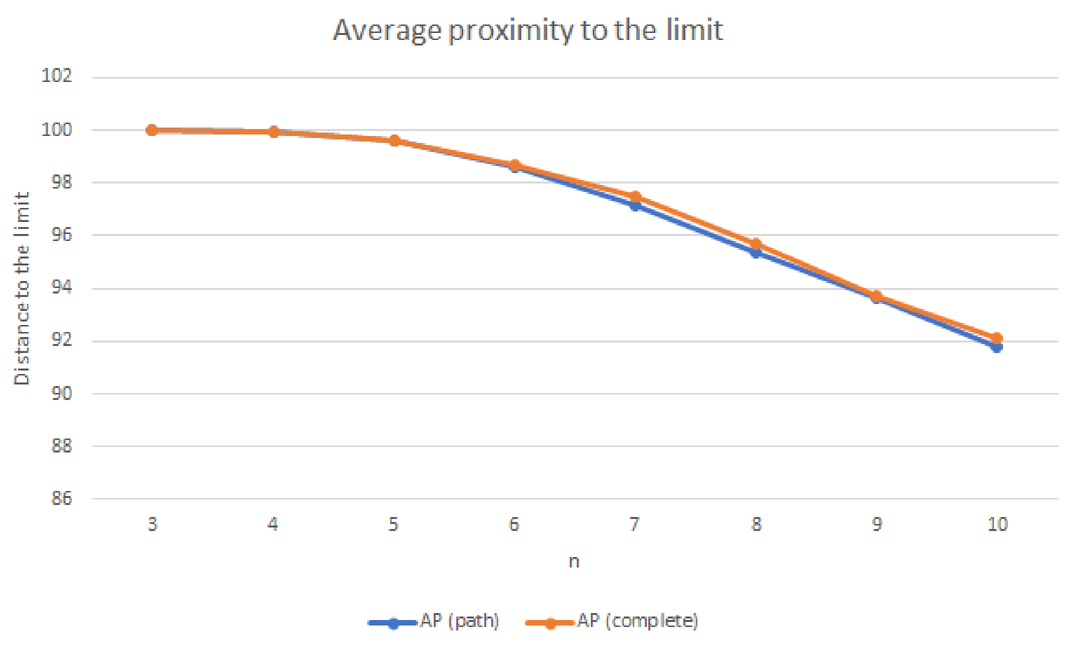


Figure 14: The closest average distance the graphs got to the limit, relative to the limit. Note that we zoomed in on this graph; the minimum distance on the y-axis is 86 instead of 0.

Once again, the slight difference between complete graphs and paths is most likely due to randomness, as there is no discernible trend and the differences are very small (a relative 0.38% between the two being the largest difference, namely for $n = 8$). Note that, while the decrease in our plot can be considered small, the fact that there is an almost 10 percent drop at $n = 10$ is actually significant, because the number of rotations for each graph is so large (1,000,000 clockwise rotations). Furthermore, because of the aforementioned precision errors, the average relative proximity for $n = 3$ is actually higher than 100; it is 100.01 for paths, and 100.02 for complete graphs.

3.2 Experiment: average planar solvability

It is worthwhile to gain more insight on how graphs behave when using the rotation operation regarding planarity, since this can be used in puzzle games. For this, we compare two approaches: a breadth-first approach, and a "random-first" approach. The latter is fairly simplistic: we simply perform 90 degree rotations on random edges until the graph is planar. This time, we do this both in clockwise, as well as in counter-clockwise direction. The choice between the two is also random, with a fifty percent for either. We restrict the maximum number of rotations for random-first search to 10,000. The breadth-first approach takes a graph and performs all possible moves on it. All resulting instances are then stored in memory. Suppose we have m edges, then there will be $2m$ instances afterwards. For the next level of depth we do the same for all those instances, etcetera. This is, once again, done until we find an instance that is planar. Because the breadth-first algorithm is exponential, we do not let it search farther than a depth of 6. Excluding this restriction in the current state of the algorithm would mean it might run for literal years, even for $n \leq 10$.

A graph obviously needs to be planar to be able to be solved. For simplicity sake, we decided to focus on paths only. We ran the random-first search as described above with randomly generated paths for $n = 4$ up to $n = 10$ and calculated the average number of moves it took to make each graph planar, including 0 (graph drawings that are generated planar). Our process of generating random graphs is still the same as in Figure 11, so within a circle with a radius of 200. The number of graphs per n is 1000. The results are plotted in Figure 15. The percentages of graphs that are generated planar (so instances that need 0 moves to be solved) per n are plotted in Figure 16.

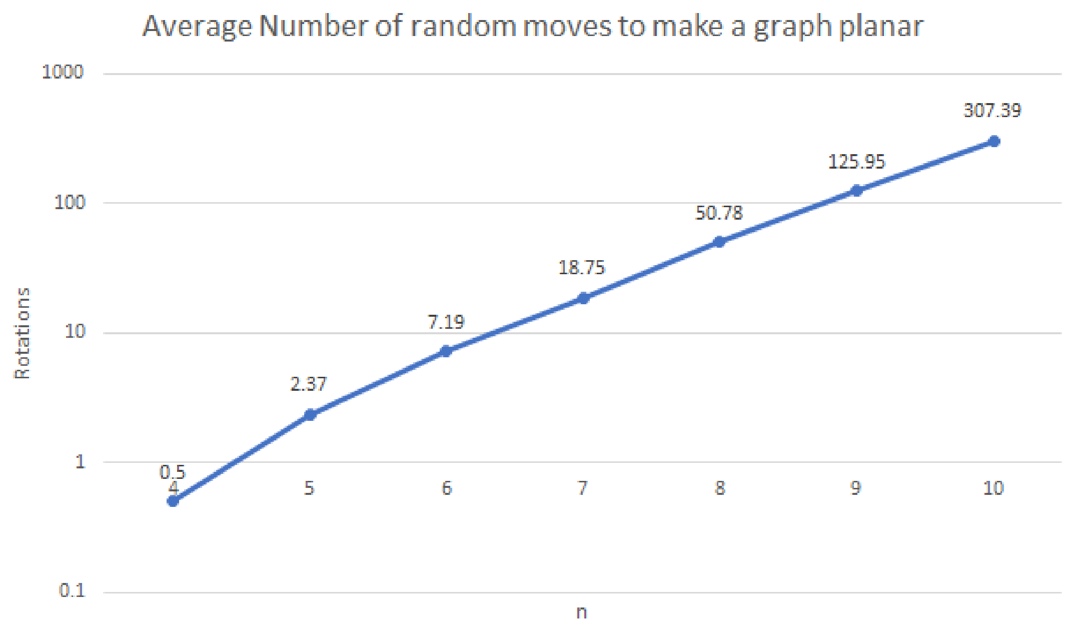


Figure 15: The average number of random rotations in both directions to solve paths of n vertices. Note that the we use a logarithmic scale for the y-axis.

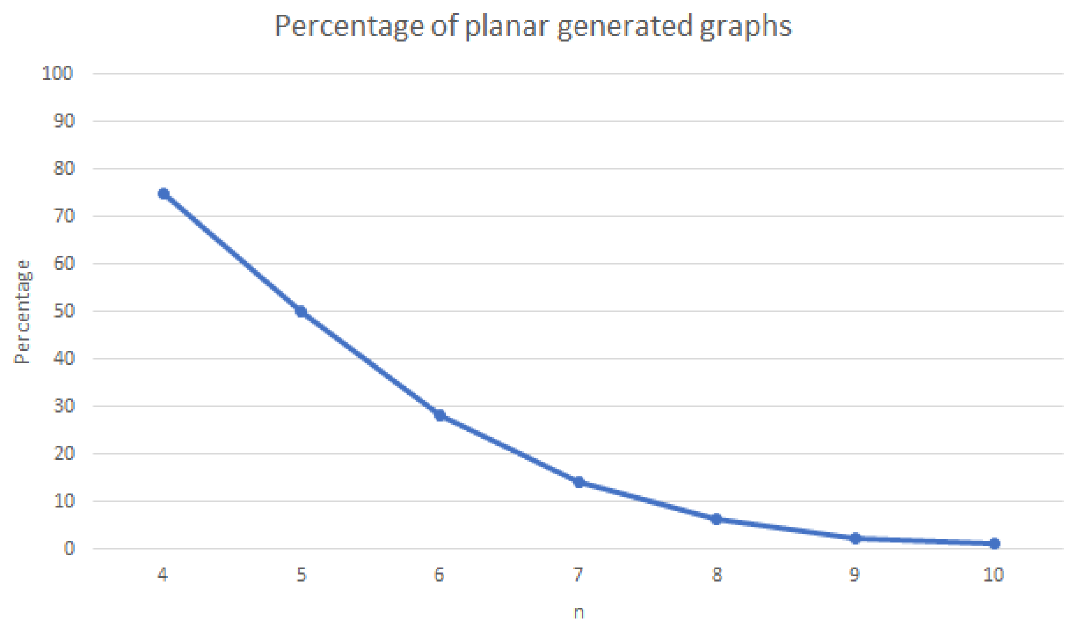


Figure 16: The percentage of paths that are generated planar for each n. The number of graphs generated per n is 1000.

To give an indication on the accuracy of our hypothesis that rotating in only one direction instead of both solves the graph faster, as it decreases the chance a move is immediately reverted, we did the exact same again but with only clockwise rotations. These results are plotted in Figure 17.

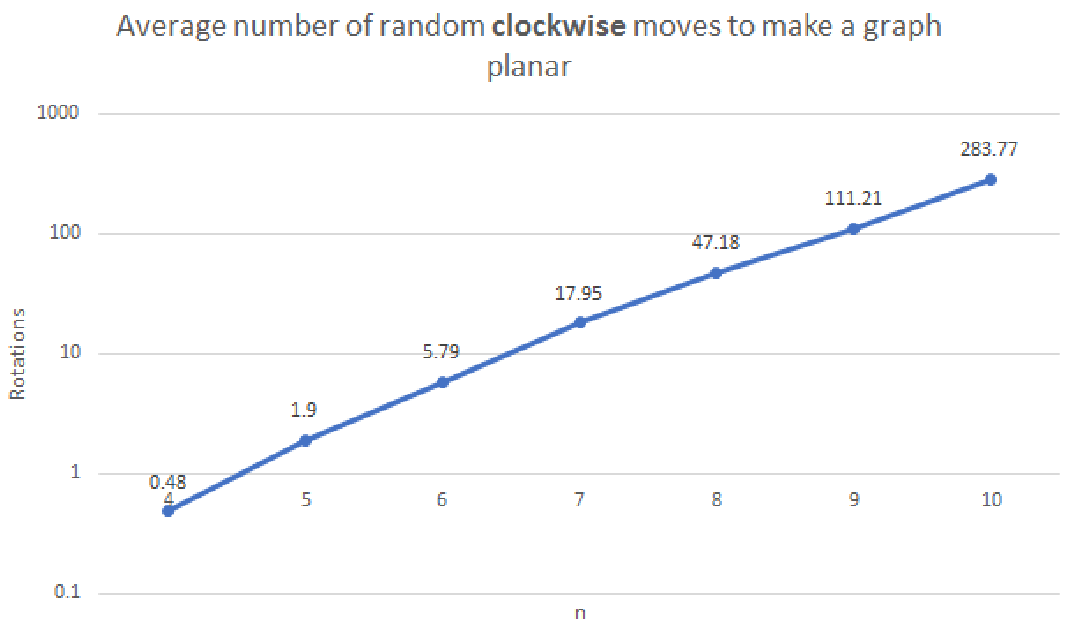


Figure 17: The average number of random clockwise rotations to solve paths of n vertices. Note that the we use a logarithmic scale for the y-axis.

Figure 15 and Figure 17 indeed show some difference. While they behave similarly, rotating in only one direction yields consistently fewer average rotations, which suggests this might be significant considering the large scale of the experiment. The obvious conclusion seems to be that yes, only rotating in one direction is indeed faster, but this difference is small. As a side note: the maximum of 10,000 rotations was never reached.

Directing our focus to the breadth-first approach, the problem of the algorithm being exponential becomes quite apparent when running it. As mentioned earlier, we cut it off at a depth higher than 6, but that means that we do not know the exact number of moves necessary to solve any graph that needs more than 6 rotations. We ran the breadth-first search for $n = 4$ up to $n = 10$, generating 100 random paths per n. Planar generated graphs count as 0 moves; the percentages of those graphs appearing is roughly the same as in Figure 16 (since we use the same method of generation). The number of graphs requiring more than 6 moves ("sevens") are plotted in Figure 18.

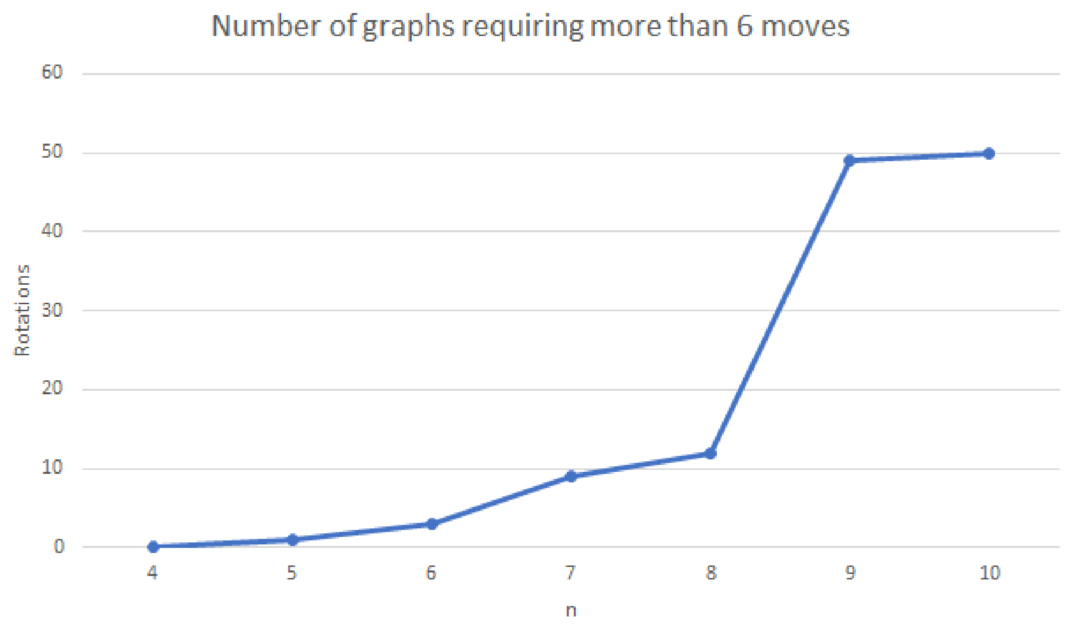


Figure 18: The number of paths found per n that required more than 6 rotations at minimum to solve.

Unfortunately, the results are less than satisfactory as we only generated 100 graphs per n, meaning randomness still plays a big factor. This is because the breadth-first algorithm in its current state simply takes too long to run to get better results. Nevertheless, some indications of possible behaviour were found. For $n \leq 8$ the number of sevens is ≤ 12, meaning that the averages are comparatively low. For $n \geq 9$ we cannot say much however, since roughly fifty percent of generated graphs are sevens, and there is no known upper bound. Interestingly, the number of sevens sharply rises between $n = 8$ and $n = 9$, but remains almost the same between $n = 9$ and $n = 10$.

The difference between random rotations and breadth-first seems large enough (for example for $n = 8$, where the number of sevens is relatively low) that we can say that the solving the puzzle based on thought-out decisions by a human is more efficient than doing random rotations.

3.3 Experiment: very good instances

Up until now we have not checked whether an instance is visually "good". An instance is good when it satisfies certain criteria drawn from graph drawing literature. We focus on a few criteria that should at least make instances playable when drawn in a puzzle context [8], [10] The criteria we will maintain are:

- The minimum angle between edges leaving a vertex should at least be δ for some $\delta > 0$.
- Any two vertices should at least be some distance d apart from each other.

We generate 1000 graphs per n and run our breadth-first algorithm. However, instead of stopping when we find the minimum rotations to make the graph planar, we stop when the graph is not good for a certain δ and d. Let a "very good" instance be one that needs at least 2 moves to become not good. Then the percentage of instances generated that are very good per n is plotted in Figure 19.

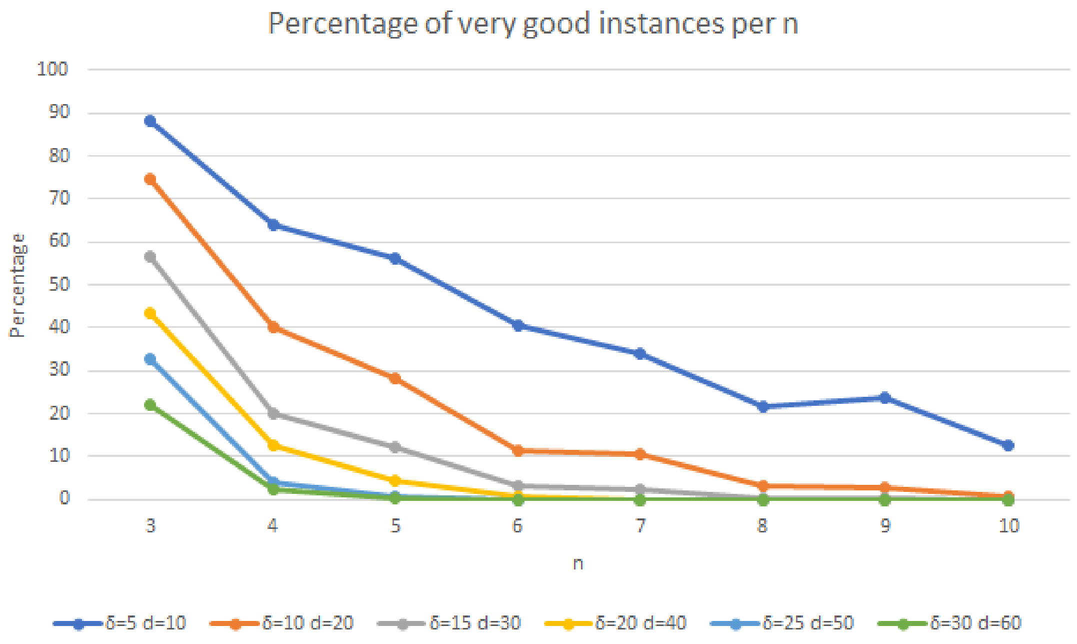


Figure 19: Percentage of very good instances found per n for different minimums of allowed angles between edges (δ) and minimums of distance between vertices (d), where 1000 graphs were generated per n.

3.4 Exploratory analysis on rotations in a certain order

Through sheer curiosity and coincidence, interesting behaviour was found when instead of rotating edges *randomly* at a large scale, we rotate them *sequentially*. The process is the following: assume any connected graph with edges e_1 through e_m. First e_1 is rotated clockwise, then e_2 in the same direction all the way to e_m. Then, when e_m has been rotated, the algorithm jumps back to e_1. This process is repeated until the program is halted manually.

Before continuing, some specifications on how the screenshots were made. In the figures, the original graph is shown with bold black edges and bold vertices with different colours. The centroid of all instances is shown as a light green disk, with a size smaller than that of the vertices, with a darker green cross in the middle. The number of rotations for each instance will usually be some number that seems arbitrary; this is because the numbers are indeed very much arbitrary. The process stops manually, but since this subsection is based on observations, and the results are abstract images anyway, round numbers such as 10,000 would not be any less arbitrary. The number of rotations will be referred to in each figure with "R".

Doing the sequential rotation process on a path of 3 randomly placed vertices, then drawing all edges of all instances in colours with a width of 1 pixel yields the result in Figure 20 after 323 rotations, and results in Figure 21 after 10,028 rotations. The colour of the edges are determined as follows: the program starts by drawing the edges in all red, $rgb(255, 0, 0)$ in rgb terms. The next instance after a rotation decreases the value for red by 1, the one after that increases blue by 1 and so forth until $rgb(0, 0, 255)$ is reached, at which point we gradually go back to $rgb(255, 0, 0)$ in the same manner. An apparent flower-like pattern shows when doing this with paths of three vertices, so there is clearly something happening here. The first edges to be rotated in the two figures below are the leftmost ones.

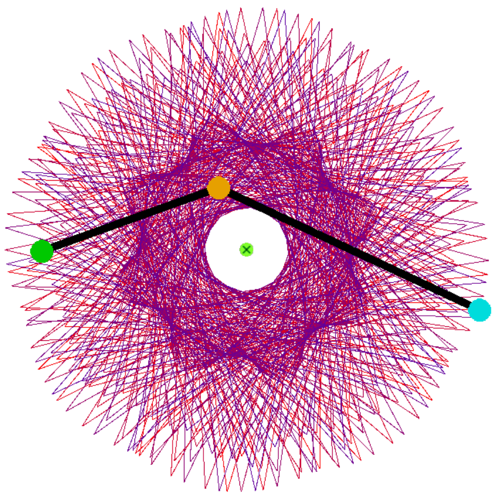


Figure 20: All edges of each instances are drawn. $R = 323$

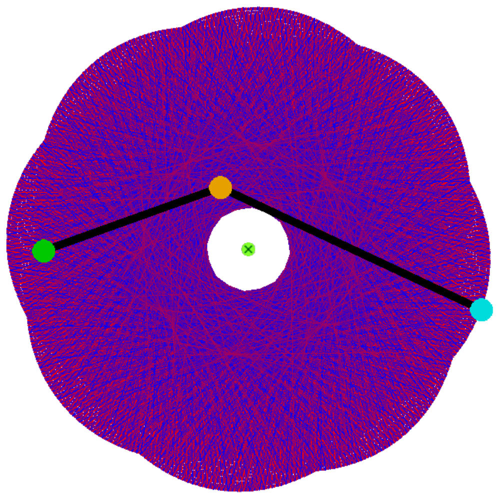


Figure 21: All edges of each instance are drawn. $R = 10,028$

We can expose even more of the actual patterns by only drawing the vertices. For all images from now on in this subsection, every instance is drawn by only drawing 2x2 pixel coloured squares at each position of each vertex. The colour of the square corresponds to the colour of the vertex. Doing the exact same, but only drawing the vertex positions of all instances in this manner with the previous graph, already gives a good impression of what actually happens. The results are shown in Figure 22, the number of rotations is 10,001.

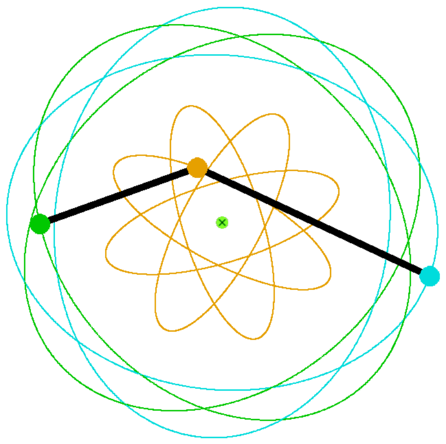


Figure 22: The graph is sequentially rotated and only the vertices of each instance is drawn with 2x2 squares of the same colours. The first edge to be rotated is the leftmost one. $R = 10,001$

Next we look at what happens when we change the angle between the two edges. Figure 23 shows eight different paths of three. Each path has two equally long edges, and the first (leftmost) edge of each is the exact same. Only the angle of the second edge is different, increasing by 45 degrees each sub image.

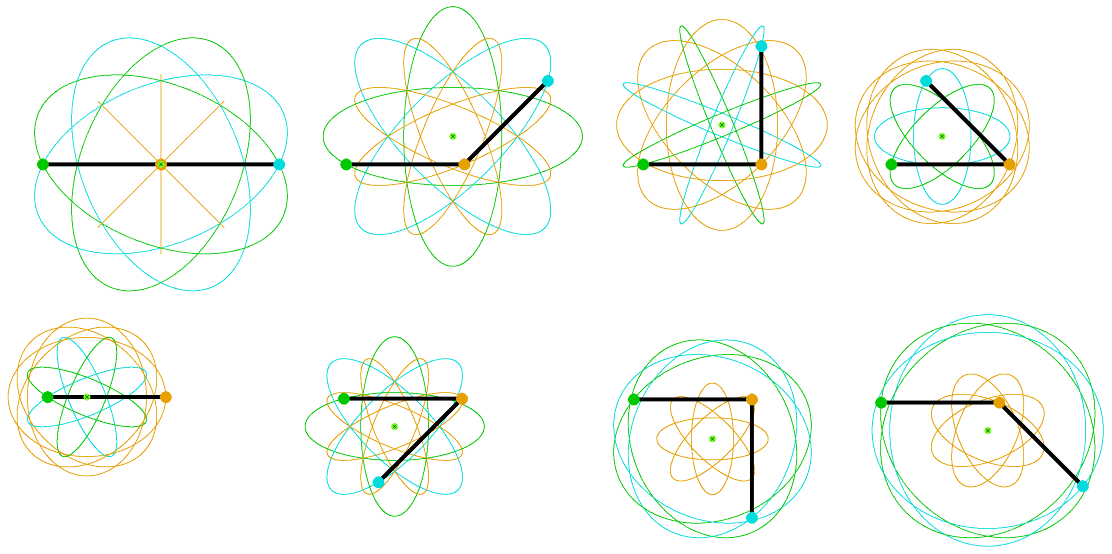


Figure 23: Eight graphs where the leftmost edge is the exact same for all graphs, and where both edges always have the same length. The only difference is the angle between the two, which increases by 45 degrees each sub-image. Note that the sequence always starts with the leftmost edge. For all graphs, $10,000 < R < 10,050$.

One interesting observation to make when even trying more paths of three and angles, is that the relation between the angle and the resulting shape appears continuous. Increasing the length of an edge does not to seem to do any more than increase the size of the result.

Looking at the results of at least a hundred more paths with three vertices, all of them appear isomorphic. In this case: the middle vertex forms a set of 4 equally sized ovals. Half of the other set of 4 equally sized ovals is formed by one of the two degree-1 vertices, the other half by the other. Inspecting individual rotations reveals more behaviour: after every rotation the middle (orange) vertex jumps to another of its own ovals in a sequential manner. The same applies to the outer vertices, but since they only move during half the rotations, it takes two rotations to move each of them. This behaviour is shown when only drawing after every four rotations. The visual after doing this with the same graph from before with 10,058 rotations is shown in Figure 24.

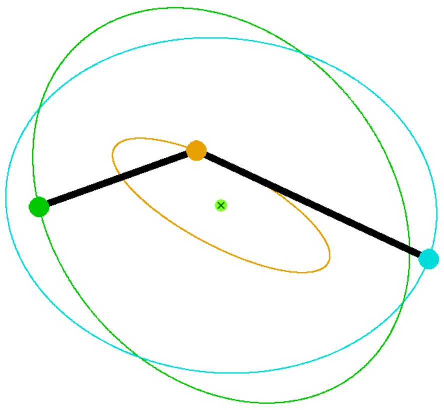


Figure 24: The exact same was done as when creating Figure 22, however now only every fourth instance is actually drawn, starting with the first. $R = 10,058$.

A logical next question could be what happens when instead of a path, a complete graph with three vertices is rotated this way. Sequentially rotating complete graphs with three vertices shows results that at first glance seem isomorphic to the ones we already had. A closer look shows that all ovals, of which there are now more, are now shared between all vertices. This is shown with the usual example graph, which has been made complete by adding a third edge, in Figure 25.

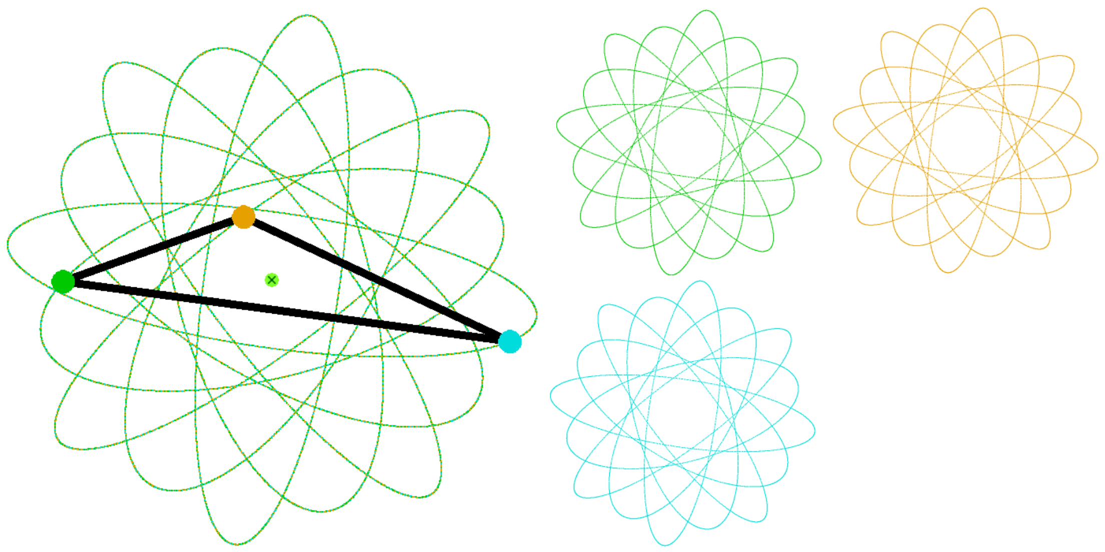


Figure 25: The resulting visual when sequentially rotating a complete $n = 3$ graph. The left image shows the figure in its entirety, all other sub-images only show what is drawn by the vertex with the corresponding colour. The newly added edge is rotated last. $R = 10,027$

Moving on to graphs where $n > 3$, since the order in which the edges are rotated are now relevant, all figures showing the graphs will display numbers showing this order. Most figures will now also contain the results without the graphs drawn, and only the visuals from each distinct vertex, as was previously done in Figure 25. Results obtained from a cycle of a $n = 4$

graph with a specific order and 50,034 rotations are shown in Figure 26. For reference, Figure 27 shows a result of rotating a random edge every time.

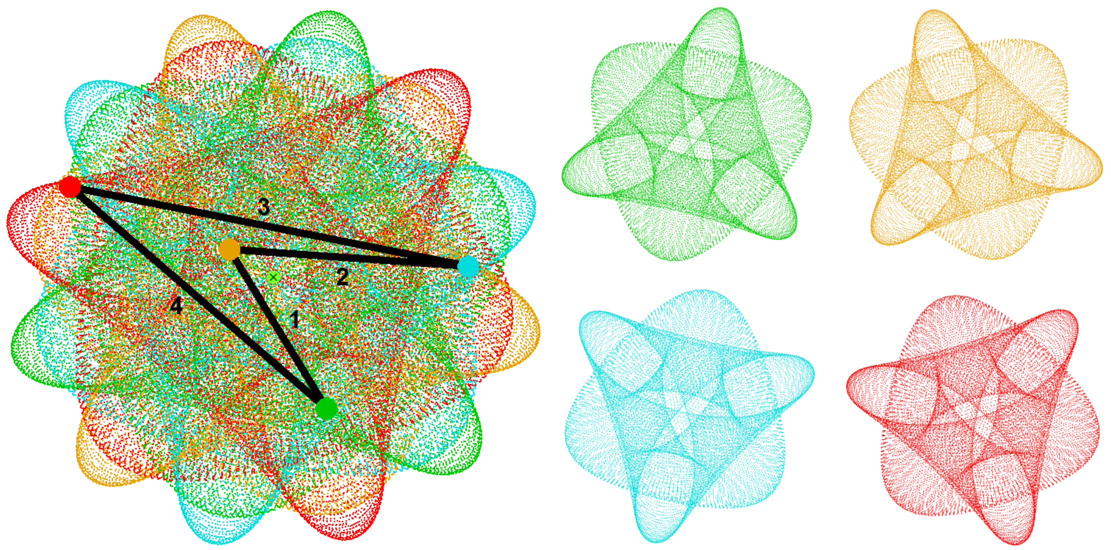


Figure 26: Results of sequential rotations on a specific $n = 4$ graph. $R = 50,034$

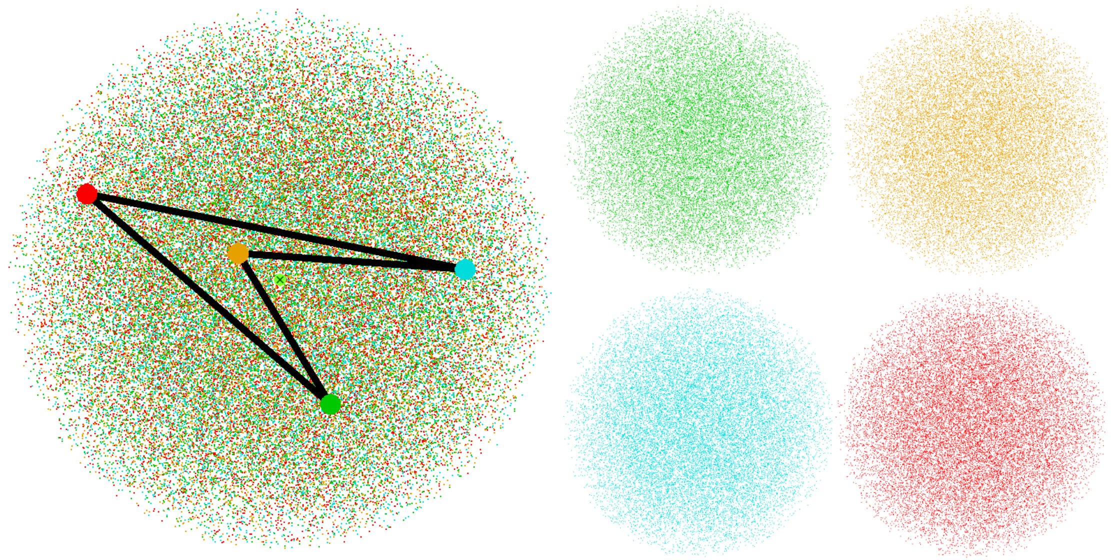


Figure 27: The exact same graph as in Figure 26, but rotations are performed randomly. Added as a point of reference. $R = 50,034$. Figure 4 shows a result of the same, but with $R = 9,510,191$

There is only one other unique order in which the edges can be rotated for a cycle of 4, this is shown in Figure 28. Remarkably, the number of resulting unique instances for this graph and rotation order seem finite. To be more precise: after eight rotations, the graph appears the exact same as the original, although each vertex seems to have moved two vertex positions further. After eight more rotations, we once again arrive at the original positions. In fact, because it is hard to say whether the previous experiments had finitely many or infinitely many positions, while this graph and rotation order results in such a small number of positions, we say its set of positions is *very finite*. In this case, it is a very finite set of 16. More rotation orders with very finite sets that were found are shown in Figure 29. This is by no means an exhaustive collection of such cases; these are just the ones that were found during this research and serve as examples.

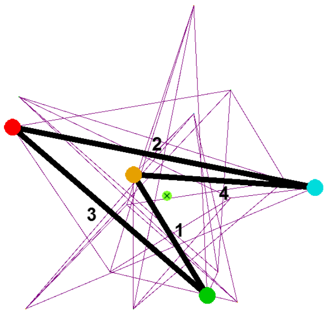


Figure 28: Sequentially rotating a cycle of $n = 4$ in this order reveals a very finite set of 16 unique instances.

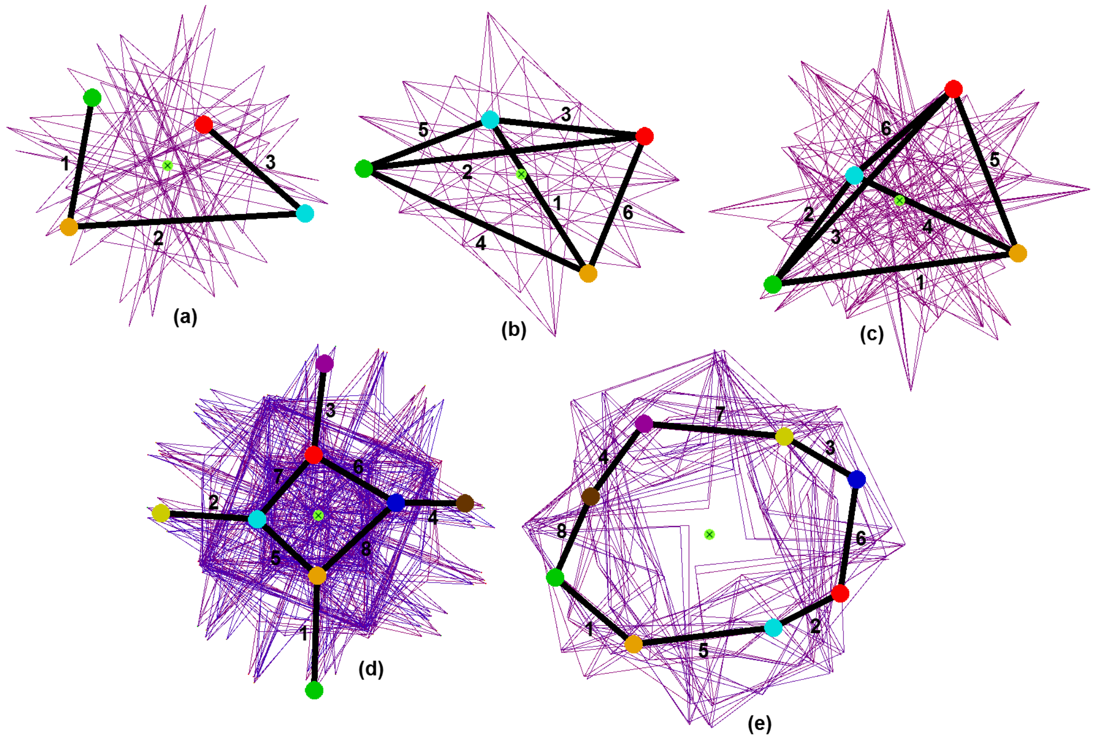


Figure 29: Graph and rotation order pairs with very finite sets of unique instances. (a) has 36, (b) has 12, (c) has 36, (d) has 192, and (e) has 96 different instances. Keep in mind that the positioning of the vertices is irrelevant, as long as the graphs are isomorphic and the rotation order is the same.

So far, our rotation orders had the implicit restriction that every edge should only be rotated once every sequence. However, expanding this sequence by repeating edges shows interesting results as well. Going back to the same graph as in Figure 22, but rotating the second edge twice, yields the results in Figure 30. We have also added repetition to the rotation order of the graph in Figure 26. Those results can be found in Figure 31.

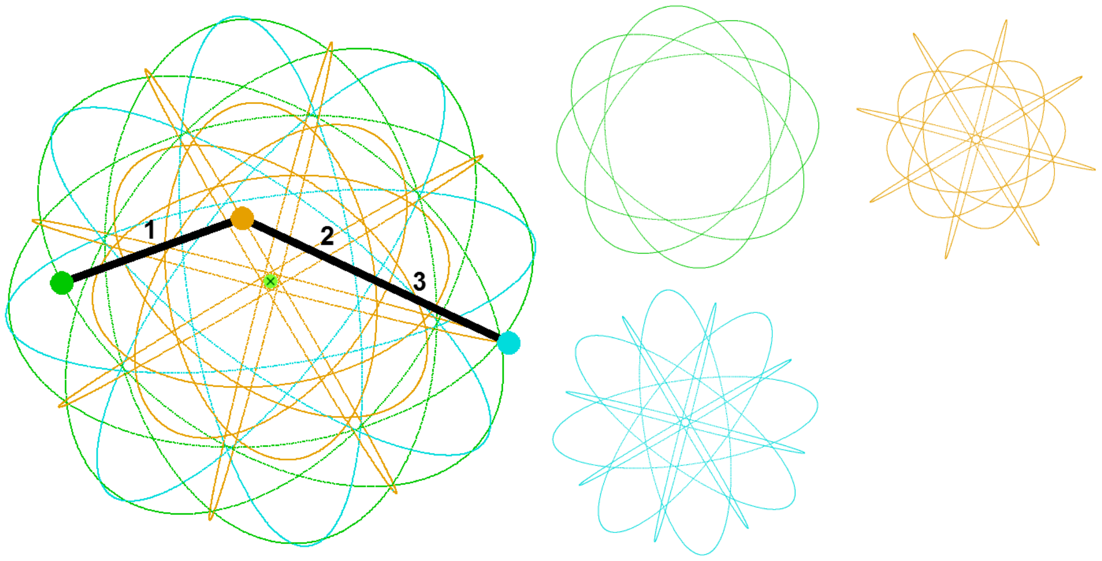


Figure 30: The graph from Figure 22 is rotated in a fixed order. This time, an edge is repeated once during the sequence. $R = 10,031$

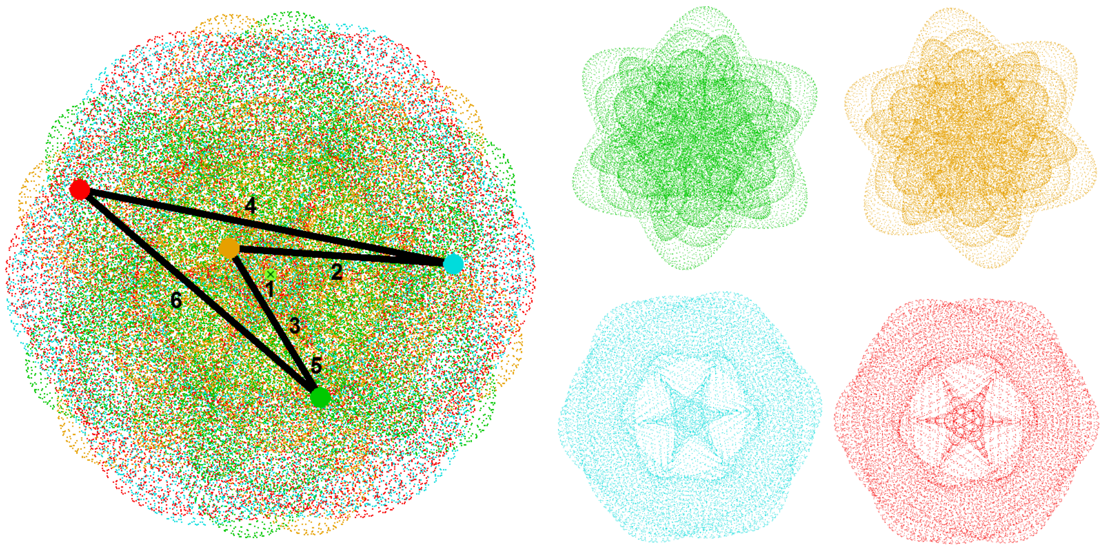


Figure 31: The graph from Figure 26 is rotated in a fixed order. This time, an edge is repeated twice during the sequence. $R = 50,034$

One final variable we examined is the angle with which we rotate the edges. So far, in line with the topic of this thesis, we have only rotated edges 90 degrees. Figure 32 and Figure 33 show some results when we take previous graphs and rotation orders, but change the rotation angle.

The results shown in this subsection are only a subset of what was found. More results that we find are visually interesting can be found in Appendix A. The coordinates of the vertices for the two main example graphs in this subsection can also be found there.

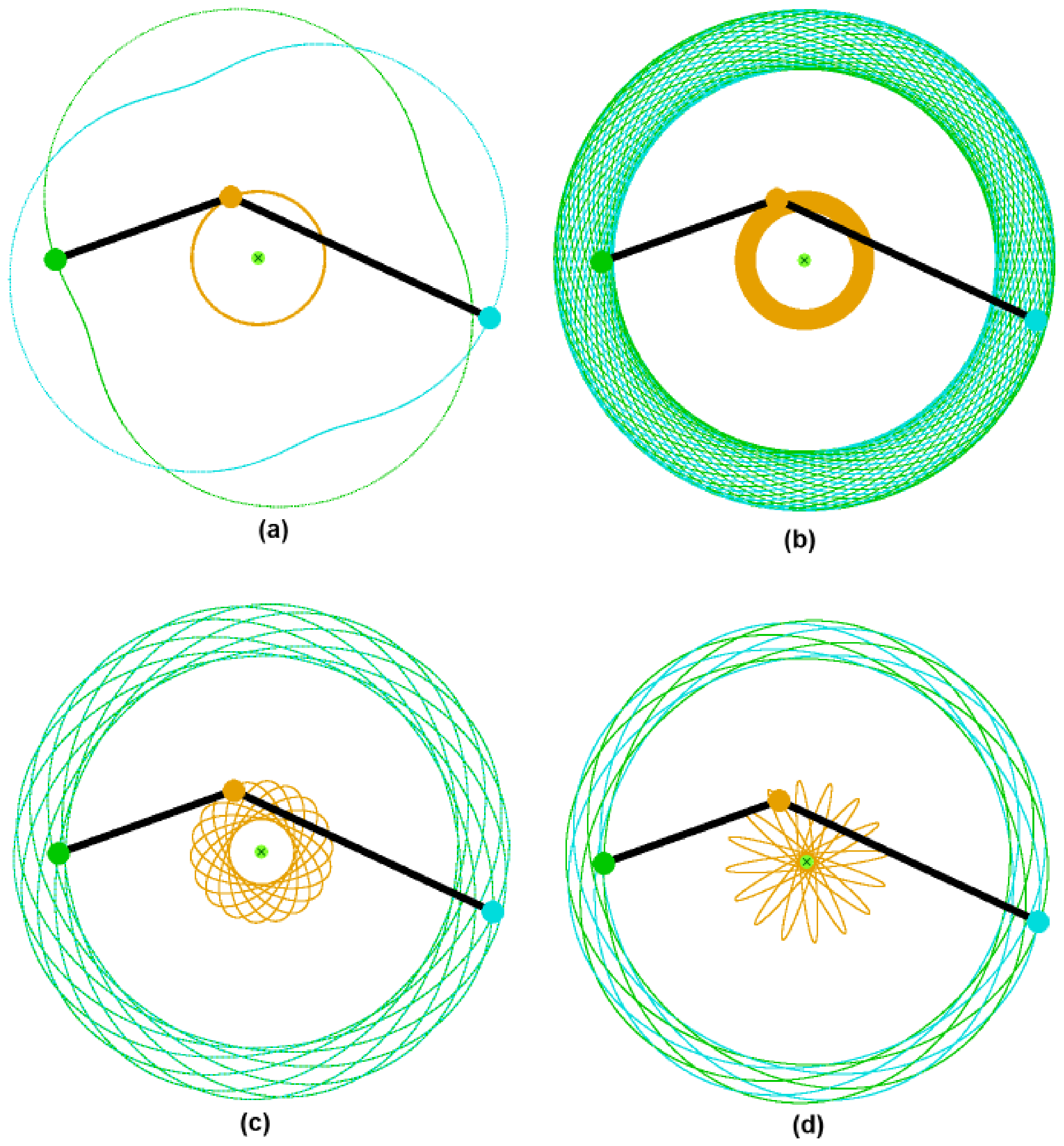


Figure 32: We take the same graph as in Figure 22 and do the same process of rotating edges, but change the rotation angles. (a) has a rotation angle of 1 degree, (b) 10 degrees, (c) 20, and (d) 45. Remarkably, for (a), (b) and (d) both lines that are formed from the green and the blue vertices appear completely separate (although this might be difficult to spot in this image), while in (c) they seem to completely overlap. For all results, $40,000 < R < 40,050$

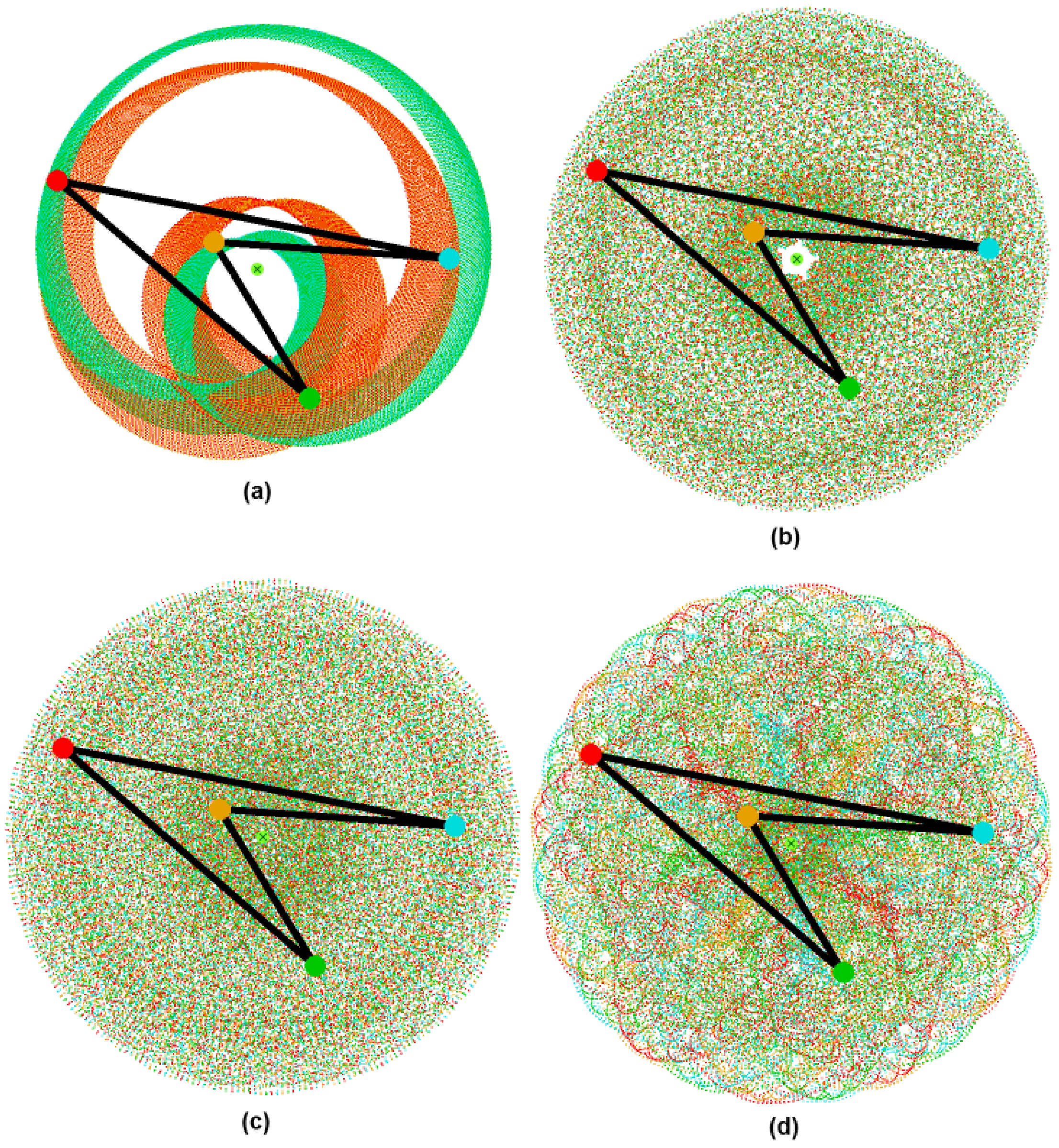


Figure 33: We take the same graph as in Figure 26 and do the same process of rotating edges, but change the rotation angles. (a) has a rotation angle of 1 degree, (b) 10 degrees, (c) 20, and (d) 45. For all results, $40,000 < R < 40,050$. Note that for this number of rotations (which we maintained to be consistent with Figure 32), (a) seems not close to being "complete", as it appears to still be filling up a full circle shape just before we halted the program. We also suspect that more patterns might be apparent when more rotations are done for at least (b) and (c).

4 The game

The rotation operation can be used as a game mechanic in puzzle games. This section details a puzzle game developed alongside this research. This game contains four different operations that can be done by the player. One of them is, as expected, rotation, which we already discussed in great detail. The next subsection will discuss the other three operations in a brief manner. Afterwards, the game and its mechanics will be discussed. Finally, the last subsection covers a user study that was conducted in order to gain a better understanding as to what sort of control schemes would be best for certain operations.

4.1 Basic properties of other moves

Three more move types will briefly be discussed, along with a quick analysis on how they interact with each other. For this subsection, assume that if two edges end up exactly the same (i.e. their respective endpoints are at the exact same locations), one of them will be removed. An edge will also be removed when it has length zero, and if two vertices end up at the same position one will be removed. Specifically which one, is irrelevant whenever an edge or vertex is removed.

4.1.1 Collapse

The Collapse move originates from Kraaijer et al. [7] where it was briefly mentioned as a possible variant of Swap. After selecting an edge e, its endpoints p_1 and p_2 will merge into one vertex p_3 located at the centre of e. Any edge with either p_1 or p_2 as an endpoint, will now instead have p_3 as an endpoint. The edge e will disappear. Assume any connected graph with m edges and n vertices. Then with only using the collapse move, there are m possible first moves. After $n-1$ moves, the number of edges will become zero, meaning that no more moves are possible at that moment. More than one edge per move might be removed, because if both p_1 and p_2 are connected to the same vertex q, then not only will e be deleted after Collapse, but $edge(p_1, q)$ and $edge(p_2, q)$ will merge into one. Since any connected graph reduces to just one vertex after $n-1$ moves, a graph can always be made planar using this move. The operation is illustrated in Figure 34.

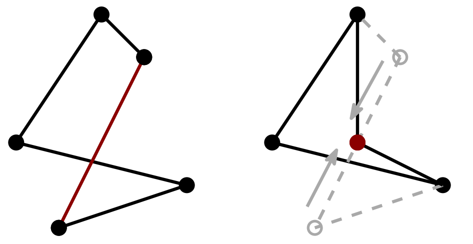


Figure 34: The Collapse move in a graph.

4.1.2 Slide

Assume edge e_1 with endpoints p_1 and p_2, and a vertex p_3 which is connected with p_2 via an edge e_2. Then, e_1 can "slide" along e_2 so that the new endpoints of e_1 become p_1 and p_3 if and only if there is not yet an edge with p_1 and p_3 as its endpoints. Afterwards there is no edge with endpoints p_1 and p_2. The only instances where no moves are possible is when the graph is either complete, or has no more than one edge. From this it follows that any non-planar embedding of a complete graph can never be made planar with this move. While not yet proven, the maximum possible moves of any instance with Slide seems to be $m * (m-1)$ (where m is the number of edges). An example of such an instance is when a vertex is connected to all other vertices, but those other vertices are not adjacent to each other. In other words, one vertex is degree $n-1$, and all others are degree 1. Note that, removing an edge e from a complete graph results in the number of possible moves increasing from 0 to $(n-2)*2$. This is because only the edges connected to what once was an endpoint of e can then slide. This "lack of an edge", or gap, can then be moved anywhere on the almost-complete graph in either one move (if the gap is next to an endpoint to where the gap should be moved next to), or two moves (any other situation). On the other hand, given any connected, non-complete graph it is possible to place any edge between any two vertices. However, it will take a minimum of $n-2$ steps to get an edge between the two degree-1 vertices of the original path. The Slide operation is shown in Figure 35.

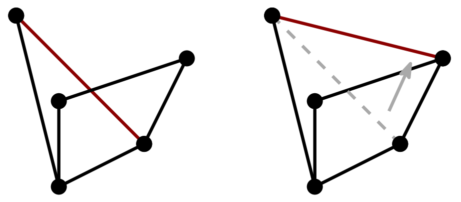


Figure 35: The Slide move in a graph. Note that the moved edge could not have slid along any other adjacent edge, since all locations it would have ended up on are already occupied by an edge.

4.1.3 Addition

Once again, assume an edge e with endpoints p_1 and p_2. After e is selected, a vertex p_3 can be placed anywhere on the plane. If p_3 is placed, e will be deleted and two new edges will be created: one with endpoints p_1 and p_3 and one with endpoints p_2 and p_3. In the game described later, the location of p_3 is restricted to a grid. Let us say there are g possible locations to place p_3. Then, for any graph with m edges and n vertices, the number of possible moves using one Addition is $m*g$. After addition, both m and n increase by 1, however the degree of any vertex will remain unchanged. A large number of Additions can be used to approximate any curve. Because of this it is logical that if a planar embedding of a graph is possible, then that graph can be made planar using only Addition. Addition is shown in Figure 36.

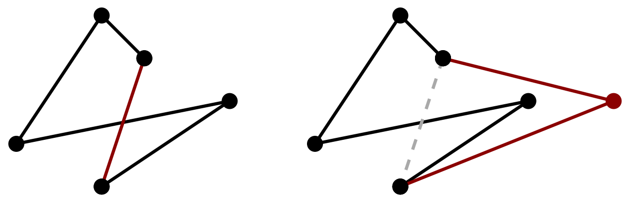


Figure 36: The Addition move in a graph.

4.1.4 Interaction between move types

In some cases order matters. For example, Addition removes an edge, so if the player wants to do something specific with the endpoints of that edge, it should be done before it is used for Addition. An example where order does not matter is when we want e_1 to slide along e_2, and also rotate e_2. Using these nuances with order creatively proved interesting when designing the puzzles for the game. Some operations can be expressed by others. A slide operation is the same as two rotation operations on the edge it would slide along, assuming no other edges are adjacent to the rotating edge.

4.2 The game: overview

Using the rotation operation along with the move types discussed in the previous section, a game was created, titled Graphs (working title). Graphs features two goal types: one with the familiar planarity as its goal and a new one, which we will refer to as the "matching" goal. As one might expect, in the planarity mode the goal remains altering the graph in such a manner that the end result is planar. The new game mode requires the player to manipulate the starting graph so that it matches (within a narrow margin) the outline of a shown goal state. A screenshot of an instance of the new mode is shown in Figure 37. Here the available moves, which are shown in the top right, are Rotate and Slide respectively. One solution would be to first rotate the upper edge counter-clockwise, and then slide the right edge along the rotated one. The other solution would be to first slide the right edge and then rotate the upper counter-clockwise.

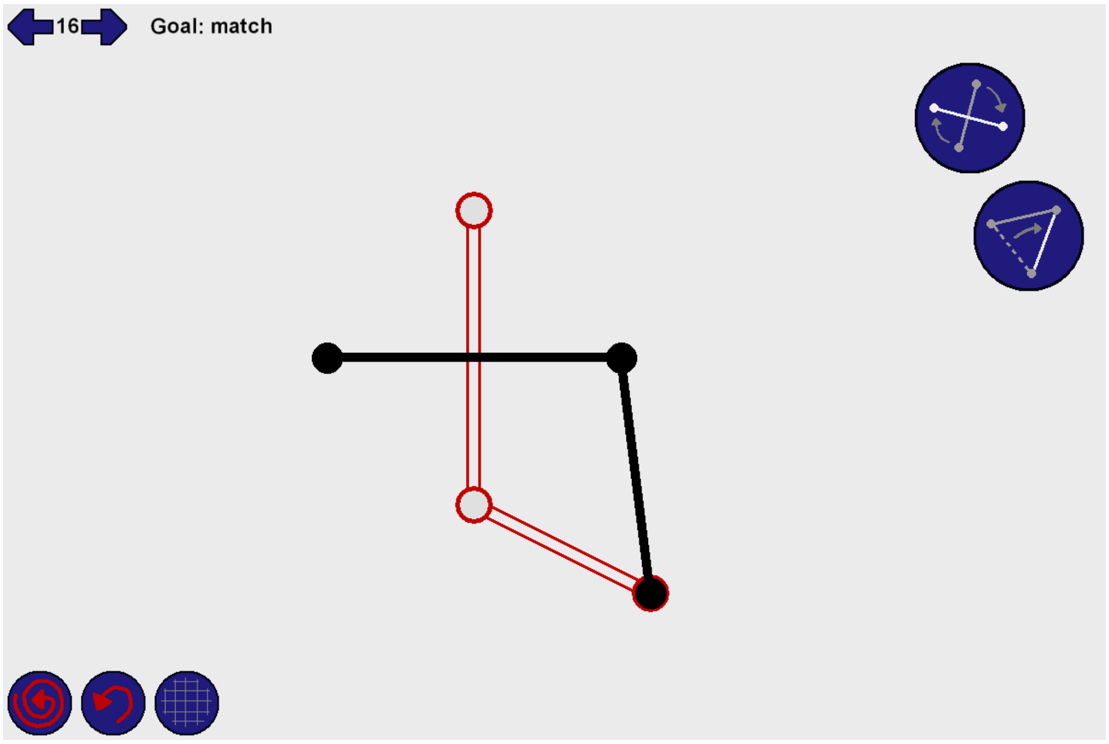


Figure 37: An example instance where the goal is to match the black graph with the red outline.

For both goal types the same rules apply regarding how moves are used. Each instance presents a number of available moves. Each of these can only be used once, although multiple of the same type can appear (e.g. an instance might require three rotations and a collapse). Not only the goal type needs to be fulfilled, it is also mandatory to have used *all* available moves. For example, an instance could start out planar, but since not all moves are used at that point, the puzzle is not yet solved.

Another mechanic is how the game handles overlapping elements. When two vertices are within very close range of each other, they are deemed "overlapping" and one is removed. Assume a situation where a vertex p_1 and p_2 are overlapping. The game arbitrarily chooses one vertex to be removed, say p_1. Next, all edges that once had p_1 as an endpoint are now assigned p_2 as endpoint instead of p_1. Due to this behaviour, it does not matter which of the overlapping vertices is removed. The same is done with edges that are at the same position, with the same rotation and length.

One final mechanic is that of "splitting" edges. What happens is the following: let e be any edge and q_1 and q_2 be the endpoints of e. Say a vertex p ends up coinciding with e after any operation (determined by taking the smallest distance between p and e and checking whether it is below a certain threshold), then the edge will split into two edges e_1 and e_2. e_1 will have q_1 and p as its endpoints, while e_2 will have q_2 and p as endpoints. Afterwards, e will be removed. Conveniently, for every operation there exist possible situations where splitting happens afterwards. This is because every operation has the potential to have at least one location where there is no vertex or edge before the move is done, but there is afterwards.

4.3 User study on control schemes

An interesting question for research is what interface and control scheme would be best for actually doing the four moves in the game. For Collapse it is rather straightforward, since we do not need to provide the system with more information than which edge should be reduced to one vertex in the middle. Therefore, simply selecting an edge is satisfactory. For Addition it is slightly more complicated, since not only do we need to select an edge, but also specify where the new vertex should be placed. This is currently handled by clicking an edge and holding the mouse button, then releasing the button where the new vertex should be placed. For Rotation and Slide two different control schemes were implemented that both had their own advantages and disadvantages. The control schemes of Rotation are as follows, keep in mind that our target platform is computers (more specifically: web-based):

- **Rotation A:** Click on an an edge with the left mouse button and hold. Moving the cursor reveals an arrow on the selected edge. The bottom of the arrow is located on the middle point of the half of the edge that the cursor was closest to when the edge was selected. Let p be the location of the cursor, and q be the location of the bottom of the arrow, then the length of the arrow is $max(dist(p, q), 80)$. The arrow needs to have a height of at least 32 for the rotation to happen (it shows it has reached this height by changing colour). The edge will rotate towards the direction of the arrow as is further illustrated in Figure 38.

- **Rotation B:** Either right-click an edge, or left-click an edge. Upon clicking, the chosen edge will either rotate clockwise when right-clicked, or counter-clockwise when left-clicked.

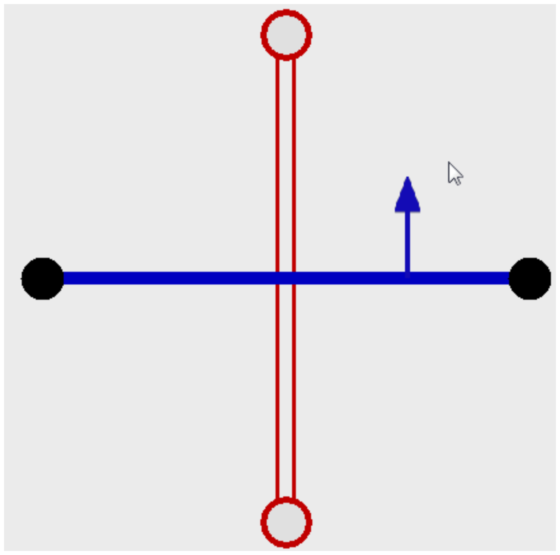


Figure 38: An example where a selected edge is about to be rotated in counter-clockwise direction using Rotation A.

The advantage of Rotation A is that, in theory, it should be more intuitive as it should be easier to visualise how the edge is going to be rotated. On the other hand, the advantage of Rotation B is that the input is quicker: Rotation A requires clicking and dragging, while Rotation B requires only a click.

The control schemes for Slide can be considered more similar to each other and are as follows:

- **Slide A:** First left-click on the edge that will be moved *and hold* the button. Hold the cursor on the edge to slide along and release.
- **Slide B:** First left-click on the edge that will be moved. Then left-click on the edge that the selected edge will slide along.

Both Slide variants require a comparable amount of work: Slide B needs two clicks, while Slide A is done by clicking and dragging. That said, Slide A requires slightly less effort than Slide B. Because both Slide options are so similar it would be interesting to see if any of the two has any strong preference over the other.

4.3.1 The experiment

The actual experiment consists of a questionnaire and a portion of the prototype of the game that needs to be played. During the questionnaire, the subject is twice asked to play a different section of the game. Because initial pilot testing showed that the splitting mechanic would be too difficult to grasp in the time span of the experiment, it was left out.

The questionnaire opens with basic instructions about the experiments, followed by four personal questions requiring the participant to fill in their age, highest obtained degree, how often they play games in general per week, and how often they play puzzle games. Afterwards they are instructed to play a part of the game, which is opened via a link in the questionnaire. This first part consists of 18 different screens:

- The first screen provides a short explanation of the goal of the game, as well as the mechanics to reach this goal. The first move, Addition, is also explained here. This screen introduces the first instance to be solved: it requires one simple Addition move.

- The second screen introduces Collapse. The puzzle is solved by simply performing Collapse on the one edge it contains.
- The third screen shows a graph with two edges and two Collapse moves available. The edges need to be collapsed in the right order, thus introducing the concept that order is important.
- The fourth screen combines a Collapse with an Addition, thus showcasing more implicit properties of these moves.
- On screens five and six, Rotation is introduced. The puzzle on screen five only requires one Rotation, while six needs two to be solved: one in either direction in order to test that the player understands that edges can be rotated both ways. Which control scheme is used here (Rotation A or Rotation B) is random, with a 50% chance for either option. The randomly chosen option is used for screens five to ten.
- Screens seven to ten contain puzzles using always at least one Rotation, often in combination with one or more Collapse or Addition moves.
- Screen eleven only contains text, and simply states that Rotation will now change control schemes.
- Screens twelve and thirteen have the exact same puzzles as screens five and six, however using the other control scheme.
- Screens fourteen up to seventeen display new puzzles, again using at least one Rotation and often combined with either Addition or Collapse.
- Screen eighteen only contains text, telling the player to continue the questionnaire.

The next question will ask which control scheme the subject preferred, either A or B. Below that is an optional question asking why they preferred their choice. The subject is then presented another link, directing them to the second part of the game. The second part is comparable to screens five to eighteen of part one, but with Collapse instead of Rotation and different puzzles. After finishing the second part, the participant returns to the questionnaire which will again ask which control scheme they preferred and why, although this time with respect to Collapse. Finally, an open question asks if there are any remarks one wishes to share. The actual questionnaire and levels as presented to the participants can be found in Appendix C.

4.3.2 Results

A total of 56 people participated with the user study. The link to the questionnaire was primarily shared with students, of which a large chunk consists of students of computer science. Taking this into consideration, as well as the age distribution shown in Figure 39 and the levels of highest attained education in Figure 40, it would be fair to say the group of participants mostly consists of students. The ages range from 18 to 65, where the mean is 24 (24.66) and the median is 23.

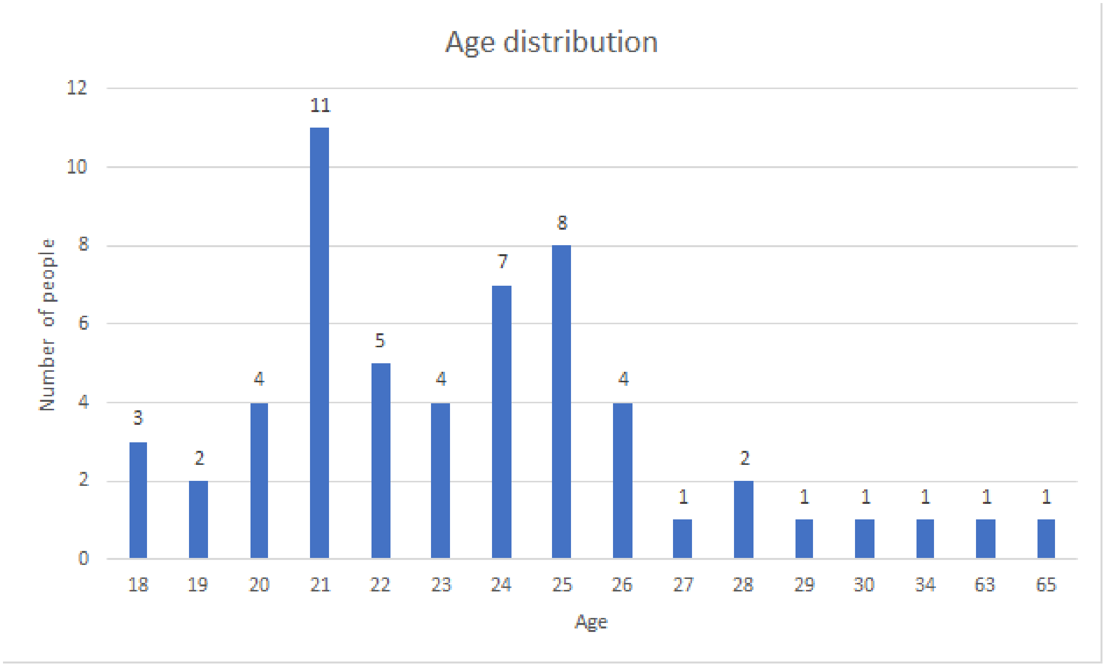


Figure 39: The ages of the participants.

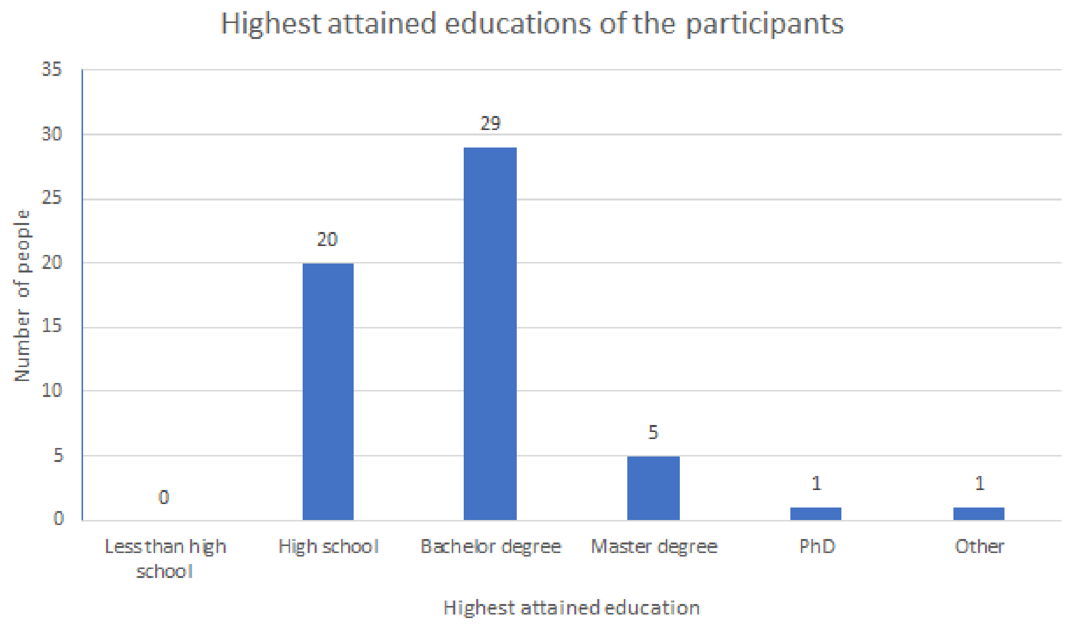


Figure 40: The highest attained levels of education of the participants.

Due to complications, we lost information of the first 21 participants regarding the order in which they got the control schemes. For this reason, we first show that we can make the assumption that this order does not matter using the remaining 35 participants, after which we can use the full group to draw conclusions on which control schemes were preferred.

While the experiment was set up with the intention that the order in which schemes A and B are presented have no influence on the results, and even though pilot tests did indicate that this assumption is indeed correct, a more academically justified indication is necessary. First of all, of these 35 people, 19 (54%) got Rotation A first, and 18 people (51%) got Slide A as the first one. This already shows that the randomisation can be assumed evenly distributed. 18 people (51%) chose the Rotation control scheme they got as first, and 16 (46%) picked the Slide control scheme they got as first. We first examine what the probability is that the distribution of choices is significantly uneven. We use Chi-squared with as null hypothesis that one control scheme that people get as first is chosen as preferred with a majority of 70% and that we should have a p-value lower than 0.05 (as is common). Our observed data are those described above. Our expected number of participants for Rotation is then 24.5 for A and 10.5 for B. For Slide our null hypothesis becomes 10.5 participants expected for A, and 24.5 for B. After chi-squared we are left with a p-value lower than 0.05 in both cases, meaning that we can comfortably reject both null hypotheses and assume the order does indeed not matter.

Now for the results from the entire group of 56 participants: 43 people (76.79%) preferred Rotation A, while 38 people (67.86%) chose Slide B as their preferred control scheme. Let the null hypothesis in both cases be an even split of 28 participants per scheme, then chi-squared with a degree of freedom of 1 yields a chi-squared of 16.07 for Rotation, and 7.14 for Slide. The p-value for the null hypothesis for Rotation is < 0.001, and the p-value for the null hypothesis for Slide is < 0.01. Since both p-values are smaller than 0.05 (in fact, considerably so), we can reject our null hypotheses, so it is safe to assume that Rotation A and Slide B both are the control schemes people prefer.

Next we examine if there is any correlation between the amount of time spent on games or how often the participants played puzzle games. The results from the questions that asked for these data are split into which option they chose. We start with the amount of time spent on games per week, the results are shown in Figure 41 and Figure 42.

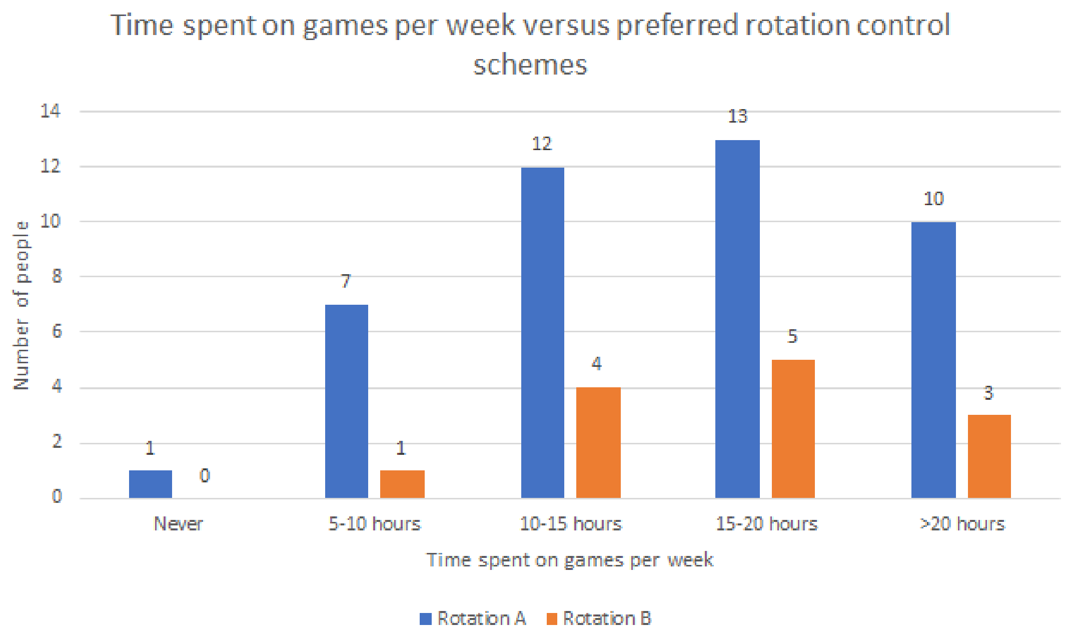


Figure 41: The participants and their choices for Rotation are grouped based on their time spent on games per week.

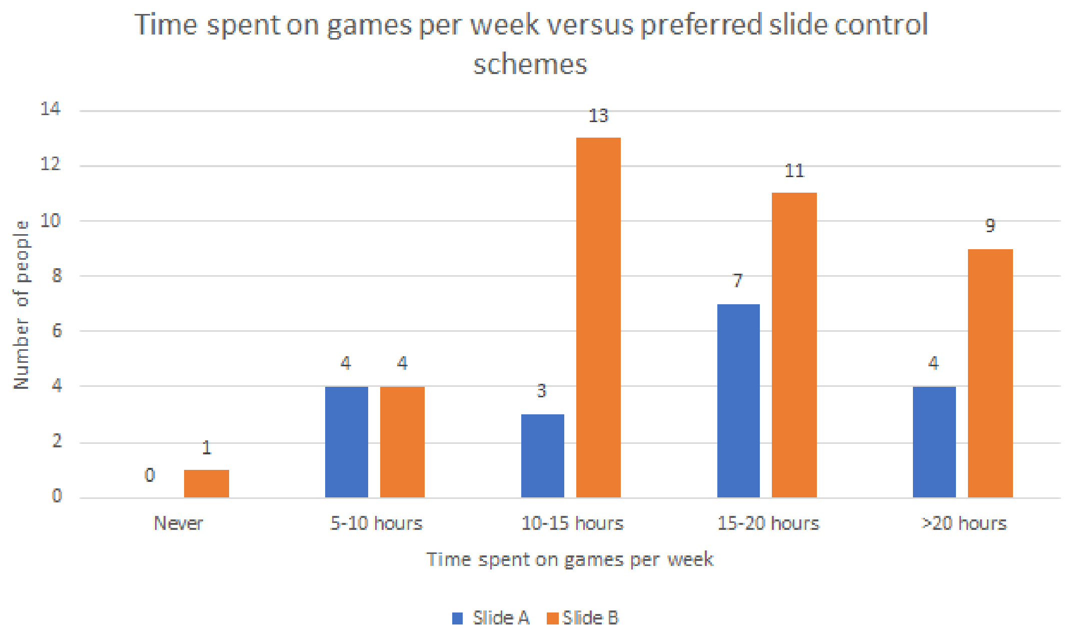


Figure 42: The participants and their choices for Slide are grouped based on their time spent on games per week.

We split the group in two, one half that spends less than 15 hours per week (group 1) and the other that spends more than 15 hours per week (group 2). The overall results yielded a 76.79% majority for Rotation A, so we use this distribution as our null hypothesis. The null hypothesis for Rotation becomes: 19 people pick Rotation A and 6 people pick Rotation B in group 1, and 24 for A and 7 for B in group 2. Compared to our observed results of 20 versus 5 in group 1, and 23 versus 8 in group 2, chi-squared yields a result of 0.40, which corresponds to a p value of 0.5. Since this is much larger than 0.05, we cannot reject the null hypothesis, meaning there is no correlation.

Doing the same for Slide, which has an expected percentage of 67.86% for option B, leaves us with a null hypothesis of 8 versus 17 for group 1, and 10 versus 21 for group 2. With the observed data being 7 versus 18 for group 1, and 11 versus 20 for group 2, we get a chi-squared value of 0.33, also resulting in a p-value greater than 0.05, greater than 0.5 even. There is no correlation between the choice for Slide and the amount of time spent on games either.

Next we examine if there is a correlation between the frequency of playing puzzle games and the preferred control schemes. Figure 43 and Figure 44 show the frequency each participant plays puzzle games, and their choices.

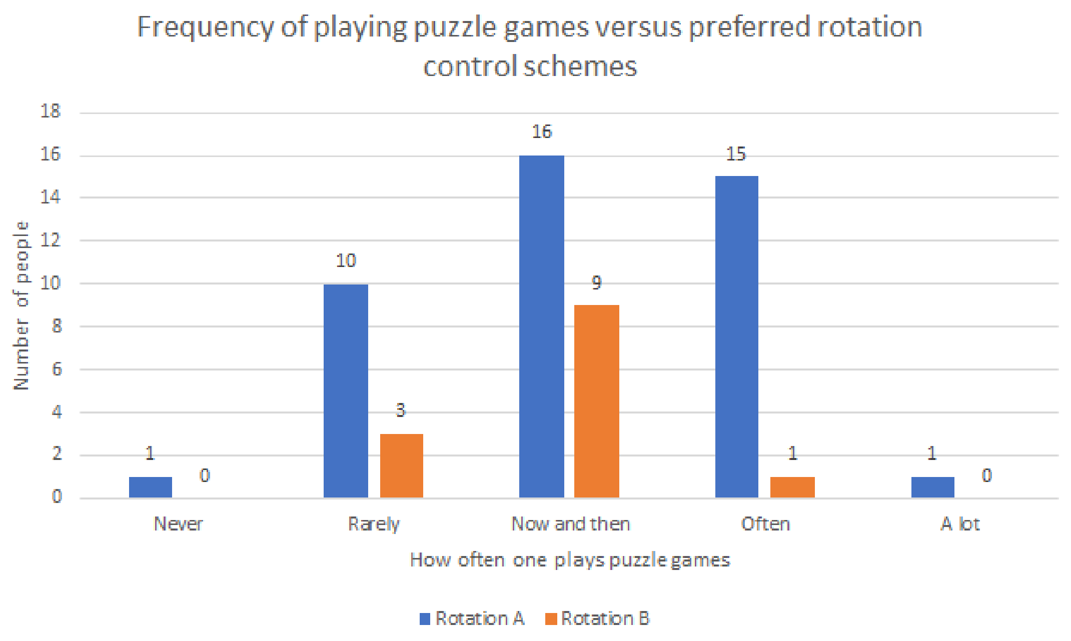


Figure 43: The participants and their choices for Rotation are grouped based on how often they play puzzle games

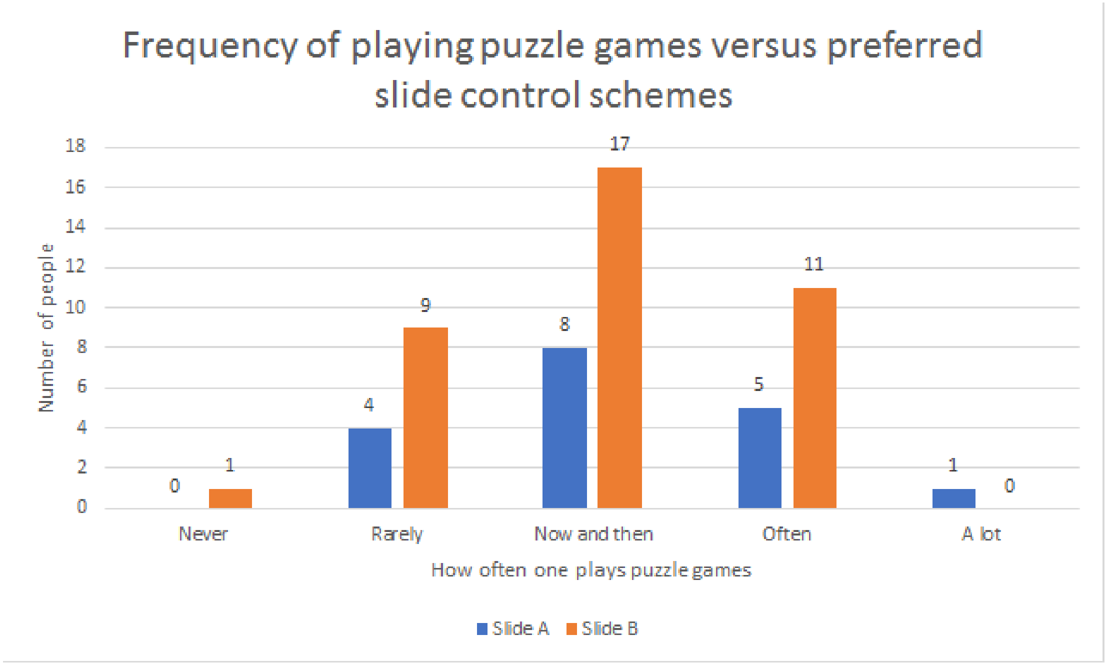


Figure 44: The participants and their choices for Slide are grouped based on how often they play puzzle games

The question on how often the participant plays puzzle game was less concrete than the question about the time spent on games, since the games question was measured in hours, while the puzzles question asks for a rough estimate, therefore implicitly asking how experienced each participant finds itself with puzzle games. It is a known phenomenon that, when presented with such choices, people tend to avoid the extreme options [6]. This is also shown in our results, as only one person chose the “Never” option, and only one person chose the “A lot” option. Because of this, we decided to group “Never” and “Rarely” with each other (group 1). We leave “Now and then” as its own group (group 2), and do the same thing we did with group 1 with “Often” and “A lot” (group 3).

Using the same methodology as above, the expected results for Rotation A are 11 for group 1, 19 for group 2, and 13 for group 3. The expected results for Rotation B are 3 for group 1, 6 for group 2, and 4 for group 3. This yields a chi-squared value of 4,92, corresponding to a p-value greater than 0.05, but smaller than 0.1. No correlation is therefore found.

The expected results for Slide A are 4 for group 1, 8 for group 2, and 5 for group 3. The expected results for Slide B are 10 for group 1, 17 for group 2, and 12 for group 3. Chi-squared yields a value of 0.28, corresponding to a p value of roughly 0.9, showing no correlation.

4.3.3 Discussion

Rotation A was the preferred control scheme to control the rotation operation. This implies that, despite Rotation B being the quicker option, people do prefer the more intuitive option. Intuitivity is also by the far most filled in reason by the participants who chose Rotation A. Within this context it is not hard to guess why; the rotating an edge can be difficult to visualise, especially since all adjacent edges move as well. This is reflected in the responses of the participants, where numerous people reported that they preferred Rotation A for this reason, and that the puzzles can be difficult to visualise. Any visual feedback on the result can therefore be assumed welcome.

For Slide, option B can be assumed the better option. Two points of discussion arose from the feedback: Slide can be hard to visualise (harder than the other variants, judging from the feedback), and the hitboxes for the edges were deemed quite small for the tasks at hand. In the case of it being hard to visualise one could argue that this might result in people needing more experience before they know what they are doing, implying that people might choose the option they get as second. We can reject this claim, as we have shown that the order in which the control schemes were presented did not alter the choices. As for the hitboxes, this is indeed a design point that should be taken into account when continuing development on the game. It is not especially important for this research as both Slide A and Slide B had the same size for the hitboxes, thus we assume this did not interfere with the results.

As for any correlations between experience with (puzzle) games and the preferred control schemes, none was found. While we cannot back this claim any more than with these results, we suspect that which control scheme anyone prefers is something innate to that person, and is unchanged by time spent with games.

In summary, we have shown which control schemes are preferred, but found no correlation between the participants' experience with games and particularly puzzle games.

5 Conclusions and future research

At the time of writing, this research can be considered quite new, so this thesis only scratched the surface of possible research and experiments that can be done on operations on edges in graph drawings, particularly 90 degree edge rotations. For this reason, and in an effort to give anyone who might be interested inspiration, this section aims to provide an extensive overview of all the possibilities found during the making of this particular research project.

5.1 Theoretical

Our research in Section 2 has shown that when only using the rotation operations, a graph will not "move away" as it is now proven that its centroid will remain constant. This is useful knowledge in the context of puzzle games, as we now know that it is sufficient to centre a camera at the centroid of a graph drawing. We have also shown that the sum of squared distances between all pairs of vertices remains constant, as well as the sum of squares of all distances between each vertex and the centroid. This in turn shows that, given a certain graph, this graph cannot scale indefinitely. In fact, the furthest distance any vertex could potentially reach from the centroid can be calculated. This is useful for cameras in puzzle games as well, as we now know how large our playing field might need to be. Finally, we defined a construction proving that for any bounded area A and any bounded area B that fully contains A, there exists a graph fully defined in A, that can grow to be partially out of B.

While some properties were proven, theoretical questions remain. One "big" question is: can every non-planar drawing of every planar graph be made planar using only 90-degree rotations? Since two rotations in the same direction is a swap, it has already been shown that this holds true for paths. As of now, it not yet clear how this question might be proved. We suspect that the fact that a rotation always yields new options for the vertex positions to change might be a first step towards a proof, and that the results from Section 3 might prove useful as an indication how one could proceed from there.

Section 4 briefly discusses three other operations. Future research could include identifying and proving similar properties for those. One example: it might be interesting how point set order types as introduced by Aichholzer et al. [1], might affect behaviour of any of these operations regarding planarity.

5.2 Experimentation

We have shown some behaviour when rotations are performed on random edges on a large scale. Here we looked at the average distance of either all the vertices and of the furthest vertex from the centroid. We also approximate how often a graph reaches a new furthest distance and how close it can get to the limit. Next, using a breadth-first solver along with a "random-first" solver, we showed the difference between thought-out decisions, and randomly doing actions. Finally, unexpected behaviour was documented regarding what happens when we rotate edges in a specific order. Apparently, through this process and when drawing all the vertices, it is possible to get figures that are visually interesting.

One fairly obvious improvement is that of our breadth-first algorithm. As of right now, it is exponential, though we do suspect there is much opportunity for optimisation. As discussed in Section 2, a rotation only affects adjacent edges along with the rotating edge, so we get many repeats. For instance, suppose graph G' that is attained by first rotating edge e_1 and then e_2 in clockwise direction in graph G, where both edges are not adjacent. We will also get to G' from G by first rotating e_2 and then e_1 in clockwise direction, although they are treated as different graphs as of now, both producing their own instances which in turn will also include a significant number of repeats. Repeats also occur when a clockwise and counter-clockwise rotation yield the same instance. This is possible when both endpoints of an edge are connected to the same set of vertices. Checking if this is the case decreases the number of repeats in some cases, although this can be considered a minor optimisation. Another simpler, but also minor optimisation, is dealing with the fact that first rotating an edge in a direction and then immediately rotating that same edge in the opposite direction will result in the same graph as before the two rotations, thus yielding a repeat. The algorithm could remember for each instance how it got to that instance, thus avoiding these repeats.

As for the findings in Section 3.4, it is clear that there is a certain behaviour waiting to be defined, although it is unclear how, and what that behaviour exactly entails. Documented in this subsection is some part of this behaviour that arises when rotating sequentially, where we looked at angles between edges, skipping instances to be drawn, repeating edges in the rotation order and changing the rotation angle. We have not looked into what would happen if we alternate between clockwise and counter-clockwise rotations. Another alteration that might be interesting is to make changing the rotation angle a part of the sequence. For example, make all even rotations 90 degree rotations, and all odd rotations 45 degrees. What would be interesting is to see if it is possible from just one instance and its order to predict what the resulting shapes are going to be through some function. If that is possible, then the process can be dramatically optimised, since then there would be no more need to do all the individual rotations. As for a practical use, the resulting images are thus far considered visually pleasing to the people we have showed it to, so entertainment seems like the obvious purpose. Some tool for educational purposes seems plausible as well.

5.3 Design

Design-wise, many more interesting research questions remain untouched. While we compared two control schemes for two of the operations, the results are not exhaustive. The results do give an indication on what kind of interface is preferred, but improvements, as discussed in Section 4.3, are certainly possible. Furthermore, the other operations thus far remain untouched with regard to how to control them. That said, the remaining operations are arguably simpler to execute and thus provide less options for an ideal control scheme.

As of yet nothing has been said about the difficulty of the available moves in the context of the developed game. It might be interesting to compare and analyse which moves, or combination thereof, is perceived as more difficult than others. The same can be said about fun. Difficulty and fun of the defined goal states (planarity and matching) have also not been analysed as of yet. Going even further, the splitting mechanic was left out of the user study, as pilot tests showed it would be too difficult to learn within the time span of the experiment. Interactions between the introduction of splitting and certain move types with regards to fun and difficulty should prove interesting as well. It is not difficult to see how the splitting mechanic in general could also provide theoretical research questions.

One idea that was formed during this research, but we did not have time to actually execute, was a study on procedural generation of levels of this game. Planarity levels are more straightforward to generate, as the goal is more open ended. The matching goal type however, is more complex as the graph needs to exactly match some goal state. For generation, there are at least two possible approaches:

- Start by randomly generating the changeable black graph, so a beginning state, and perform a number of moves on it to get the goal state.
- Start by randomly generating the goal state, and perform a number of *inverse* moves on it to get the black start graph.

A criterion for both the generated graphs and all instances the player gets when doing the correct moves towards the winning goal state would be that they all have to be good, using criteria from Section 3.3. A drawback of this is it then might be easier for experienced players to find the right solution, as instances that are "ugly" can not be part of the right solution. However, as we have shown, requiring all possible instances resulting from the available moves to be good is not reasonable. An acceptable middle ground option might be to first generate a number of different graph drawings from the same graph with some combination of moves and choose the one that has the most good instances as results of those moves. It would be interesting to see if there is any reason to choose one generation approach over the other. This can be determined through user studies. Our full initial plan for the experiment can be found in Appendix D.

References

[1] Aichholzer, O., Aurenhammer, F., & Krasser, H. (2002). Enumerating order types for small point sets with applications. *Order, 19*(3), 265-281.

[2] Altshiller-Court, N. (2007). Properties of the Triangle. In *College Geometry: An Introduction to the Modern Geometry of the Triangle and the Circle* (pp. 70-71). Mineola, NY: Dover Publications, Inc.

[3] Bandyapadhyay, S., Banik, A., Das, S., & Sarkar, H. (2015). Voronoi game on graphs. *Theoretical Computer Science, 562*, 270-282.

[4] Goaoc, X., Kratochvíl, J., Okamoto, Y., Shin, C. S., & Wolff, A. (2007, September). Moving vertices to make drawings plane. In *International Symposium on Graph Drawing* (pp. 101-112). Springer, Berlin, Heidelberg.

[5] Hearn, R. A., & Demaine, E. D. (2009). The Constraint-Logic Formalism. In *Games, puzzles, and computation* (pp. 15-24). AK Peters/CRC Press.

[6] Kalton, G., Roberts, J., & Holt, D. (1980). The effects of offering a middle response option with opinion questions. *The Statistician*, 65-78.

[7] Kraaijer, R., van Kreveld, M., Meulemans, W., & van Renssen, A. (2018). Geometry and generation of a new graph planarity game. In *2018 IEEE Conference on Computational Intelligence and Games (CIG)* (pp. 1-8). IEEE.

[8] Van Kreveld, M. (2011). Bold graph drawings. *Computational Geometry, 44* (9), 499-506.

[9] Lam, T. K. (1997). Connected sprouts. *The American mathematical monthly, 104*(2), 116-119.

[10] Purchase, H. C. (2002). Metrics for graph drawing aesthetics. *Journal of Visual Languages & Computing, 13*(5), 501-516.

[11] Shoham, Y. (2008). Computer science and game theory. *Communications of the ACM, 51*(8), 74-79.

[12] Spillner, A., & Wolff, A. (2008, January). Untangling a planar graph. In *International Conference on Current Trends in Theory and Practice of Computer Science* (pp. 473-484). Springer, Berlin, Heidelberg.

[13] Tamassia, R., Di Battista, G., & Batini, C. (1988). Automatic graph drawing and readability of diagrams. *IEEE Transactions on Systems, Man, and Cybernetics, 18*(1), 61-79.

[14] Tantalo, J. Planarity. Retrieved January 13, 2019, from http://planarity.net/game.html

[15] Teramoto, S., Demaine, E. D., & Uehara, R. (2006, May). Voronoi game on graphs and its complexity. In *Computational Intelligence and Games, 2006 IEEE Symposium on* (pp. 265-271). IEEE.

[16] Thomassen, C. (1980). Planarity and duality of finite and infinite graphs. *Journal of Combinatorial Theory, Series B, 29*(2), 244-271.

[17] Verbitsky, O. (2008). On the obfuscation complexity of planar graphs. *Theoretical Computer Science, 396*(1-3), 294-300.

A Results of exploratory experiments

This appendix is supplementary to Section 3.4, and provides more details on previous figures (Section A.1) as well as more visuals of results from exploratory analysis (Section A.2). For all figures, the colours of the vertices are as follows:

- Vertex 1: $rgb(0, 200, 0)$, "green".
- Vertex 2: $rgb(230, 160, 0)$, "orange".
- Vertex 3: $rgb(0, 220, 220)$, "light blue").
- Vertex 4: $rgb(250, 0, 0)$, "red".
- Vertex 5: $rgb(0, 0, 205)$, "dark blue".
- Vertex 6: $rgb(204, 204, 0)$, "yellow".
- Vertex 7: $rgb(150, 0, 150)$, "purple".
- Vertex 8: $rgb(102, 52, 0)$, "brown".
- Vertex 9: $rgb(255, 100, 100)$, "pink".
- Vertex 10: $rgb(64, 64, 64)$, "grey".
- All other vertices after vertex 10: $RGB(0, 0, 0)$, "black".

The edges are defined by endpoints, which are refereed to by numbers. These are the same numbers as in the list above. So for example, "1" will always refer to the green vertex.

A.1 Vertex coordinates of previous graphs

Below are the raw coordinates of the vertices used in most figures in Section 3.4. They are in the same order as the previously mentioned list that detailed their colours. The x and y coordinates are separated by one white space.

Vertex coordinates of Figures 20, 21, 22, 24, 25, 30, and 32:

1. 595 468
2. 824 386
3. 1162 544

Vertex coordinates of Figures 26, 27, 28, 31, and 33:

1. 857 625
2. 725 409
3. 1050 432
4. 508 324

A.2 Many more results

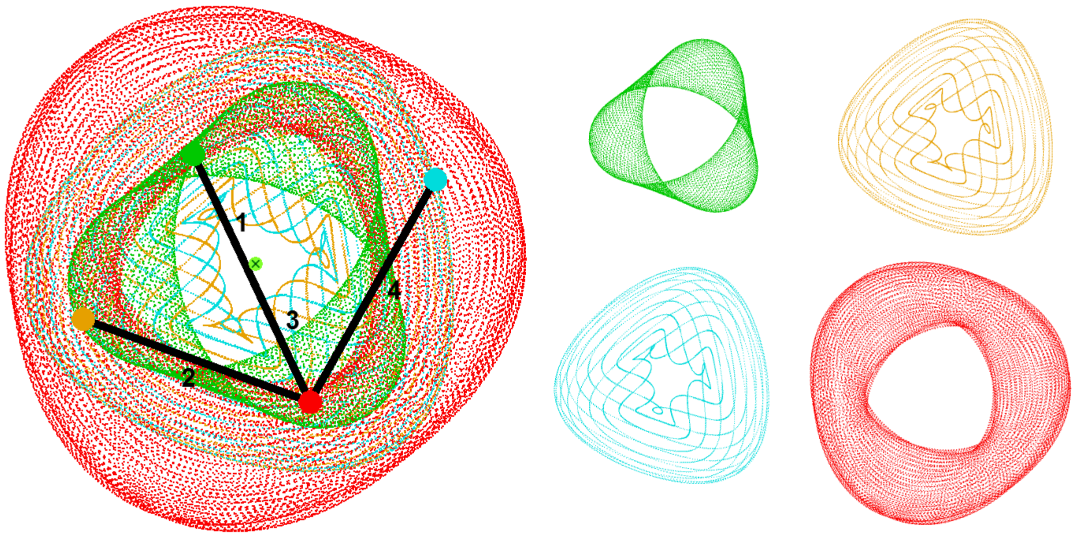


Figure 45: $R = 30,006$

Vertex coordinates for Figure 45:

1. 670 331
2. 530 540
3. 977 363
4. 818 645

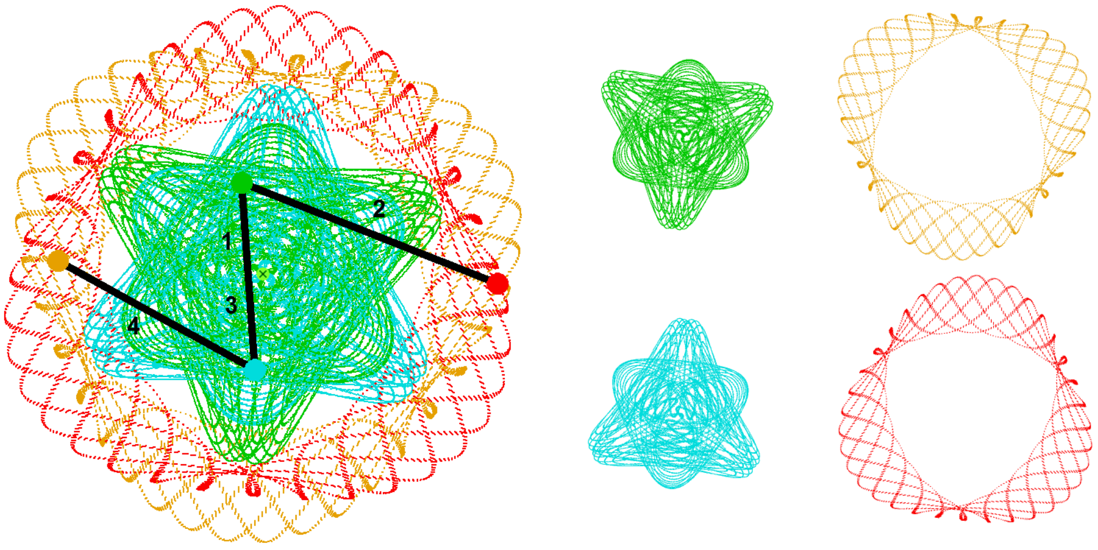


Figure 46: These results were used for the image on the front of this thesis. $R = 50,022$

Vertex coordinates for Figure 46:

1. 765 338
2. 515 444
3. 783 592
4. 1113 475

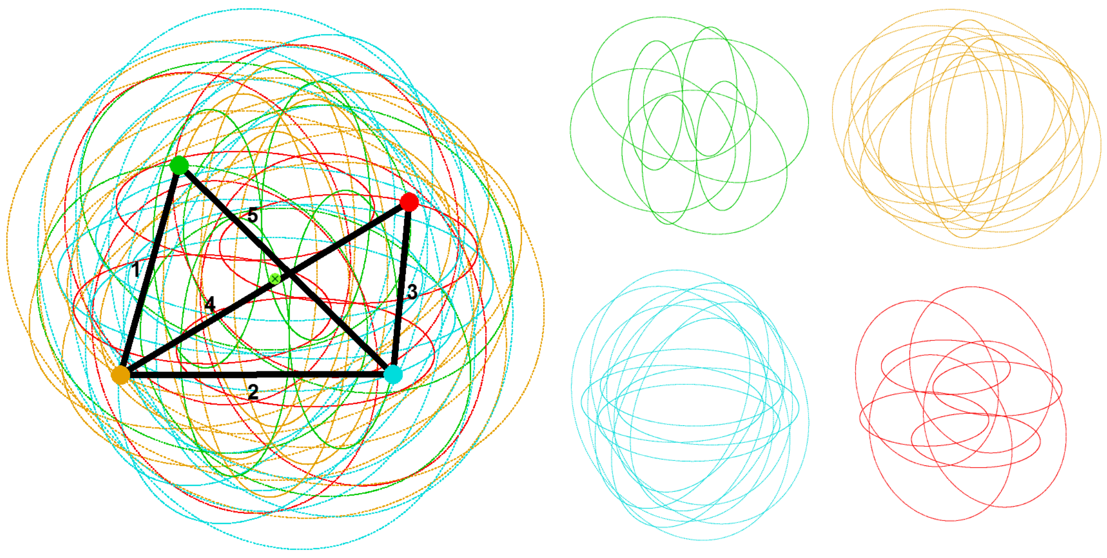


Figure 47: $R = 50,025$

Vertex coordinates for Figure 47:

1. 630 297
2. 539 622
3. 962 621
4. 987 354

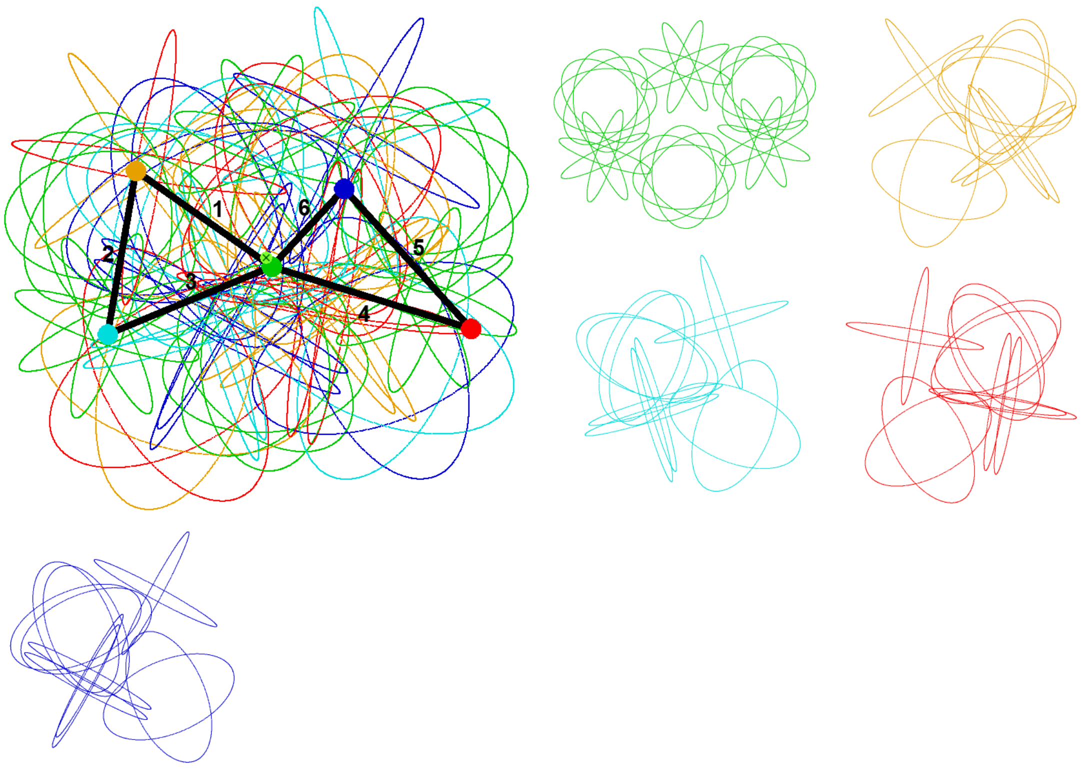


Figure 48: $R = 25,051$

Vertex coordinates for Figure 48:

1. 722 461
2. 525 322
3. 484 559
4. 1011 551
5. 827 348

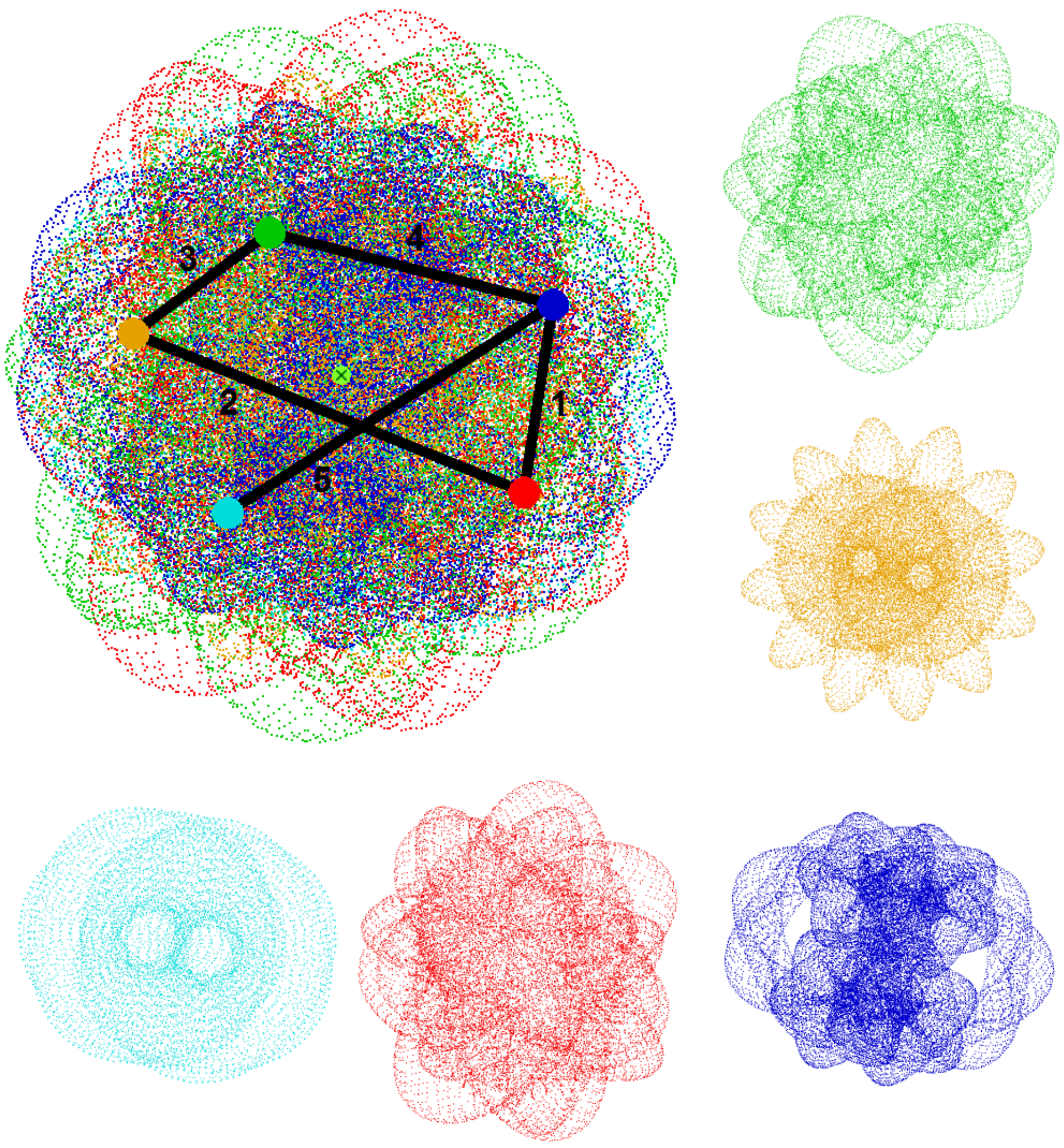


Figure 49: $R = 50,058$

Vertex coordinates for Figure 49:

1. 676 349
2. 549 443
3. 637 610
4. 913 591
5. 940 416

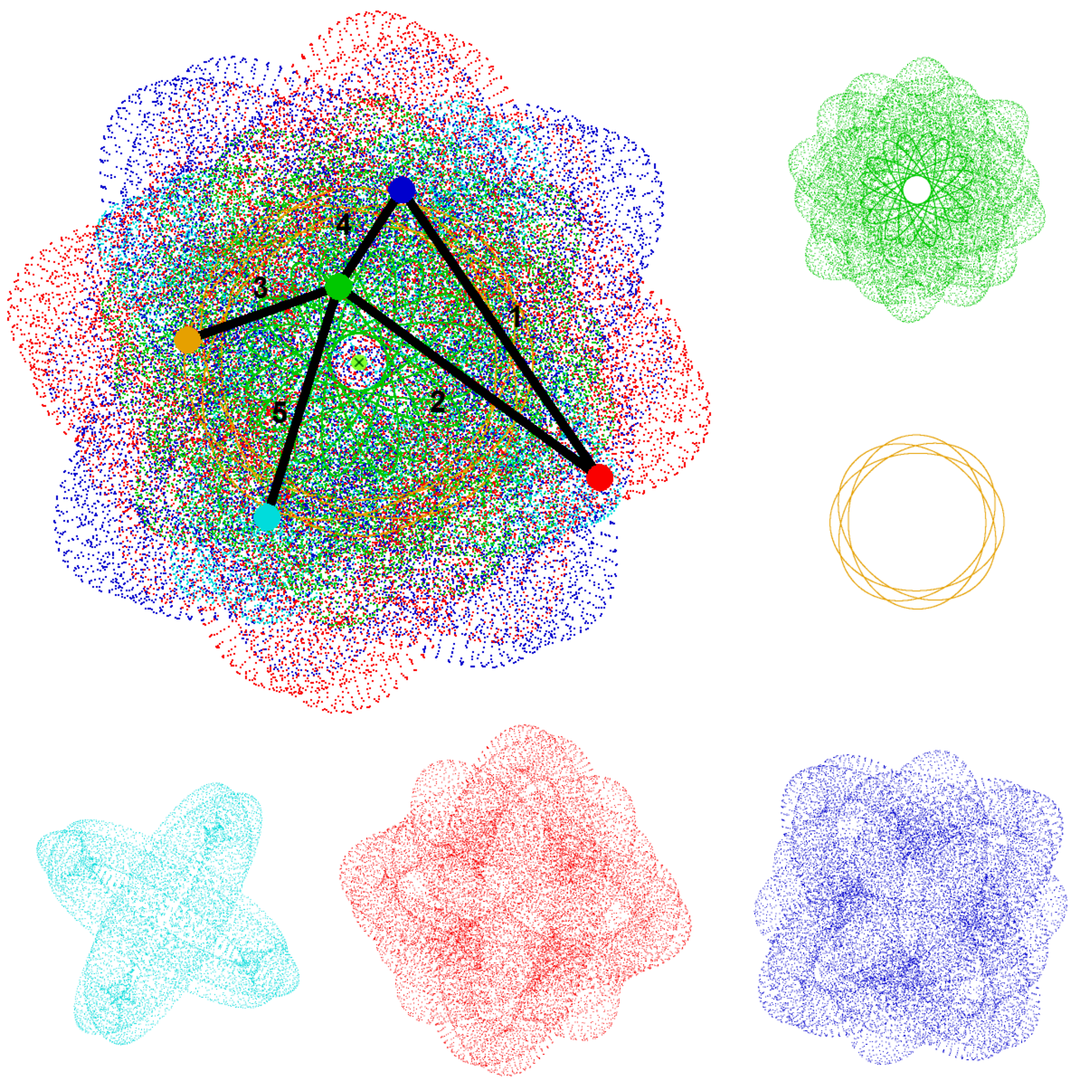


Figure 50: $R = 50,039$

Vertex coordinates for Figure 50:

1. 759 367
2. 592 426
3. 680 622
4. 1048 578
5. 829 260

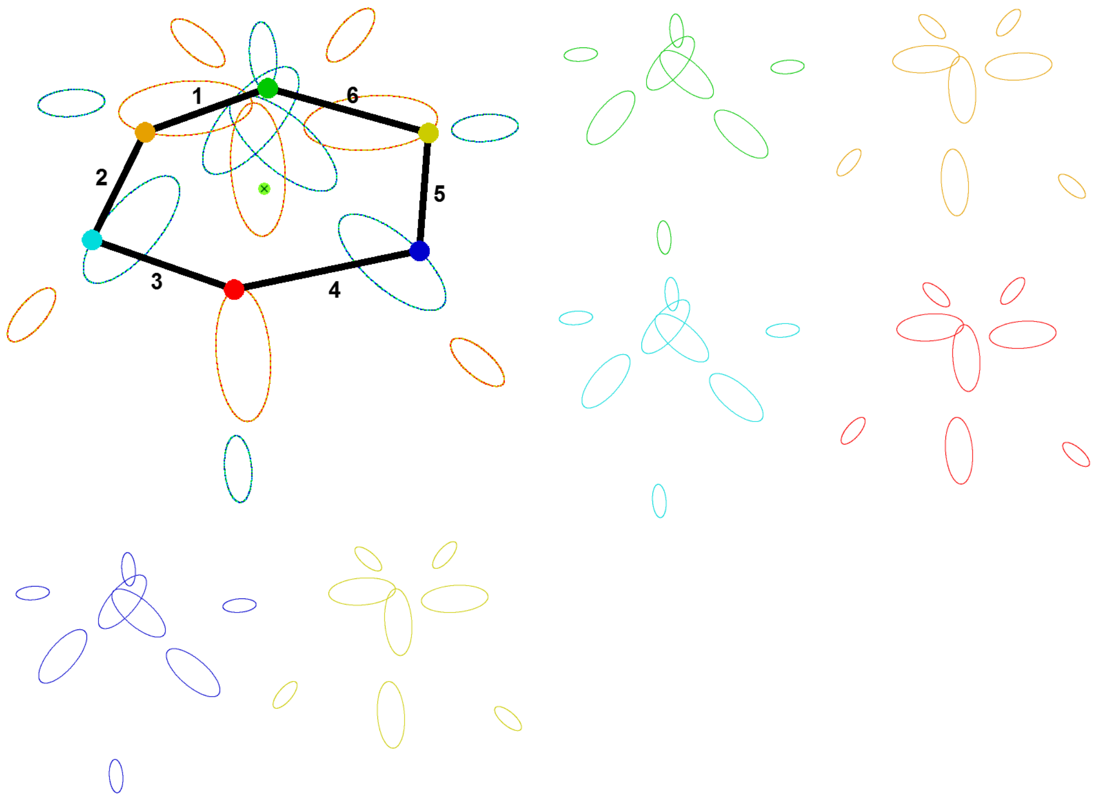


Figure 51: $R = 10,031$

Vertex coordinates for Figure 51:

1. 703 269
2. 525 333
3. 448 489
4. 654 561
5. 923 505
6. 936 334

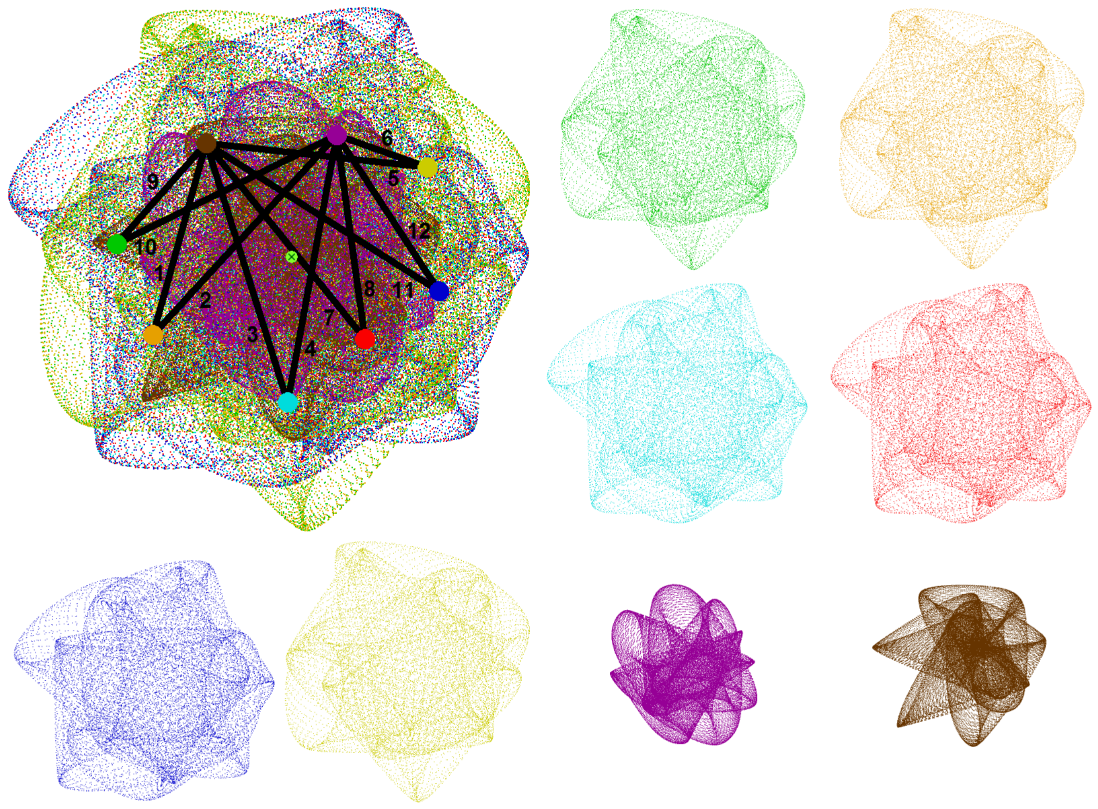


Figure 52: $R = 100,030$

Vertex coordinates for Figure 52:

1. 630 412
2. 684 547
3. 884 647
4. 999 553
5. 1109 482
6. 1092 298
7. 957 250
8. 763 262

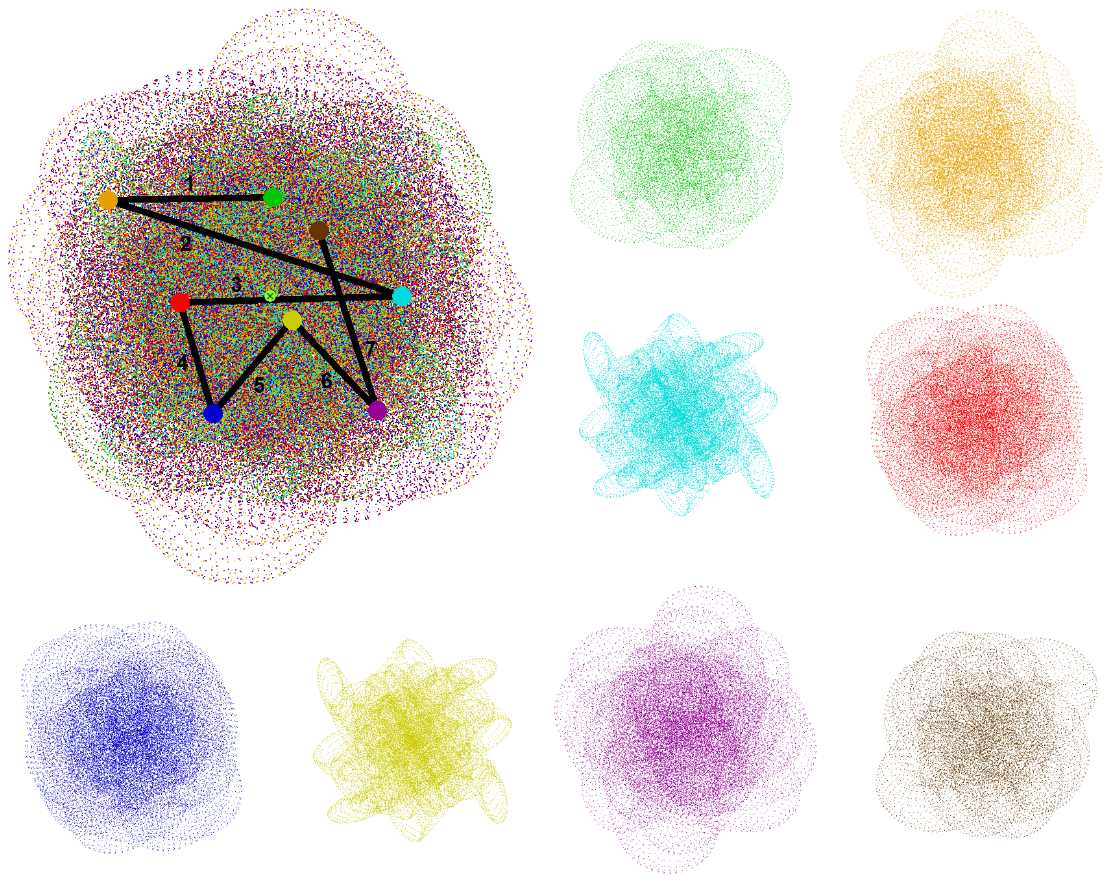


Figure 53: $R = 100,021$

Vertex coordinates for Figure 53:

1. 788 328
2. 534 332
3. 987 480
4. 645 490
5. 696 661
6. 818 517
7. 949 656
8. 859 379

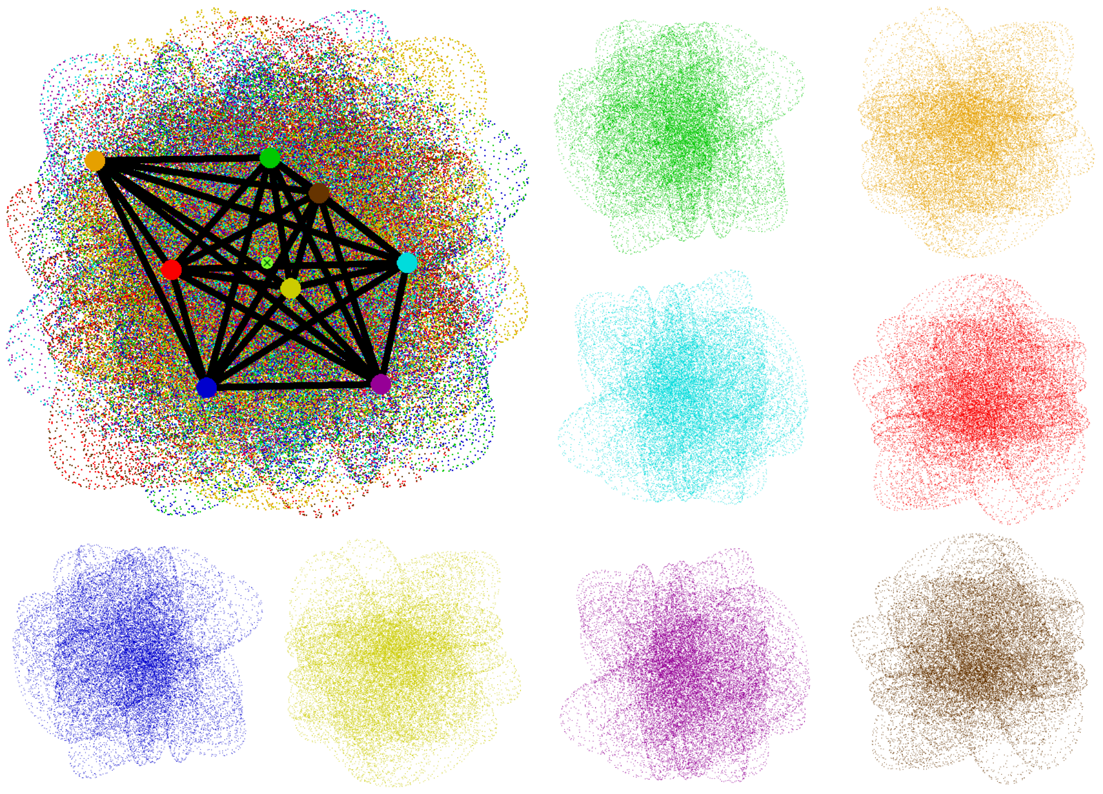


Figure 54: $R = 100,028$

Vertex coordinates for Figure 54:

1. 788 328
2. 534 332
3. 987 480
4. 645 490
5. 696 661
6. 818 517
7. 949 656
8. 859 379

Edge order for Figure 54 (edges defined by endpoints):

1. 1 2
2. 1 3
3. 1 4
4. 1 5
5. 1 6
6. 1 7
7. 1 8
8. 2 3
9. 2 4
10. 2 5
11. 2 6
12. 2 7

13. 2 8
14. 3 4
15. 3 5
16. 3 6
17. 3 7
18. 3 8
19. 4 5
20. 4 6
21. 4 7
22. 4 8
23. 5 6
24. 5 7
25. 5 8
26. 6 7
27. 6 8
28. 7 8

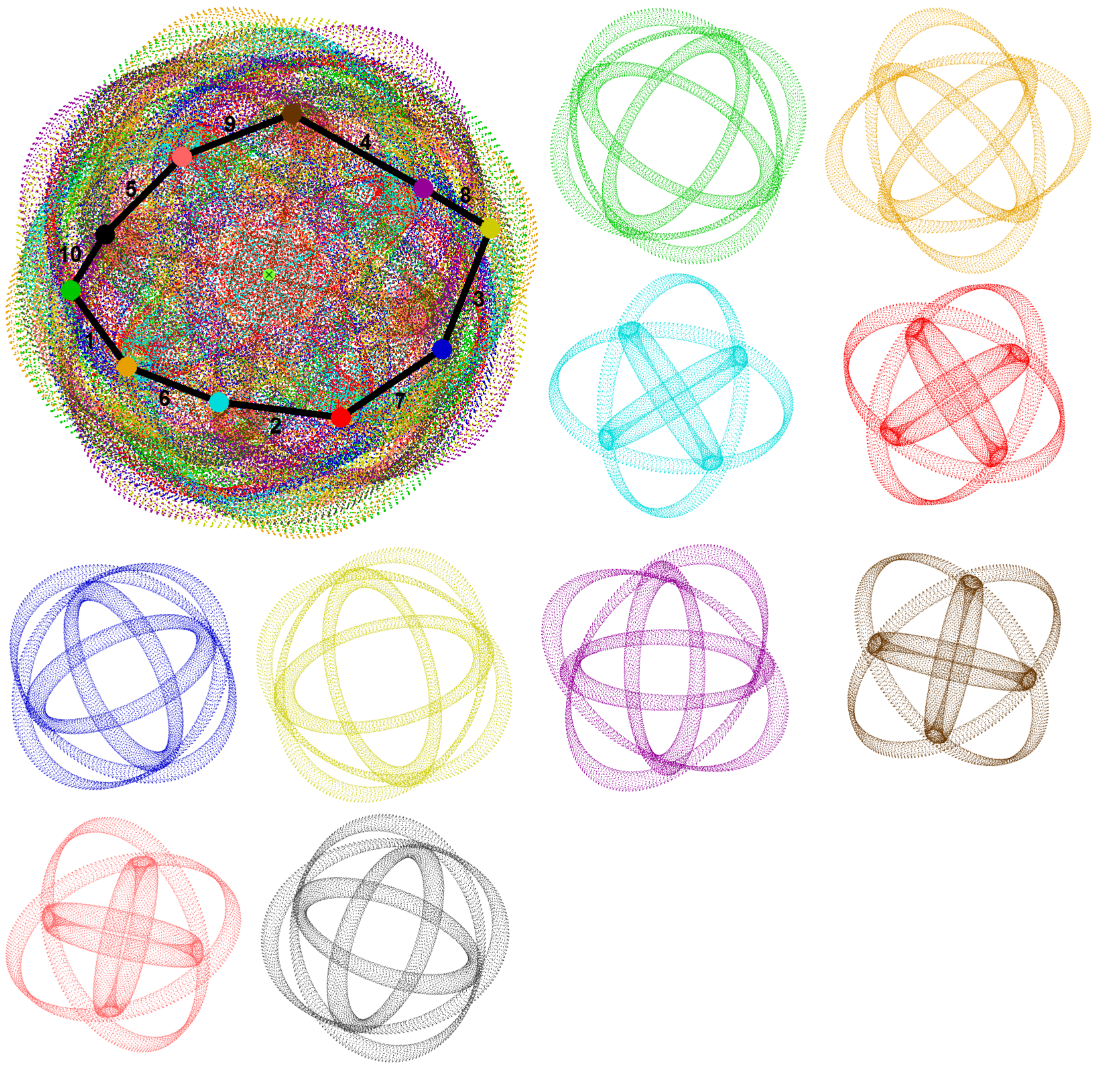


Figure 55: $R = 100,023$

Vertex coordinates for Figure 55:

1. 452 458
2. 536 572
3. 673 625
4. 855 648
5. 1006 546
6. 1078 366
7. 978 306
8. 782 195
9. 618 259
10. 503 375

B Manual for the program used for Section 3

In this section of the appendix you will find a slightly informal, but hopefully comprehensive manual of the program used to do the experiments in Section 3. In particular, this program in its current state can be used to make figures such as those shown in Section 3.4 and in Appendix A.

Welcome to the controls of the supplementary experimental program of my thesis on, among other things, edge rotations in graphs. All functions are mapped to some button. Some functions are arbitrary, many choices for which button are used are even more arbitrary.
Link to the online version (*significantly* slower than the local version):
`https://ferociter.itch.io/graphs-experimental?secret=g3f0u6Sv2pTqFgoy83DIJPqFoo`

CONTROLS
R: reset room (very important)
D: while hovering over an edge, rotates the edge clockwise
A: while hovering over an edge, rotates the edge counter-clockwise
Left Mouse Button: create vertex
Right Mouse Button (on a vertex): (de)select vertex, select 2 vertices to create an edge
F: make the graph complete (NOTE: as of now only works if there are no edges yet!)
P: draw a path between all vertices (NOTE: as of now only works if there are no edges yet!)
Shift (hold): when holding shift the next drawn vertex will snap to the nearest grid crossing
Ctrl: create an edge between the last 2 drawn vertices
G: toggle grid
I: toggle HUD
Alt: update info on screen (mostly unnecessary, but does show you an updated centroid anytime)
Backspace: remove latest vertex (WARNING as of now breaks the program if that vertex is connected)
Del: delete all vertices and edges (have not tested this enough)
F1: please no touch
F7: make screenshot and txt file with data, does not work in HTML5 version
F8: make instance "screenshot-ready"
H: hide graphs and centroid
L: show the surface of only one vertex
K: switch between surfaces
Right arrow key: do next single rotation: next in sequence when not in non-random mode.
Left arrow key: reverse previous rotation in non-random mode
Enter: do random-first search on current graph drawing
End: do breadth-first search on current graph drawing (not recommended)

Most of the buttons below toggle values shown on the right of the screen
Spacebar: rotation mode (rotates edges, behaviour depends on modes below). Rotations are always clockwise
M: toggle draw mode of the edges. If off, vertices get drawn
N: toggle random mode (in rotation mode the next edge to rotate is chosen randomly when random mode is on, they are rotated sequentially when it is off, in the order they were created)
B: circles: show circles around the original graph (mostly pointless)
S: stop mode (stops rotation mode whenever we find a graph with a new furthest distance)
V: fast mode (does 1000 rotations before the next is drawn, really fast! This is on per default so if you want to draw everything, turn this off)
C: if on, all drawn edges get cool colours

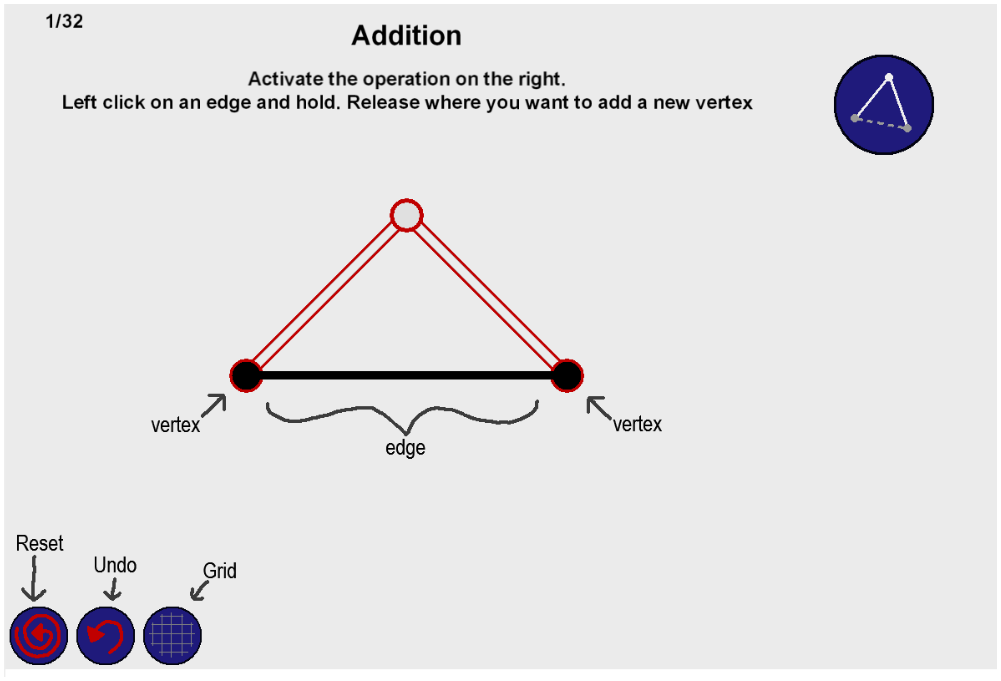
1: quickest growth pre-set
2: 3 vertices on 1 spot, 1 on another
3: 3 equally distributed points on a straight line pre-set
4: right angle pre-set
5: one of the graphs used in Section 3.4
6: one of the graphs used in Section 3.4
0: random points in 200 r circle (not recommended, has weird bugs when done this way)

INFORMATION ON SCREEN
The top right represents all modi, 1 means it's on, 0 means off. Pay attention to them.
Information on the bottom:
R: number of rotations
V: number of vertices
E: number of edges
TD: total distance between all vertices
FD: furthest distance measured
P: prediction (i.e. the furthest the graph could potentially stretch from the centroid)
RFD: the rotation number in which we found the FD

WORKFLOW
Example 1 *Make a cycle of 6 and rotate it randomly as fast as possible, while drawing the edge without any fancy colours:*
Press C to turn off gradient, M to draw edges, V to turn on fast mode, N to turn on random mode. Left click anywhere on the room to place a vertex, do this 6 times. Press P to create a path. Then, right-click on your last vertex, and right-click on your first vertex (in any order) to complete the cycle. Press Space to begin rotation.
Example 2 *Make a complete graph of 3 vertices, with repetitions, and rotate in a certain order while drawing all vertices of all instances:*
Let us assume you just did example 1 and the program is still running with the settings of that example. Press R to reset the room; everything is reset, except for the settings of the modi (shown top right). Press M to stop drawing edges, V to draw all instances, N to turn off randomness. Place three vertices by left-clicking anywhere at three locations. Press F to make the graph complete. For repetitions, simply make more than one edge between two vertices. Press Space to begin rotation. While rotating, you can press H to see the emerging figures better and press L to only view one vertex-layer. Then press K to switch between layers.

C User study levels and Questionnaire

Below are screenshots of all the "levels" (some only contain instructions) used for the user study in Section 4.3. The final image is a compilation of all the questions in the questionnaire.


1/32

Addition

Activate the operation on the right.
Left click on an edge and hold. Release where you want to add a new vertex

Figure 56: Screen 1

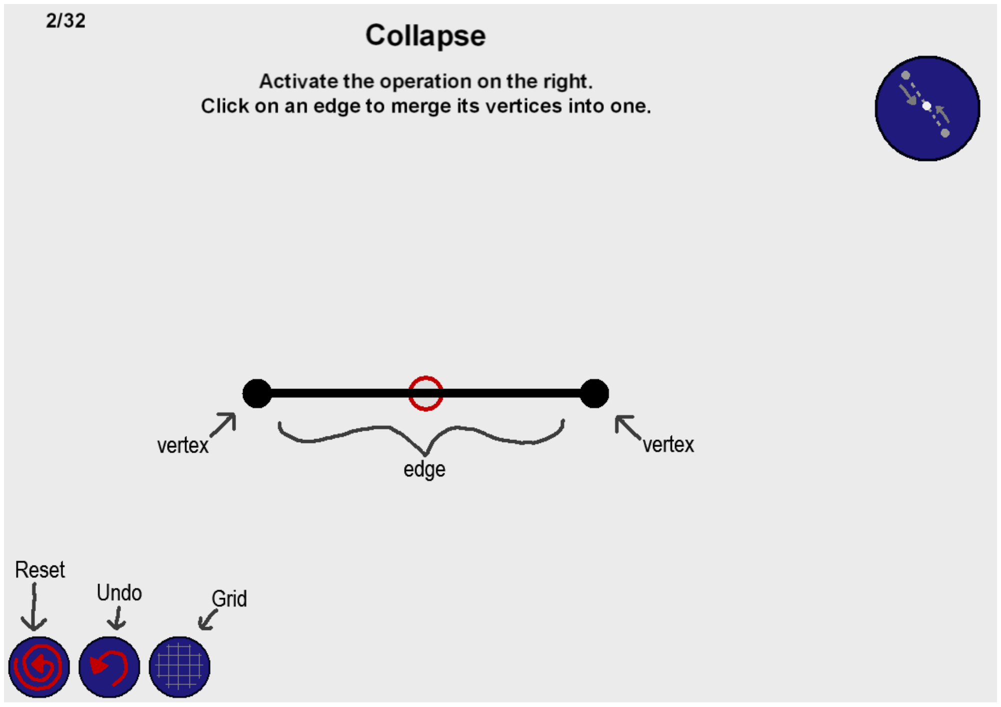

2/32
Collapse
Activate the operation on the right.
Click on an edge to merge its vertices into one.
vertex
edge
vertex
Reset
Undo
Grid

Figure 57: Screen 2

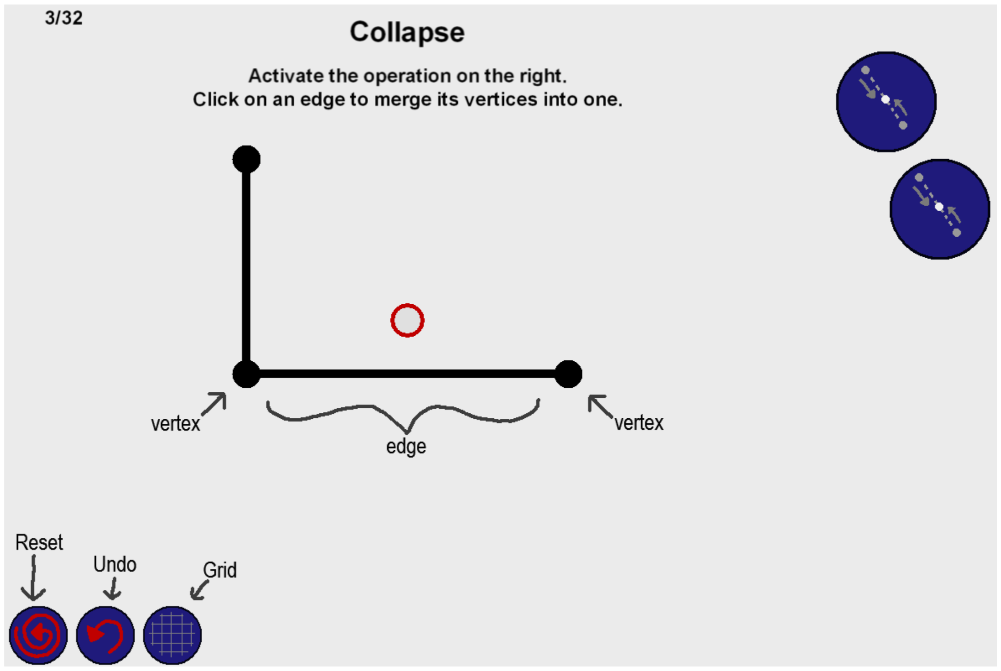

3/32
Collapse
Activate the operation on the right.
Click on an edge to merge its vertices into one.
vertex
edge
vertex
Reset
Undo
Grid

Figure 58: Screen 3

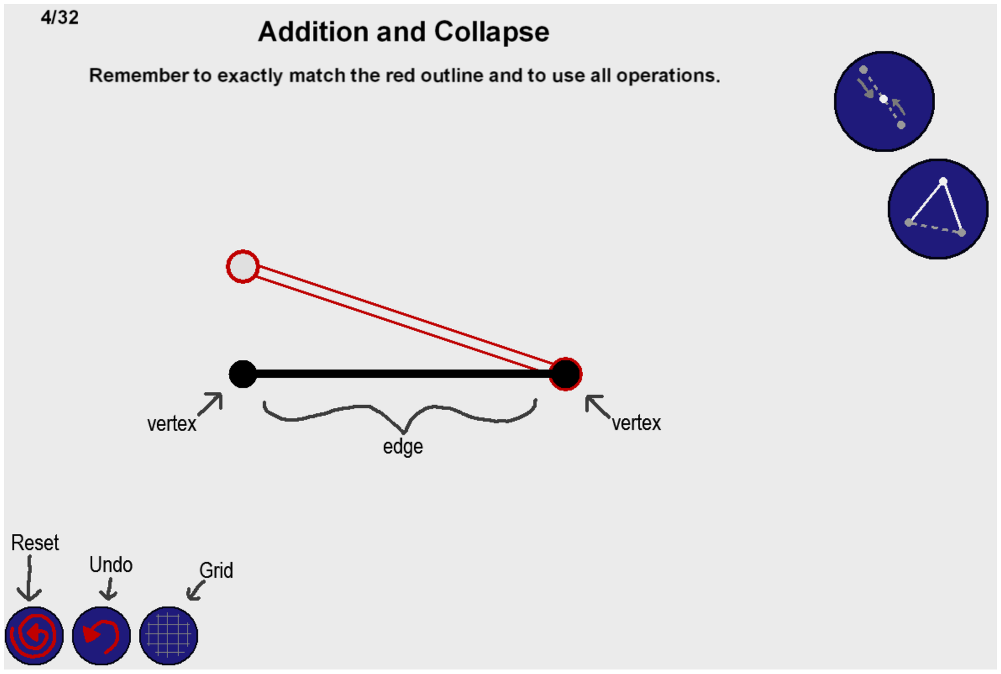

4/32
Addition and Collapse
Remember to exactly match the red outline and to use all operations.
vertex
edge
vertex
Reset
Undo
Grid

Figure 59: Screen 4

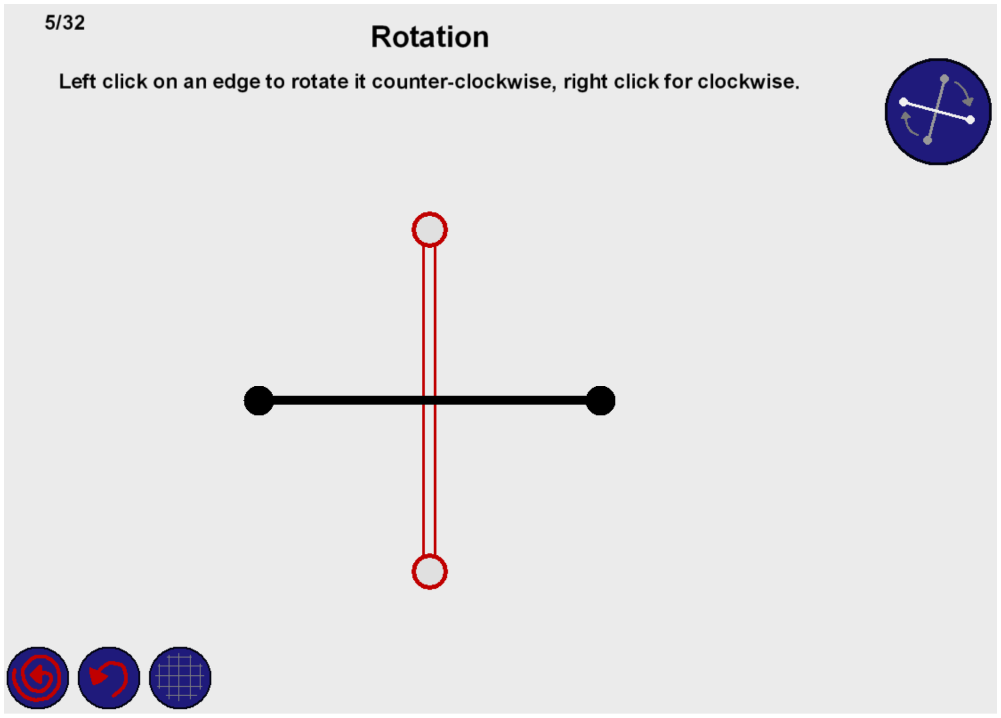
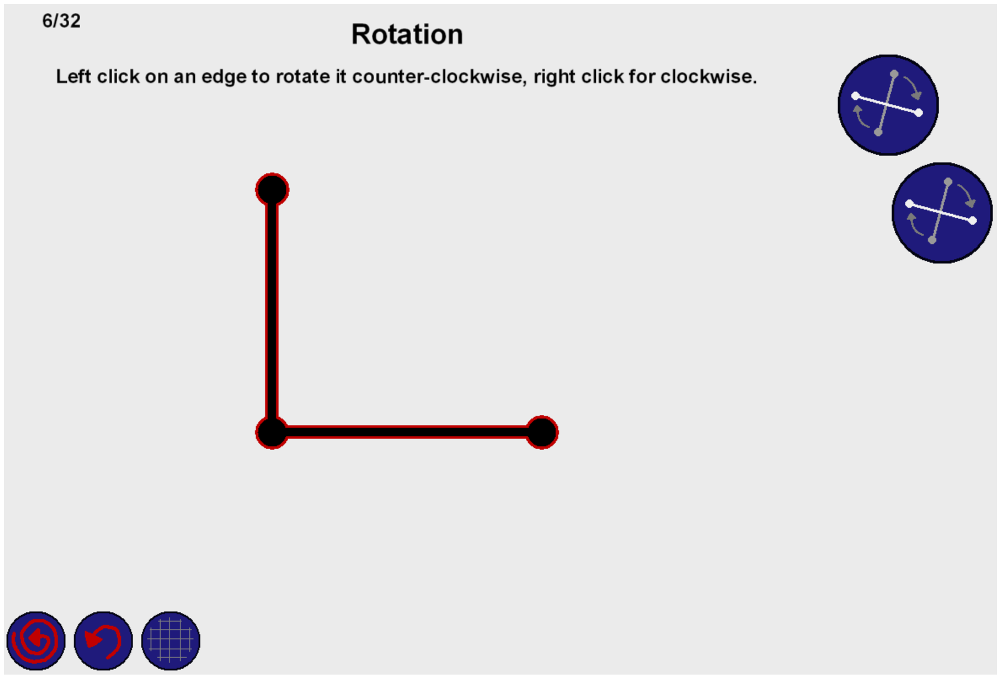

5/32
Rotation
Left click on an edge to rotate it counter-clockwise, right click for clockwise.

Figure 60: Screen 5


6/32

Rotation

Left click on an edge to rotate it counter-clockwise, right click for clockwise.

Figure 61: Screen 6

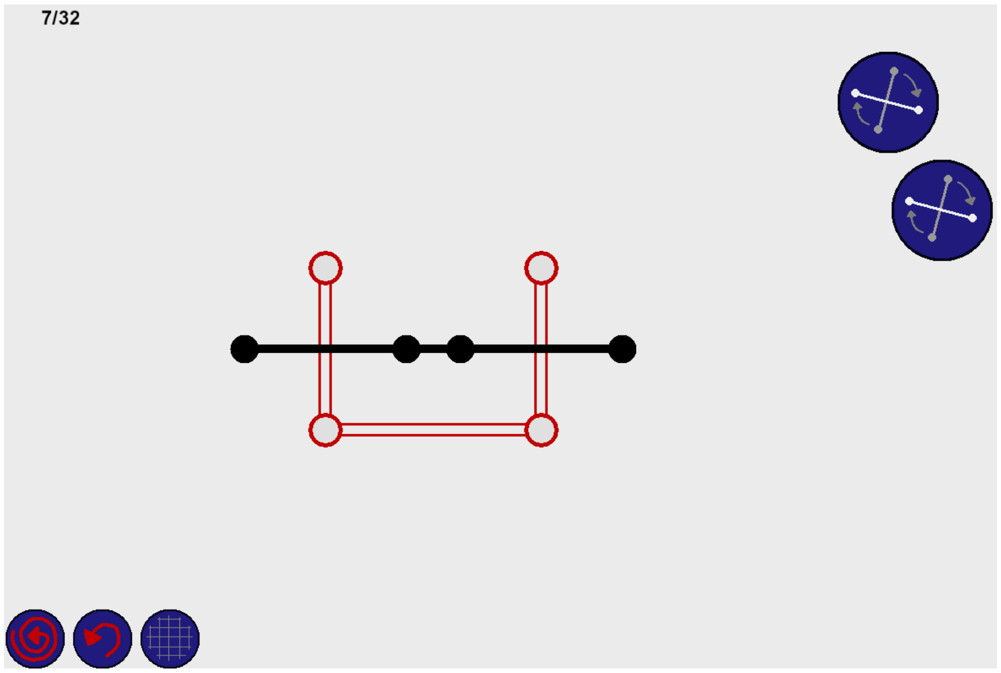


Figure 62: Screen 7

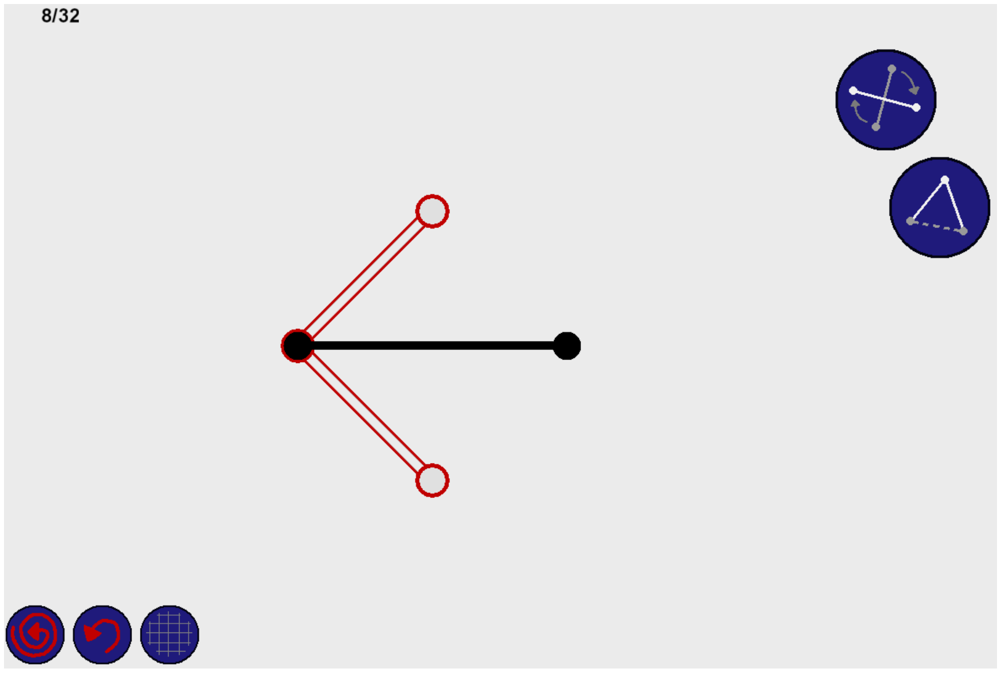


Figure 63: Screen 8

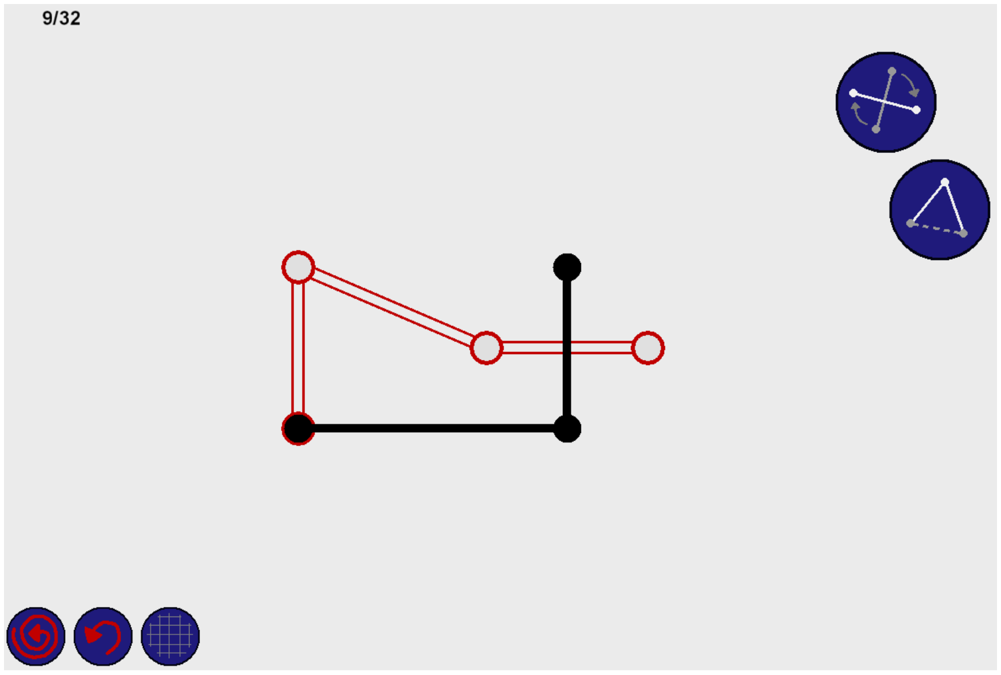


Figure 64: Screen 9

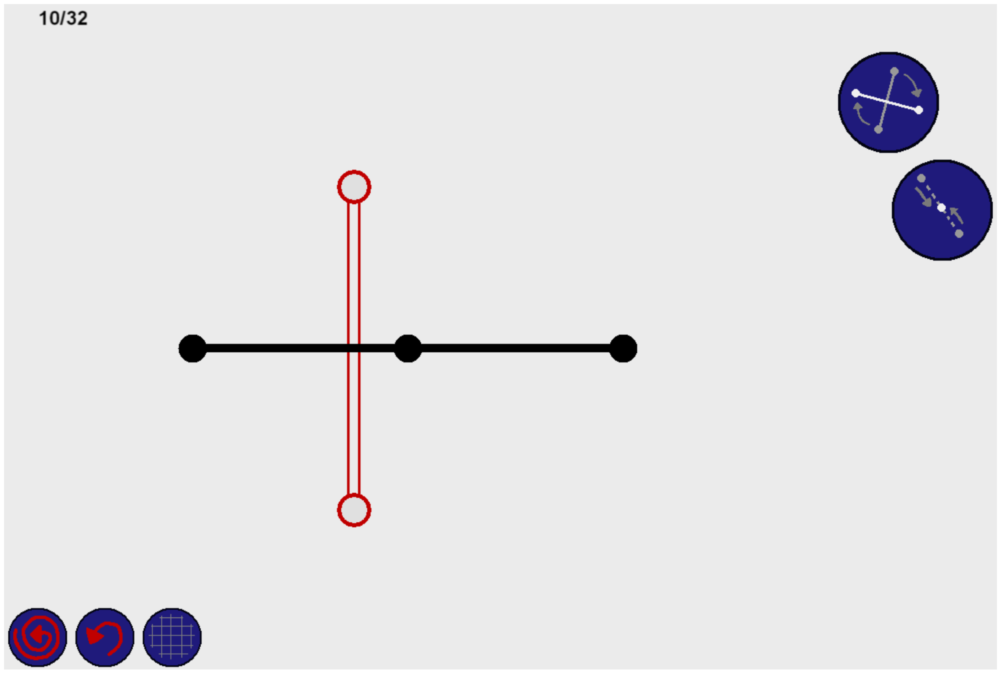
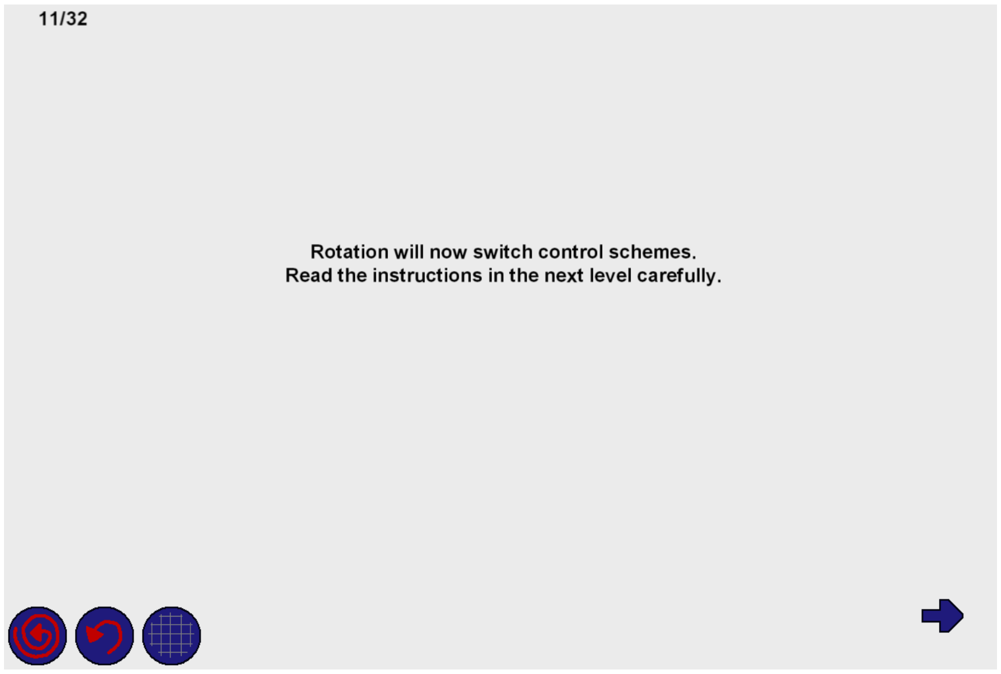
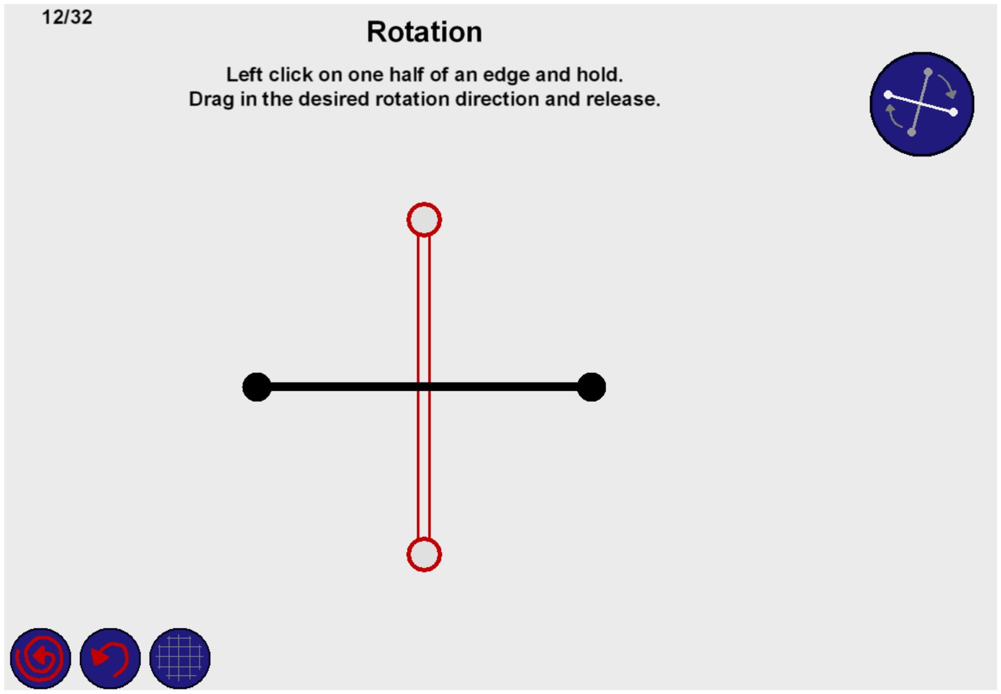


Figure 65: Screen 10



Rotation will now switch control schemes.
Read the instructions in the next level carefully.

Figure 66: Screen 11

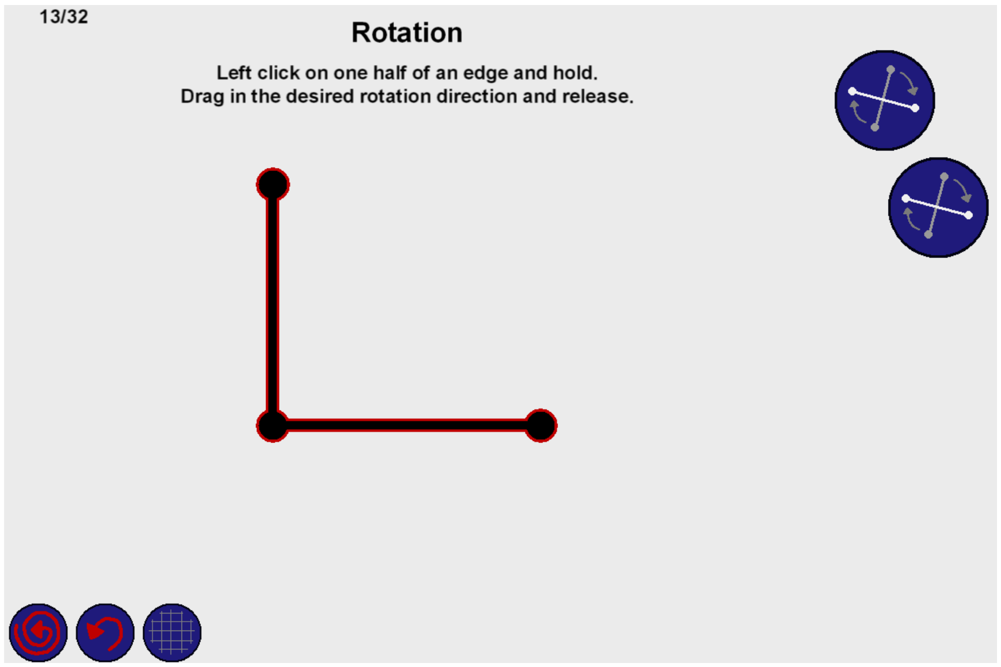


12/32

Rotation

Left click on one half of an edge and hold.
Drag in the desired rotation direction and release.

Figure 67: Screen 12



13/32

Rotation

Left click on one half of an edge and hold.
Drag in the desired rotation direction and release.

Figure 68: Screen 13

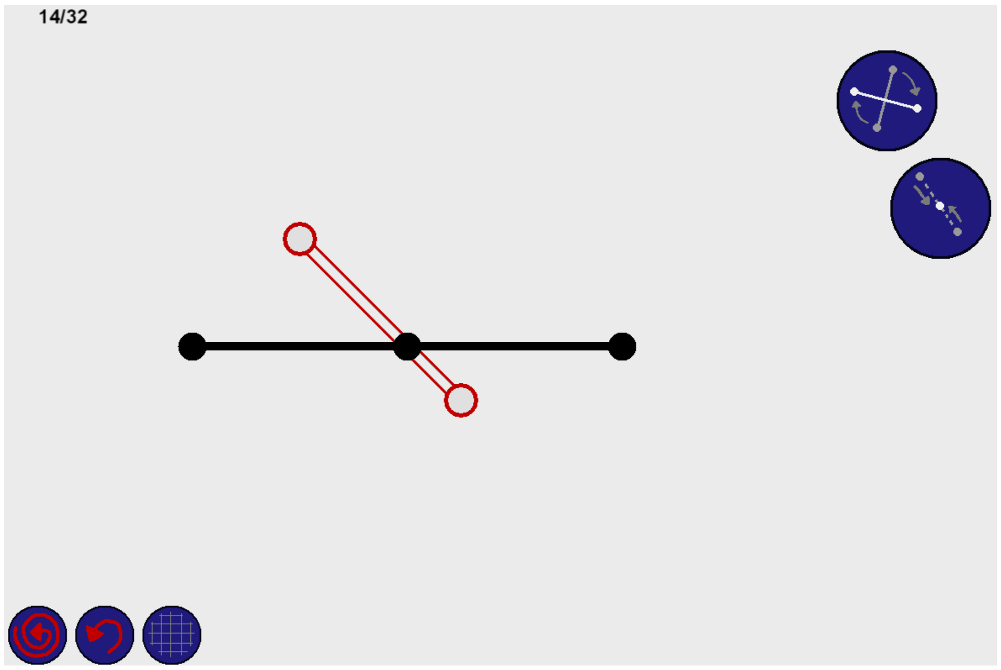


Figure 69: Screen 14

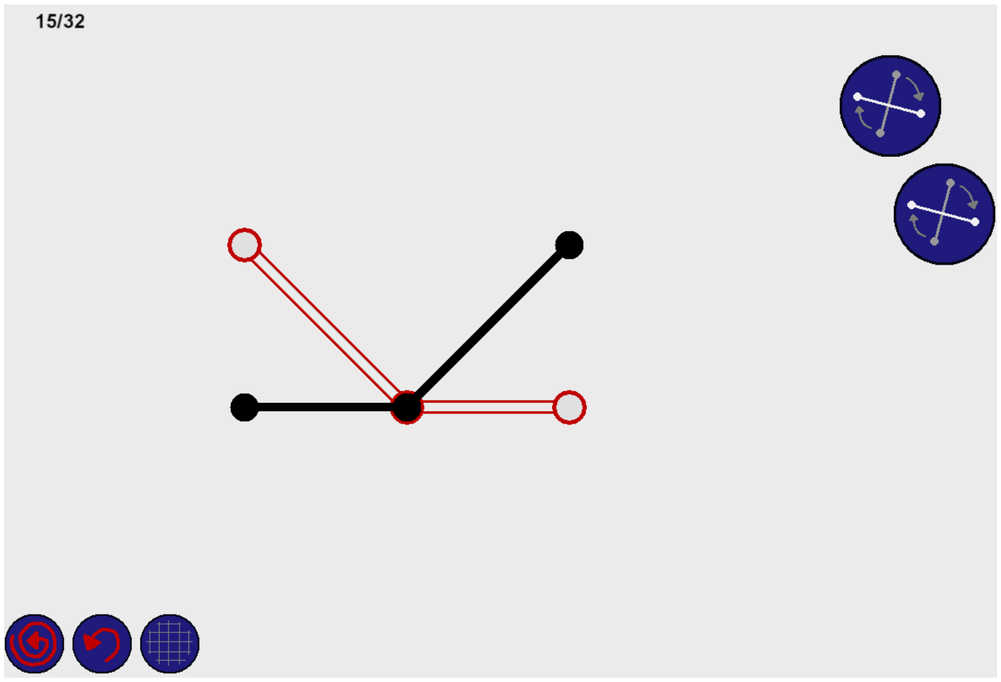


Figure 70: Screen 15

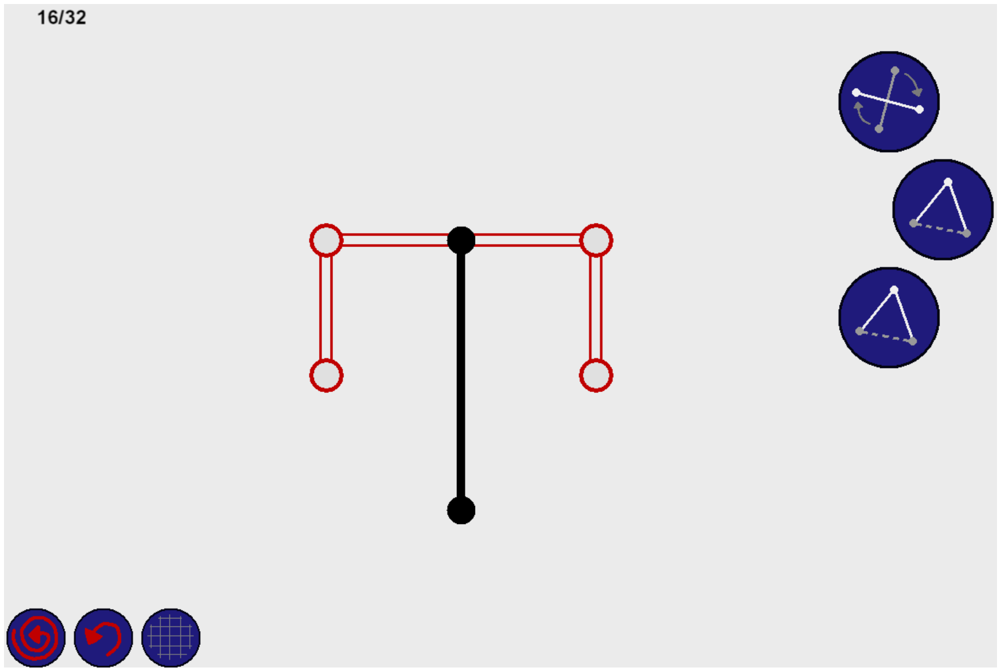


Figure 71: Screen 16

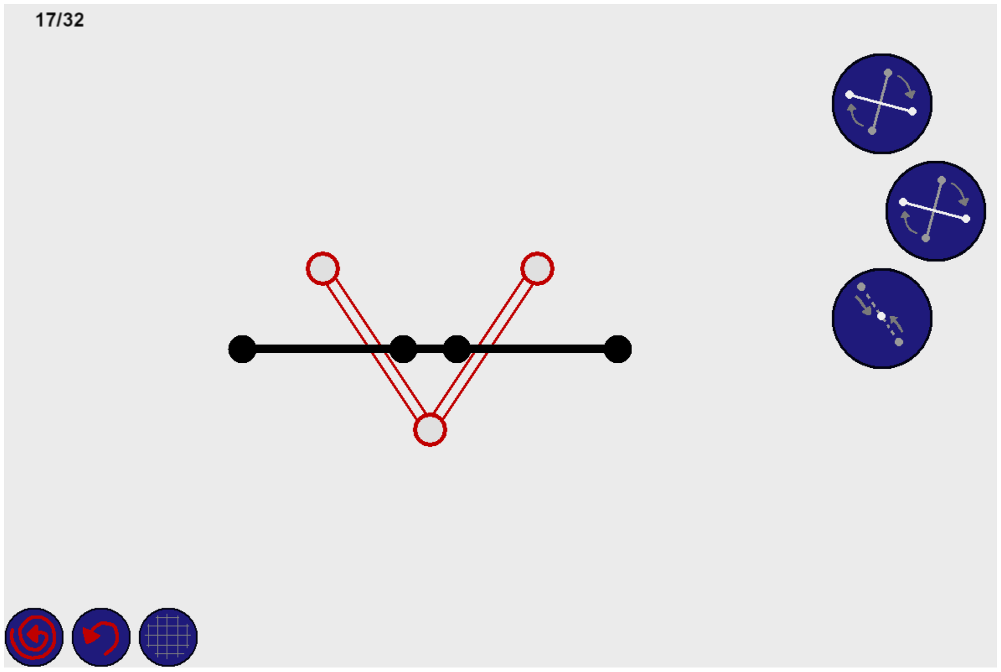
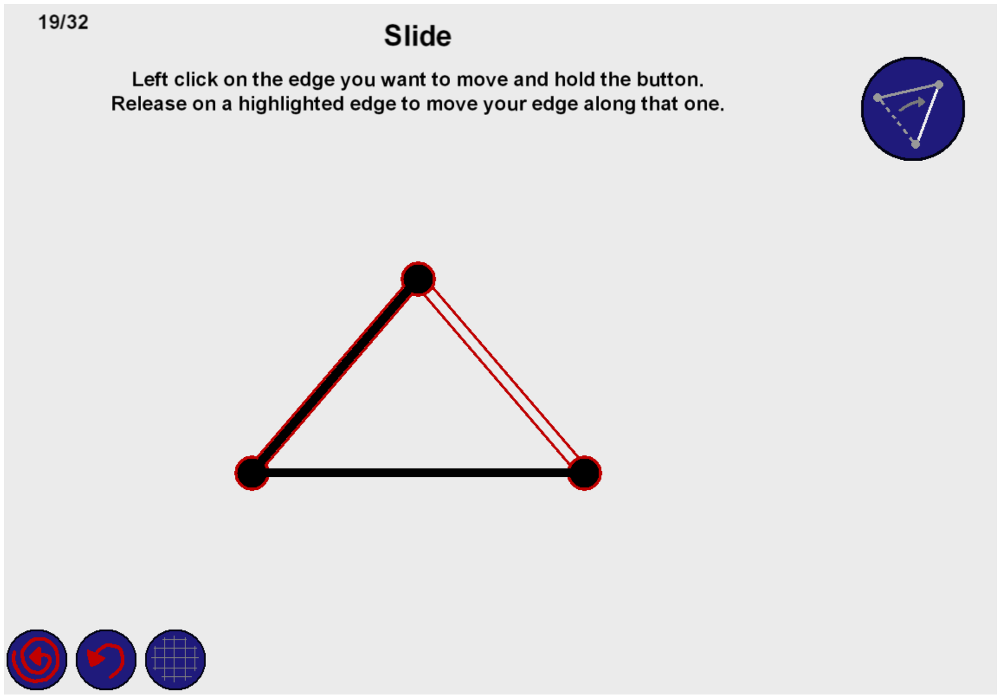


Figure 72: Screen 17

18/32

Please return to the questionnaire.

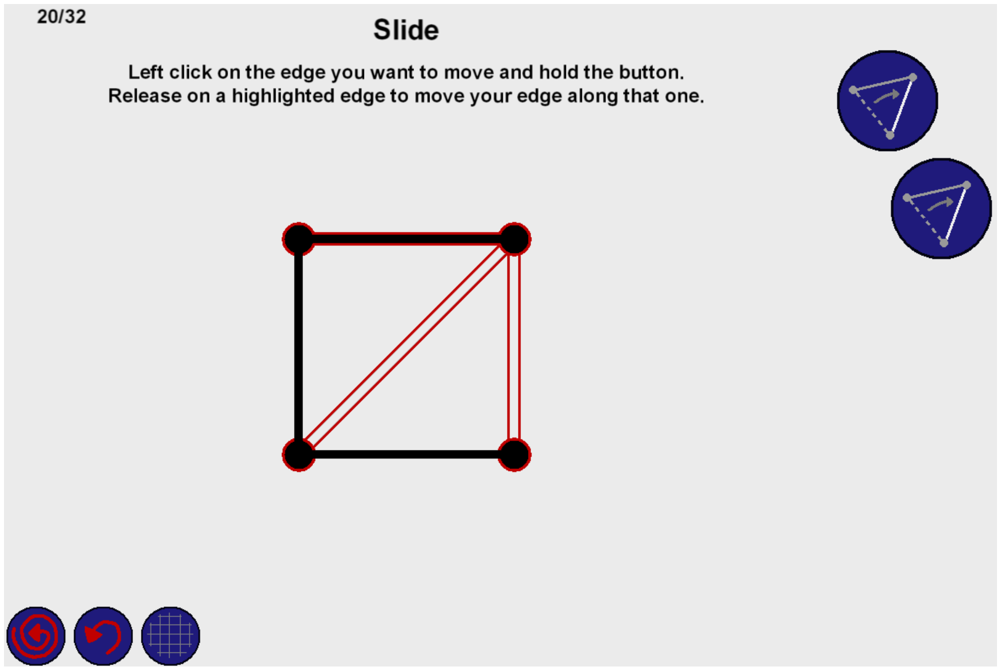
Figure 73: Screen 18


19/32

Slide

Left click on the edge you want to move and hold the button.
Release on a highlighted edge to move your edge along that one.

Figure 74: Screen 19


20/32

Slide

Left click on the edge you want to move and hold the button.
Release on a highlighted edge to move your edge along that one.

Figure 75: Screen 20

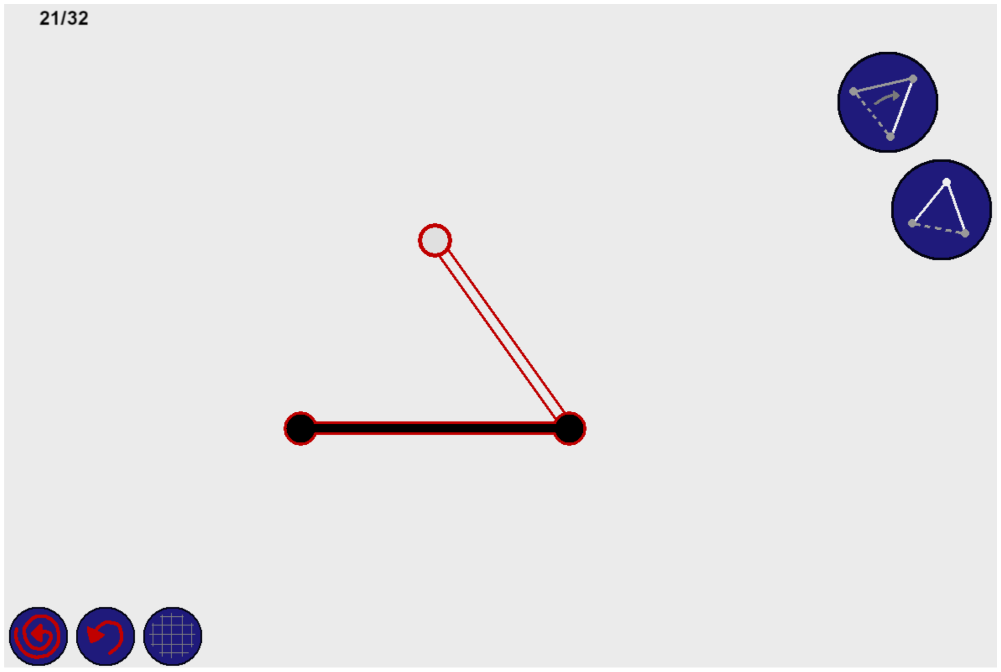


Figure 76: Screen 21

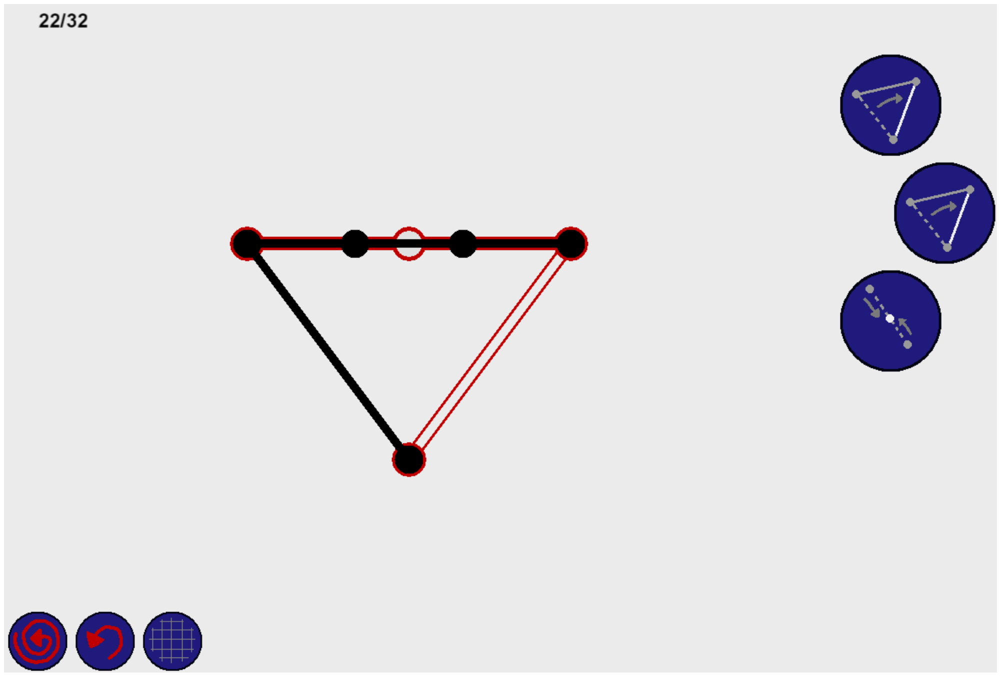


Figure 77: Screen 22

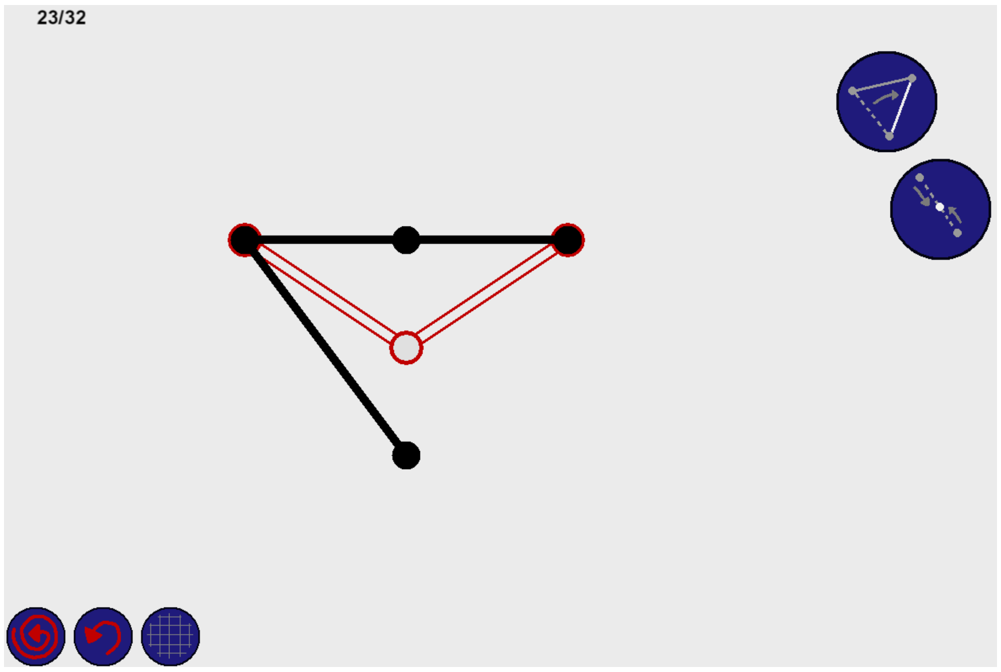


Figure 78: Screen 23

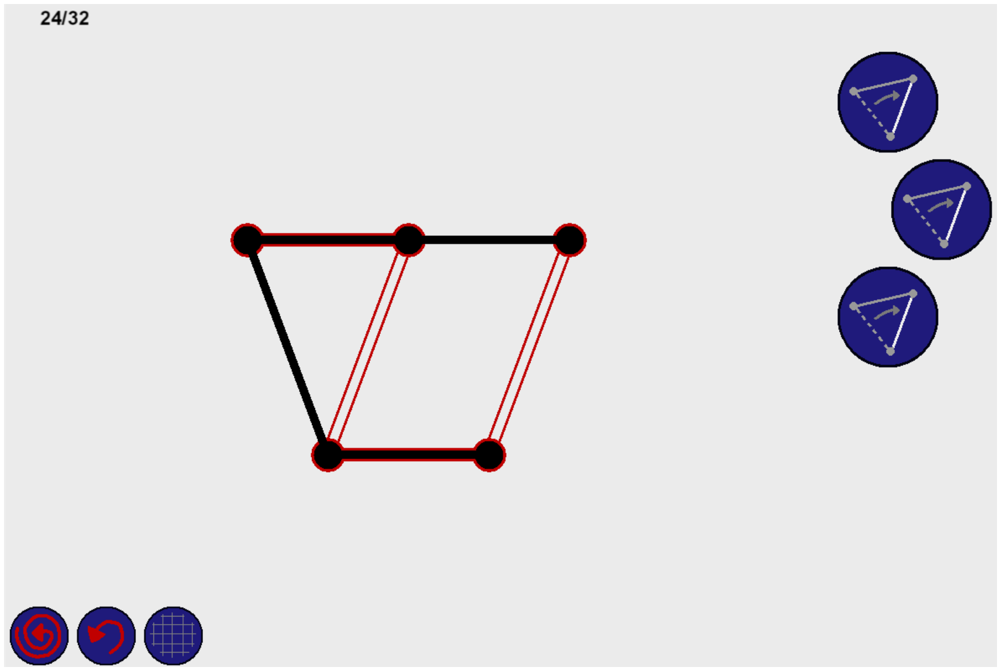
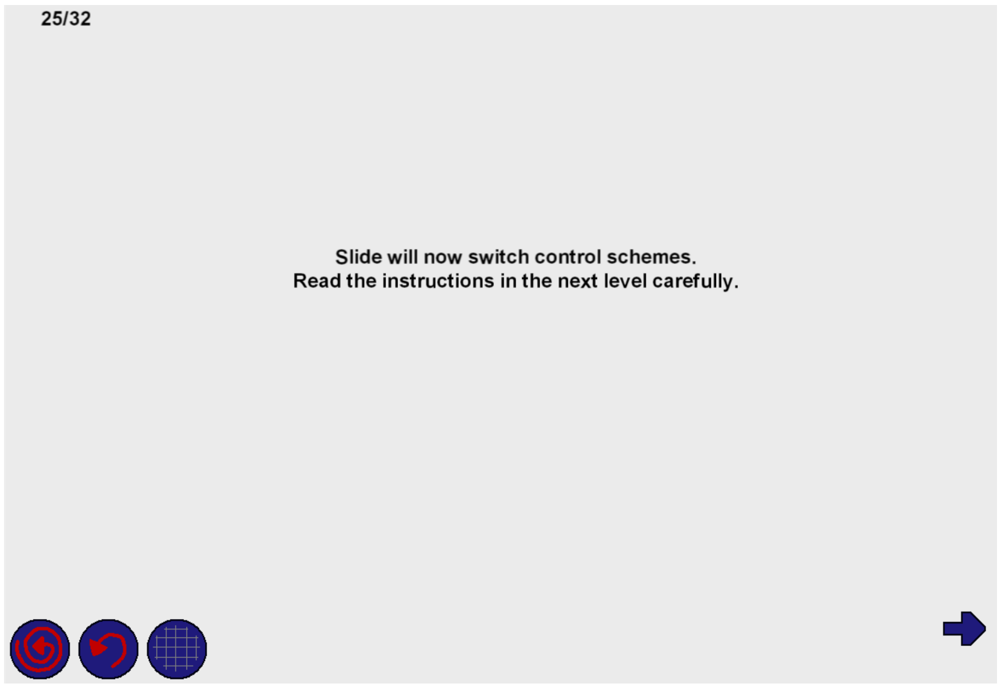
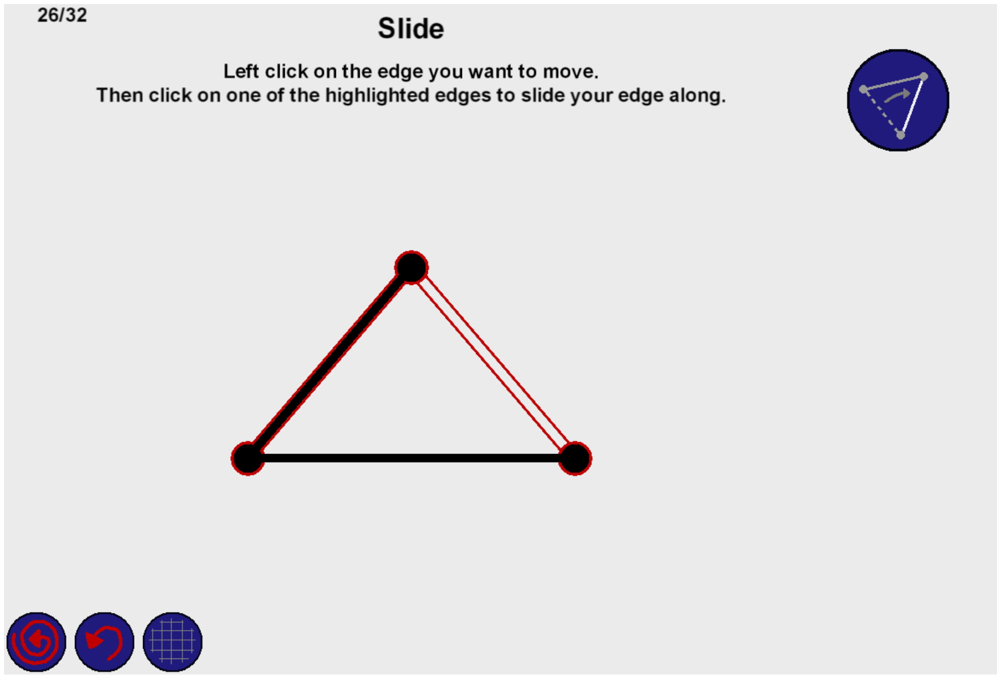


Figure 79: Screen 24



Slide will now switch control schemes.
Read the instructions in the next level carefully.

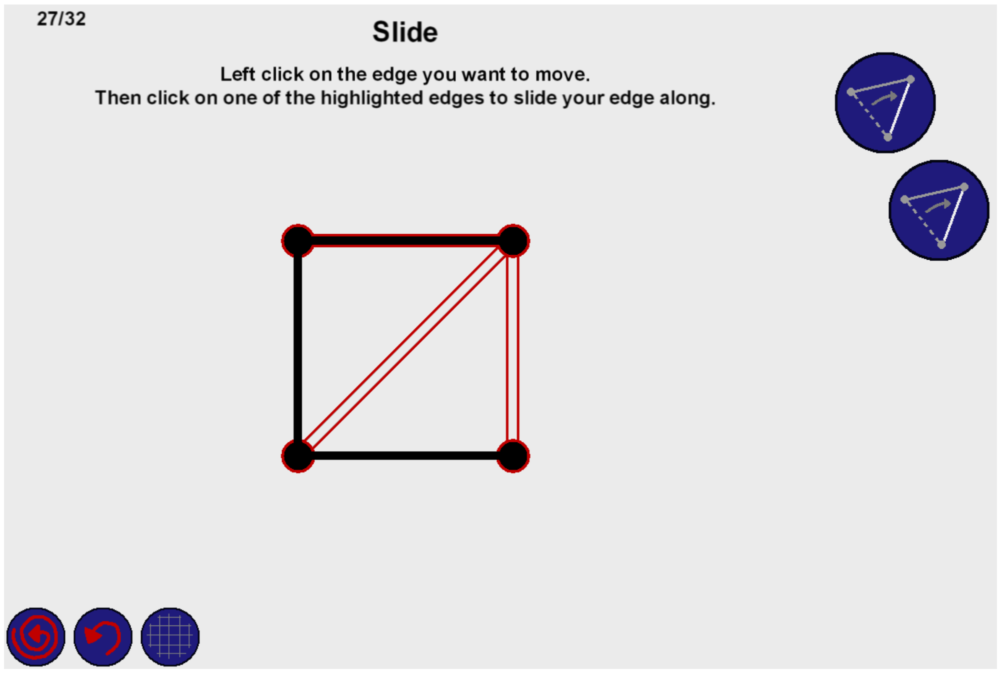
Figure 80: Screen 25


26/32

Slide

Left click on the edge you want to move.
Then click on one of the highlighted edges to slide your edge along.

Figure 81: Screen 26


27/32

Slide

Left click on the edge you want to move.
Then click on one of the highlighted edges to slide your edge along.

Figure 82: Screen 27

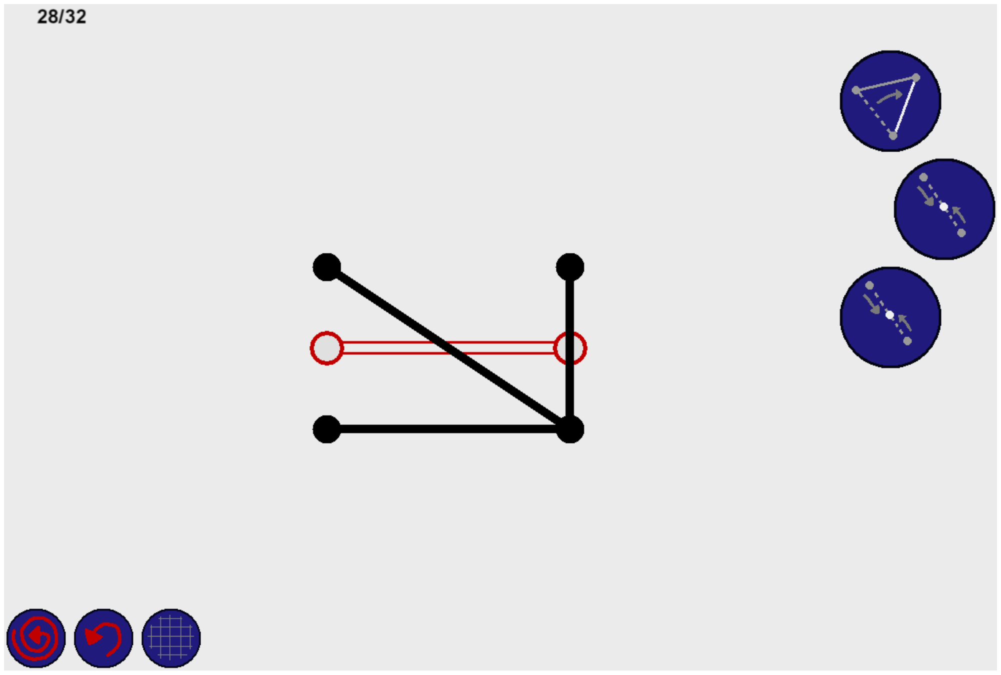


Figure 83: Screen 28

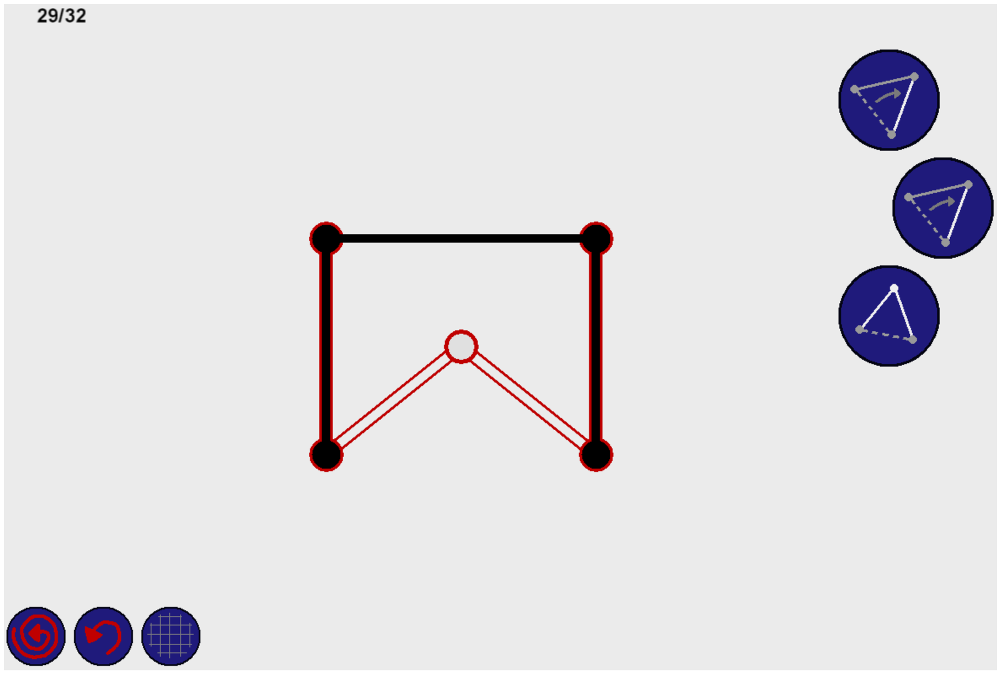


Figure 84: Screen 29

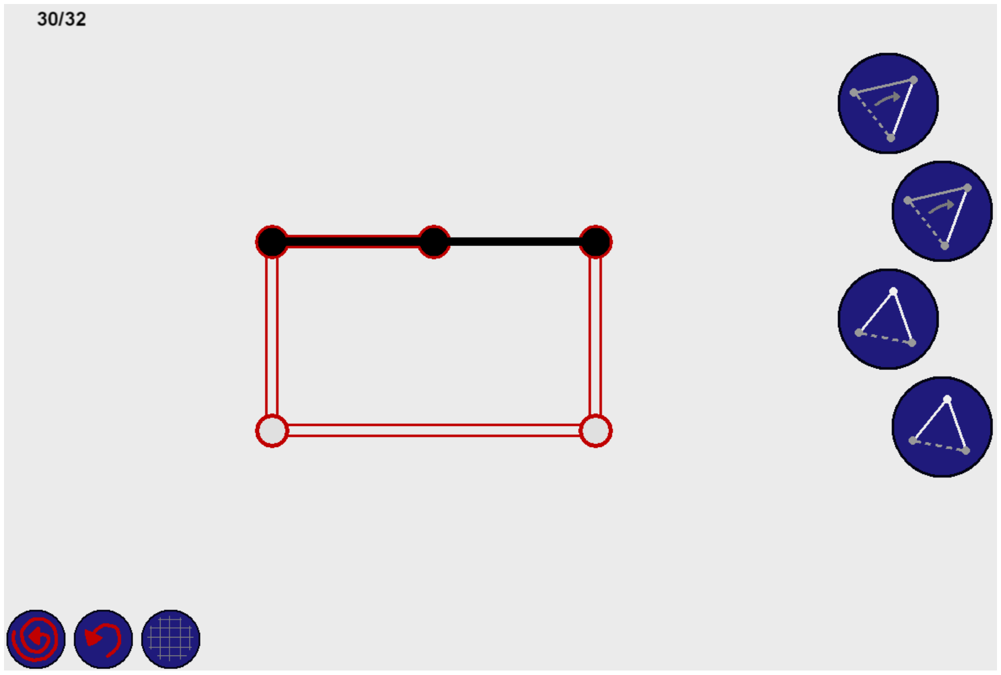


Figure 85: Screen 30

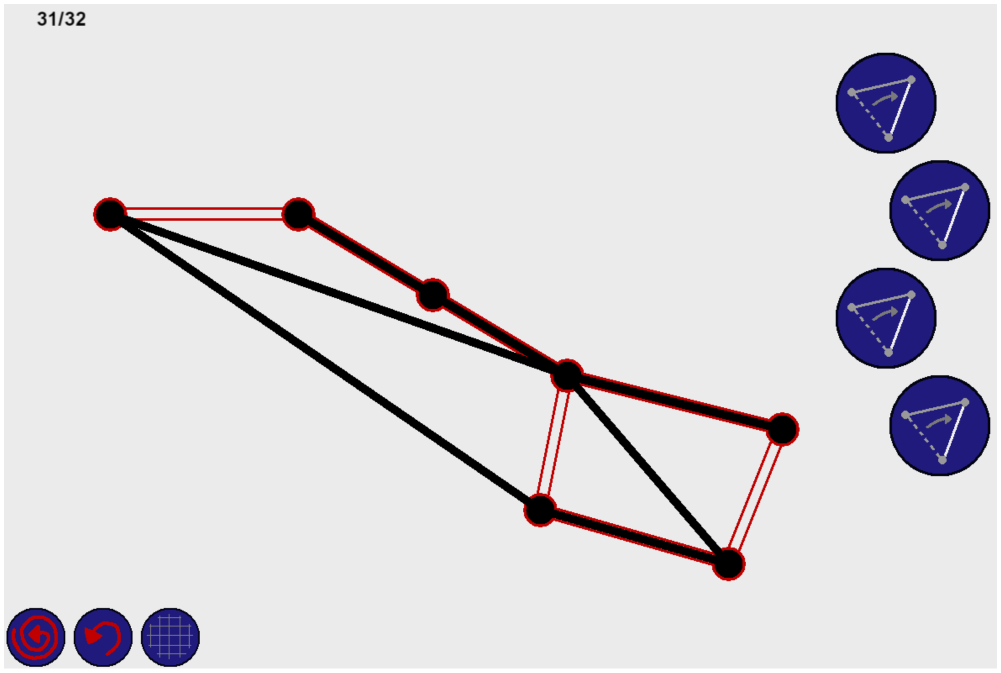
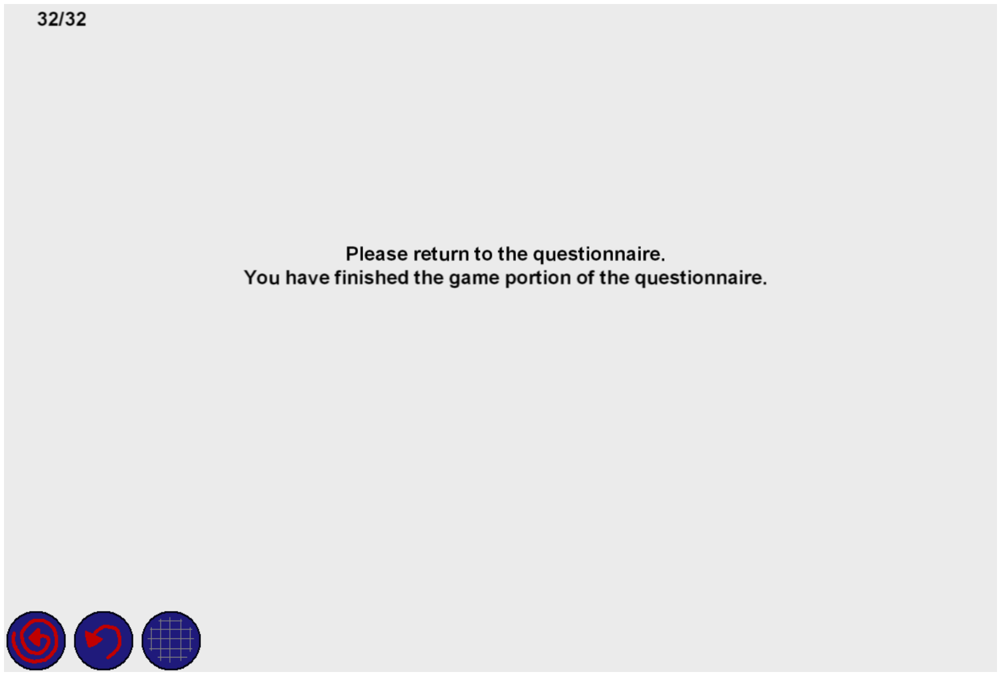
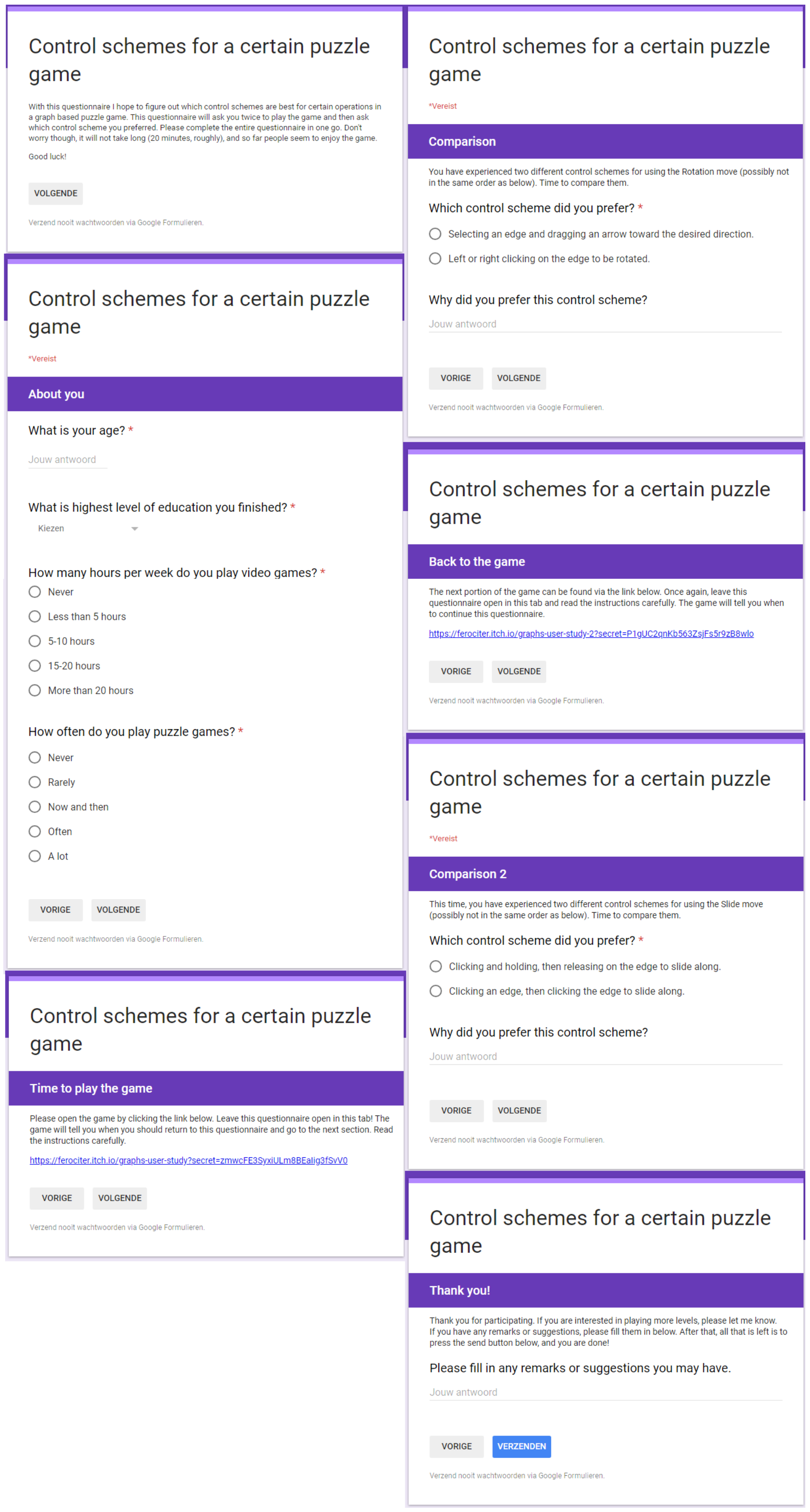


Figure 86: Screen 31


32/32

Please return to the questionnaire.
You have finished the game portion of the questionnaire.

Figure 87: Screen 32



Control schemes for a certain puzzle game

With this questionnaire I hope to figure out which control schemes are best for certain operations in a graph based puzzle game. This questionnaire will ask you twice to play the game and then ask which control scheme you preferred. Please complete the entire questionnaire in one go. Don't worry though, it will not take long (20 minutes, roughly), and so far people seem to enjoy the game.

Good luck!

VOLGENDE

Verzend nooit wachtwoorden via Google Formulieren.

Control schemes for a certain puzzle game

*Vereist

About you

What is your age? *

Jouw antwoord

What is highest level of education you finished? *

Kiezen

How many hours per week do you play video games? *

- Never
- Less than 5 hours
- 5-10 hours
- 15-20 hours
- More than 20 hours

How often do you play puzzle games? *

- Never
- Rarely
- Now and then
- Often
- A lot

VORIGE VOLGENDE

Verzend nooit wachtwoorden via Google Formulieren.

Control schemes for a certain puzzle game

Time to play the game

Please open the game by clicking the link below. Leave this questionnaire open in this tab! The game will tell you when you should return to this questionnaire and go to the next section. Read the instructions carefully.

https://ferociter.itch.io/graphs-user-study?secret=zmwcFE3SyxiULm8BEalig3fSvV0

VORIGE VOLGENDE

Verzend nooit wachtwoorden via Google Formulieren.

Control schemes for a certain puzzle game

*Vereist

Comparison

You have experienced two different control schemes for using the Rotation move (possibly not in the same order as below). Time to compare them.

Which control scheme did you prefer? *

- Selecting an edge and dragging an arrow toward the desired direction.
- Left or right clicking on the edge to be rotated.

Why did you prefer this control scheme?

Jouw antwoord

VORIGE VOLGENDE

Verzend nooit wachtwoorden via Google Formulieren.

Control schemes for a certain puzzle game

Back to the game

The next portion of the game can be found via the link below. Once again, leave this questionnaire open in this tab and read the instructions carefully. The game will tell you when to continue this questionnaire.

https://ferociter.itch.io/graphs-user-study-2?secret=P1gUC2qnKb563ZsjFs5r9zB8wlo

VORIGE VOLGENDE

Verzend nooit wachtwoorden via Google Formulieren.

Control schemes for a certain puzzle game

*Vereist

Comparison 2

This time, you have experienced two different control schemes for using the Slide move (possibly not in the same order as below). Time to compare them.

Which control scheme did you prefer? *

- Clicking and holding, then releasing on the edge to slide along.
- Clicking an edge, then clicking the edge to slide along.

Why did you prefer this control scheme?

Jouw antwoord

VORIGE VOLGENDE

Verzend nooit wachtwoorden via Google Formulieren.

Control schemes for a certain puzzle game

Thank you!

Thank you for participating. If you are interested in playing more levels, please let me know. If you have any remarks or suggestions, please fill them in below. After that, all that is left is to press the send button below, and you are done!

Please fill in any remarks or suggestions you may have.

Jouw antwoord

VORIGE VERZENDEN

Verzend nooit wachtwoorden via Google Formulieren.

Figure 88: The questionnaire used for the user study in Section 4.3

D Level generation pilot test

A pilot test was designed as part of this research, but we never executed it. This appendix contains the design document for this experiment, in case the reader is interested in our initial views on how to set up and perform such a study.

Setup and evaluation in short
The game has a start graph and goal graph. To win, the start graph needs to match the goal graph after using a set of available moves on the start graph: the game can choose between 2 and 4 moves to be available, in this case from 4 different kinds; multiple of the same kind can occur. All available moves need to be used.

To generate a level, option 1 would be to start with a pre-designed "neat" start graph and randomly perform some moves to get a goal graph. Option 2 would be to start at the goal graph, and use moves to create the start graph. For non-reversible moves inverse variants need to be used for option 2.

So which one is "better": option 1 or 2? The differences are:

- Option 1 has a "neat" start graph and most likely a "messier" goal graph, option 2 is the exact opposite.
- Option 1 uses the four moves as intended, option 2 uses inverse variants of 2 of the 4 moves, the other 2 moves are reversible.

To get some idea which option might be better, a pilot test is conducted with at least 5 people. Each will first receive a small explanation, followed by having to finish 6 pre-designed levels so that the mechanics are clear. Next, they will play 10 ad hoc generated levels from pre-made graphs in random order using either option 1 or 2. Then they will play 10 levels using the other option. So the start graphs for option 2 and the goal graphs for option 1 will always be the same, albeit in different order, but the goal and start graphs respectively for these options will be different for each player, since those are generated then.

During the session, the person is encouraged to commentate while playing. Afterwards they will be asked (1) which group of levels they preferred (if any), (2) what differences they noticed, (3) if they noticed anything weird or ugly in any of the levels, (4) what could be improved upon, (5) if they have any other comments not covered by these questions.

This test is mostly meant to provide some insight on which option is probably better, what should be changed in implementation, and what could be improved for a potential full scale user study.

Below are extensive descriptions of the game and the implementation of level generation.

Available moves Four move types are implemented. These moves are:

- Rotate: after selecting an edge, its endpoints will rotate 90 degrees around the centre of that edge. This can be either in clockwise or counter-clockwise direction. [Click left mouse button on an edge to rotate counter-clockwise, right mouse button for clockwise]
- Collapse: after selecting an edge, its endpoints will move towards the centre of that edge and merge. Afterwards, there will be one vertex and one edge less. [Simply left click an edge]
- Addition: The player can replace an edge by a vertex that can be placed anywhere on a grid. There will be 2 new edges: between the new vertex and one endpoint of the former edge, and between the new vertex and the other endpoint of the former edge. [Click an edge and hold. Release to create the vertex]

- Slide: “slides” an edge along another that is connected to one of its endpoints. Not possible when there already is an edge there. [Click an edge and hold. Release on a viable edge to slide along]

To perform a move, it needs to be selected on the right first.

Implementation These are the steps for a level to be generated:

1. A preset graph is selected. These presets are designed beforehand, and consist of graphs that can be considered “nice”: These are mostly symmetrical, and form some simple shape. An example would be a perfectly horizontal edge in the middle of the screen (along with two endpoints of course), or some graph consisting of perfect squares. All vertices are located on the grid.
2. A random move is performed on the graph. Rotation and Slide are reversible, Addition and Collapse are not. The normal options are used for option 1, for option 2 we use the inverse of Addition and Collapse. More specifically:
    - For all Rotation, a random edge is rotated either clockwise or counter-clockwise chosen at random.
    - For all Slide, the action is simply performed on a random edge.
    - For normal Addition (option 1), a random edge is chosen after which a random location for the new vertex is chosen. The criteria for the vertex location is that it is located on the specified grid (coordinates with 32 pixel distances between) and lies on the specified play field.
    - For inverse Addition (option 2), a vertex h that has exactly two edges connected, and where the vertices i and j it is connected to are not connected to each other, is randomly selected. h and the aforementioned edges are removes and a new edge is created between i and j.
    - For normal Collapse (option 1), a random edge is selected and basic collapse is performed.
    - For inverse Collapse (option 2), a random vertex is selected. Two vertices that are directly opposite from the selected vertex, with the same distance to the selected vertex are created. The distance and angle are random, although they need to be located on the grid.
3. After a move, the resulting instance is checked if it is “good”. For it to be good, the following criteria must be met:
    - If r is the radius of each vertex, then no two vertices can be closer than 4r to each other.
    - The angle between any two edges that share an endpoint can be no smaller than 10 degrees.
    - All vertices are within the specified playing field.

    If an instance is not good, we undo the last move and go back to step 2.
4. Repeat steps 2 and 3 randomly between 1 and 3 more times.
5. The start and end graphs are created and drawn, along with the moves used, yielding a playable instance.