https://doi.org/10.7577/formakademisk.4633

Peter Haakonsen
Assistant Professor
Oslo Metropolitan University
peterh@oslomet.no

Laila Belinda Fauske
Professor (PhD)
Oslo Metropolitan University
lailaf@oslomet.no

Learning to create images with computer code[1]

ABSTRACT
Programming is becoming a part of the school curricula in Norway both in lower and upper secondary education – this includes subjects such as art, design and craft. What can programming contribute to the learning processes of these subjects? 'Tinkering' is a creative phase in a learning/working process, emphasising both creation and learning. In this project, visual images are created via computer programming to enhance the main author's learning. The process is structured into stages. The important phases of the learning process are realised as a result of tinkering with existing codes. An important discovery for the learner, and one key aspect of programming images is that, as a mode, it opens up ways to create repetitions effectively, resulting in various patterns. This turned out to be motivating for the learner. This paper discusses tinkering as a learning process that is relevant to programming within art, design and craft education.

Keywords:
Processing, programming, tinkering, learning, digital images

INTRODUCTION

Programming is a skill that is connected to the term 'computational literacy': It is often defined as a 21st-century skill which is increasingly relevant for children to learn in the schools of today (Bocconi, Chioccariello, & Earb, 2018). In Norway, programming has been an elective subject in lower secondary schools since 2016 (Regjeringen, 2017) but it is eventually going to become a compulsory part of the new curricula in both primary, lower secondary and upper secondary schools (Bocconi et al., 2018; European Schoolnet, 2017). This new curricula for primary and lower secondary schools will be implemented in 2020. In Sweden and Finland, programming is already integrated into the creative subjects (Bocconi et al., 2018). In Norway, programming will become a part of the school subject of art and crafts in lower secondary schools (Regjeringen, 2018). A new vocational program that will become available within upper secondary education – called ICT and media productions (IKT og medieproduksjon) – is also due in 2020 (Utdanningsdirektoratet, 2018). These changes in curricula indicate an educational shift in Norwegian schools.

This paper reflects on how programming can be a relevant part of creative school subjects such as art, design and craft. We maintain a special focus on motivation and the learning process when learning to create images with a computer code. The examples in this paper use a programming language with a difficulty level that will possibly be applicable to lower and upper secondary schools, but one that also aims to be transferable to easier block-based languages such as Scratch (https://scratch.mit.edu) – this learning process is also relevant to children in primary schools. Using the creative process as a point of departure, this paper specifically aims to examine tinkering (Berland, 2016) as a learning process that is relevant to programming within art, design and craft education.

PROGRAMMING AS AN EMERGING PART OF ART, DESIGN AND CRAFT EDUCATION

In the report, The Nordic Approach to Introducing Computational Thinking and Programming in Compulsory Education (Bocconi et al., 2018), the authors investigate how programming and other new subjects are implemented – or about to be implemented – into Nordic schools. When commenting on the British situation, the report shows that programming emerged in the British curricula during 2014. This initiative is a part of the process of implementing 21st-century skills into the curricula, a set of skills to prepare today’s children with the ability to adapt to their future occupations and social life. This involves ‘promoting foundation skills, digital literacies, higher order thinking competencies as well as social and emotional skills’ (Bocconi et al., 2018).

One key term in the report is ‘computational thinking’ (CT), which can be seen as an umbrella term that includes programming. In Norway, the term ‘algorithmic thinking’ is used instead of CT – it is not a direct translation. There is no need to compare these two terms with each other here. We will say, however, that they are similar in the sense that they are both connected to the modern problem-solving skill set; programming, along with digital competence, are both included in this skill set (Bocconi et al., 2018). The new curricula that are to be implemented into Norwegian schools in 2020 are the result of a comprehensive process that was carried out over several years. Ludvigsen-utvalget (English: The Ludvigsen Committee) made two public NOU [English: Norwegian public statement] reports, the first in 2014 and the second in 2015. In these reports, 21st-century skills and in-depth learning are both emphasised (Ludvigsen, 2014; NOU 2015: 8). These two reports were the starting point from which the ongoing process aiming towards the new Norwegian curricula originated. The Norwegian process illustrates a common trend in Nordic countries:

> A common trend emerging from analysis of policy documents and interviews with experts is that Nordic countries have included CT and Programming as part of a broad and evolving definition of digital competence, one that embraces key 21st century skills like problem solving, logical thinking and creativity. (Bocconi et al., 2018, p. 9)

This paper argues that programming as a subject in compulsory education may prepare children not just to be digital consumers but also to be digital creators. This argument is formulated in the light of Richard Sennet's statement warning about the dangers when people let the machines do the learning and the people become passive consumers of an expanding competence (Sennett, 2008). With this in mind, this paper aims to contribute to the knowledge-building discourse (Scardamalia & Bereiter, 2010) on programming and 21st-century skills in art, design and craft education at different levels.

Tinker

There are many different programming languages. Some are text-based while others are block-based. What they have in common is that the programmers get instant feedback from the material when creating codes: Either the code works, or it doesn’t. Making things with a programming language presupposes computational literacy. Berland (2016), following diSessa (2001), describes material literacy as one of three component aspects of computational literacy:

> This kind of material literacy can come in many forms: through facility in making a complex spreadsheet in Excel; from the ability to pack your luggage using the most efficient procedure; or through evaluating various data structures in program code. Most of what is considered to be material computational literacy is simply the ability to write and craft program code to express oneself and build usable artifacts. (Berland, 2016, p. 197)

Berland's description can be interpreted in relation to the 21st-century skills and upcoming curricula that are soon to be included within Norwegian schools. Digital competence has a place in art, design and craft subjects at all levels; for example, children in primary schools making stop-motion animation with an iPad app and plasticine and those in upper secondary schools manipulating photographs in Adobe Photoshop or 3D-models in SketchUp or Tinkercad – these examples combine the material with the digital. Tinkering, according to Berland, Martin, Benton, Smith and Davis (2013) and Berland (2016), is a way of playing with a material without necessarily knowing the outcome, thus enacting itself as distinct from experimenting. Tinkering can be defined in various ways. However, Berland and colleagues claim that there are certain common benefits to be drawn out of the various definitions:

> Tinkering describes both (a) an orientation and (b) a set of activities. […] The set of activities described as tinkering include trial and error, messing around or fussing, finding and using feedback mechanisms (such as testing), or combinations of those activities. (Berland et al., 2013, p. 568)

This paper illuminates the process of producing digital images created by computer codes through tinkering. A basic assumption is that tinkering can enact itself as a good learning process when introducing programming into creative subjects within school. In this paper, programming language becomes the only tool in a creative process, excluding all elements of physical materiality: It becomes digital imaging within its own premises, with tinkering embodying the mode through which the learners engage with programming.

Approach

This paper illuminates a case where the main author (who has taught digital design software at university level for several years) explores the learning process that stems from creating images through computer programming. His prior knowledge includes a deep understanding of how digital colours behave on a screen with red, green and blue (RGB) as values between 0 and 255. Together with other basic understandings of what a digital image is and how it contains a given number of pixels, this knowledge works as a foundation when starting to program images. Within Processing, dealing with these numbers is common and they make sense to a learner of programming in an environment where everything else is new and unknown.

The main author's practical experiences and reflection on actions (Schön, 1991) are articulated through a dialogue between a colleague and the co-writer of this paper. The case is retrospectively divided into stages, where the main author's 'basic textbook learning' started in stage one. In stage two, playing and tinkering with advanced codes led to new learning outcomes for the author. In the third and current stage, he goes deeper into a function that the programming language accrued.

PROCESSING

Processing is a text-based programming language developed for artists and designers. As its founders, Casey Reas and Ben Fry (2009), describe, it is easy to start with – anyone can create an image – and it has been developed with artists and designers in mind. It could have been interesting to write about block-based coding (e.g. *Scratch*), which is suitable for younger children, but the inherent possibilities of *Processing* make it more applicable to this case.

Roger Antonsen (2018) uses *Processing* to visualise complex mathematical phenomena. A recent example of how one might use *Processing* is his creation of eight images which visualise different card shuffles. Antonsen's example shows that *Processing* holds many possibilities in terms of creating images that act in ways that are different to what you may achieve in other image-editing programs such as

Adobe Photoshop or *Illustrator,* as these programs have a visual approach to image editing. Processing is designed to create and modify images, and it has the potential to create complex and generative art through coding.

Stage one: Basic textbook learning

Driven by his ambitions to create complex interactive images, the main authors' learning of *Processing* started by reading the different handbooks as well as examining *Processing'*s own online resources (Reas & Fry, n.d.). It began as a hobby or leisure activity for the first few years of usage but later he began to implement *Processing* into his professional work. From creating simple rectangles and ellipses on backgrounds, a basic and well-known illustration (of how three circles of RGB colours mix into cyan, magenta, yellow (CMY) and white as they overlap) was drawn (Figure 1). This was a simple illustration based on the foundations that were established through the textbooks – for example, how to place the circles on the background with given *x* and *y* coordinates related to the canvas's size in pixels. These foundations also included how to write the three RGB colours. To make the colours mix and achieve the CMY palette, a function telling the colours to blend according to additive colour mixing was included. The blend mode (ADD) was also similar to software such as *Photoshop*. As in *Photoshop*, it is important to use a black background colour when mixing colours additively: Avoiding white is a key part of this process.

This is an example of the practice and testing stage. The goal was to write codes such as this without looking at recipes or textbooks. This kind of visualisation is based on additive colour systems (Kuehni, 2004). As a teacher of digital media at university level, the main author had made such visualisations using *Photoshop* in the past. At this stage, there is very little creativity involved. The focus is on learning how to use *Processing* and to gain a fundamental understanding of what it takes to write simple programs. The target was to let using the blend modes act as a method through which one could experiment with colours. In addition, different ways to create moving images were tested. An easy way to do this was to replace the *x* and *y* coordinates (in numbers) with *mouse x* and *mouse y*, making a shape's coordinates change simultaneously with the mouse cursor. The drawn shapes could then move around on the screen.

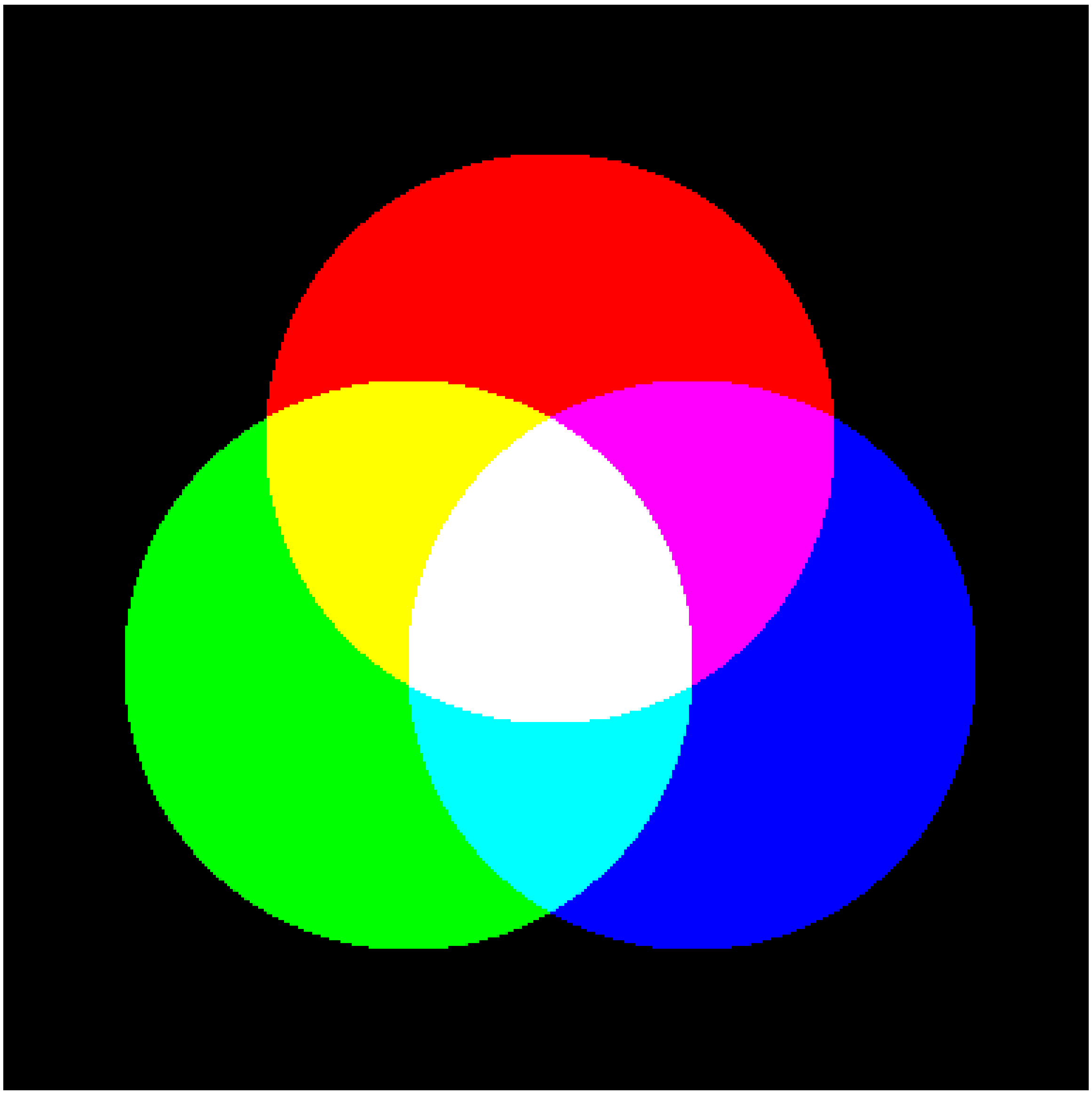


FIGURE 1. RGB circles with a blend mode set to additive colour mixing creates CMY and white.

Stage two: Playing and tinkering

After a period of learning the basics, it was time to create something more advanced: What if these circles could move and bounce across the screen in different ways? And could these movements be influenced and modified when exposed to stimuli such as sound? Could they first orbit or jump across the screen and then change directions if someone were to shout into a microphone? After searching online, different interactive examples were found that were suitable for modification. Three main products came out of this process, the first of which was a depiction of three RGB circles bouncing across a black canvas, mixing colours additively as they overlapped (similar to Figure 1 but with moving parts).

Advancing from the code for bouncing circles, this project moved into a new stage. At this point, three circles were coloured with RGB contours but without a fill colour. They were bouncing across the screen in the same way as in the previous figure. This time the image became successively lighter as the circles overlapped because the background colour was not in the looping part of the code – which means it is only 'played' once in the loop. The circles were also allowed to change size and proportions when moving the mouse: Both *mouse x* and *y* had an impact on both size and shape. Finally, a reset function for when the mouse button was clicked was created, making the screen darker when it started to become white (Figures 2 and 3). The mouse actions were quite basic and easy to replace with fixed *x* and *y* numbers.

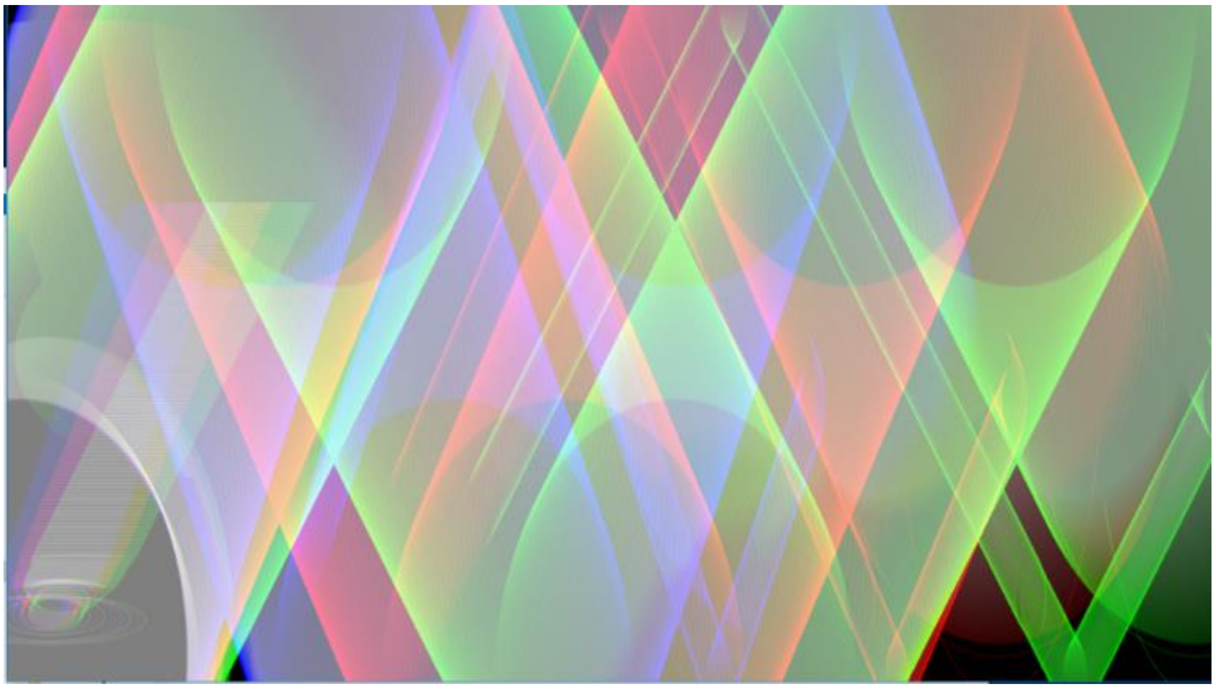
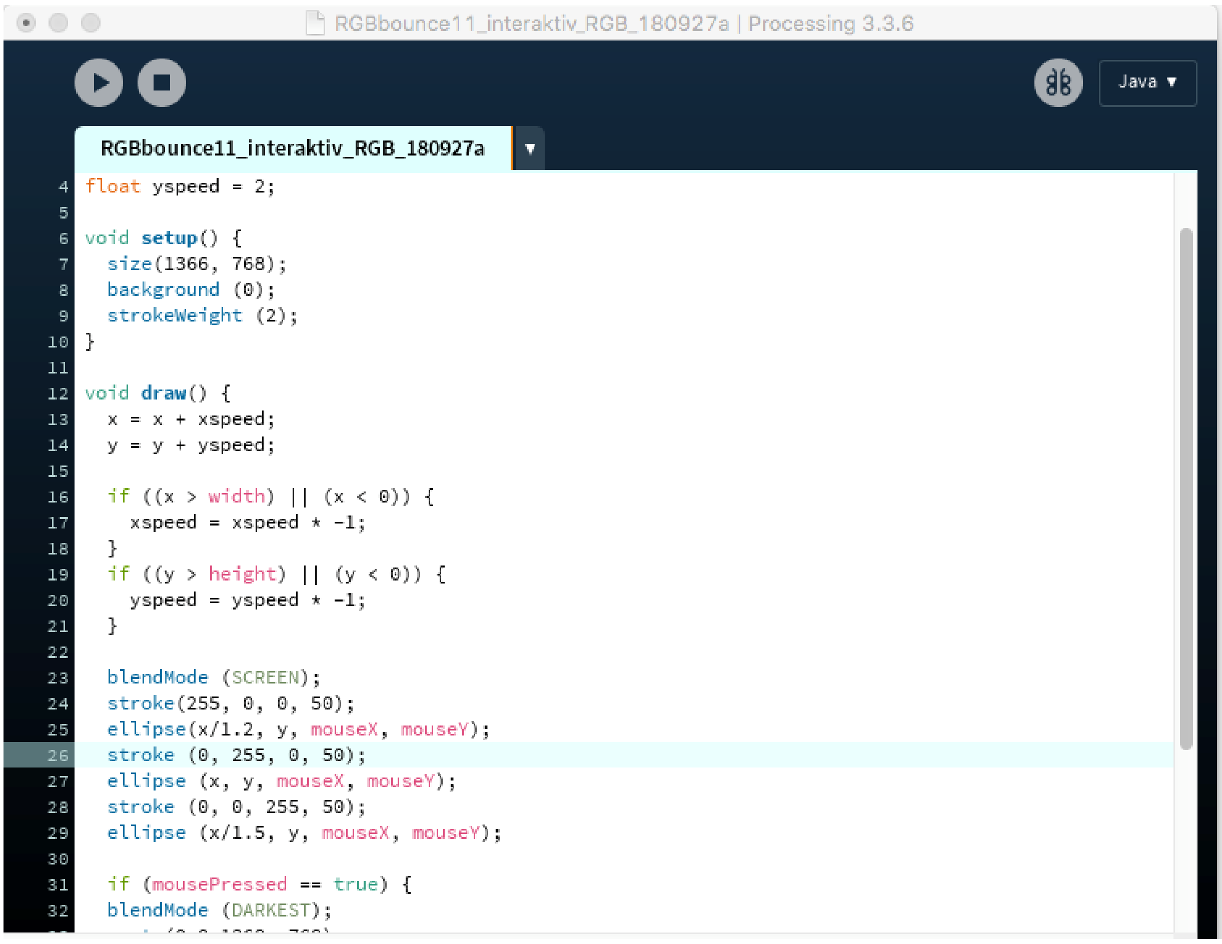


FIGURE 2. RGB-contoured circles on a black canvas, creating secondary colours (CMY). The canvas gets successively lighter as the circles move across the screen. The different RGB shapes move diagonally and you can alter the pattern by moving the mouse.


RGBbounce11_interaktiv_RGB_180927a | Processing 3.3.6
Java
RGBbounce11_interaktiv_RGB_180927a

```
float yspeed = 2;

void setup() {
  size(1366, 768);
  background (0);
  strokeWeight (2);
}

void draw() {
  x = x + xspeed;
  y = y + yspeed;

  if ((x > width) || (x < 0)) {
    xspeed = xspeed * -1;
  }
  if ((y > height) || (y < 0)) {
    yspeed = yspeed * -1;
  }

  blendMode (SCREEN);
  stroke(255, 0, 0, 50);
  ellipse(x/1.2, y, mouseX, mouseY);
  stroke (0, 255, 0, 50);
  ellipse (x, y, mouseX, mouseY);
  stroke (0, 0, 255, 50);
  ellipse (x/1.5, y, mouseX, mouseY);

  if (mousePressed == true) {
  blendMode (DARKEST);
```

FIGURE 3. Screenshot from Processing showing parts of the code for Figure 2.

At this part of the second stage, a search was conducted online for 'fancy' code scripts with the goal in mind being to combine the RGB colour mixing with complex movements and interactivity – this resulted in the described examples. Here, tinkering with the codes and not always knowing what would happen was an important part of the approach. Testing (instead of reading) the theory was a crucial working process, advancing the work's progress several steps further.

Soon, however, the process stagnated. The code making the circles bounce automatically was too advanced to understand. It was easy to modify the circles to bounce slower or faster, but it was not easy to write new lines of code that would make the circles move automatically and exactly as was wanted. This experiment was put on hold so as to try out a new approach. Figure 4 is based on the available codes that were found online while looking for a way to create transitions between two colours. The code was tinkered with to change the colours and the principles behind this function – called *lerp* (Reas & Fry, 2006) – which was also learned by the main author through this process.

A milestone in the learning process was thus achieved. Other parts of the code started to become interesting. Another function in this code was *easing*, which could make the red circle move more elegantly across the screen as one moved the mouse along the *x* and *y* axis. Without understanding this function well enough, a closer look at a part of the code revealed the *for* structure. This structure seemed essential within different settings but had not been explored by this point. By the end of this stage, it seemed possible to start to learn how to use it. Now it was time to go back to the books and the online resources (the reference guide at processing.org), reading and trying to understand the logic behind it.

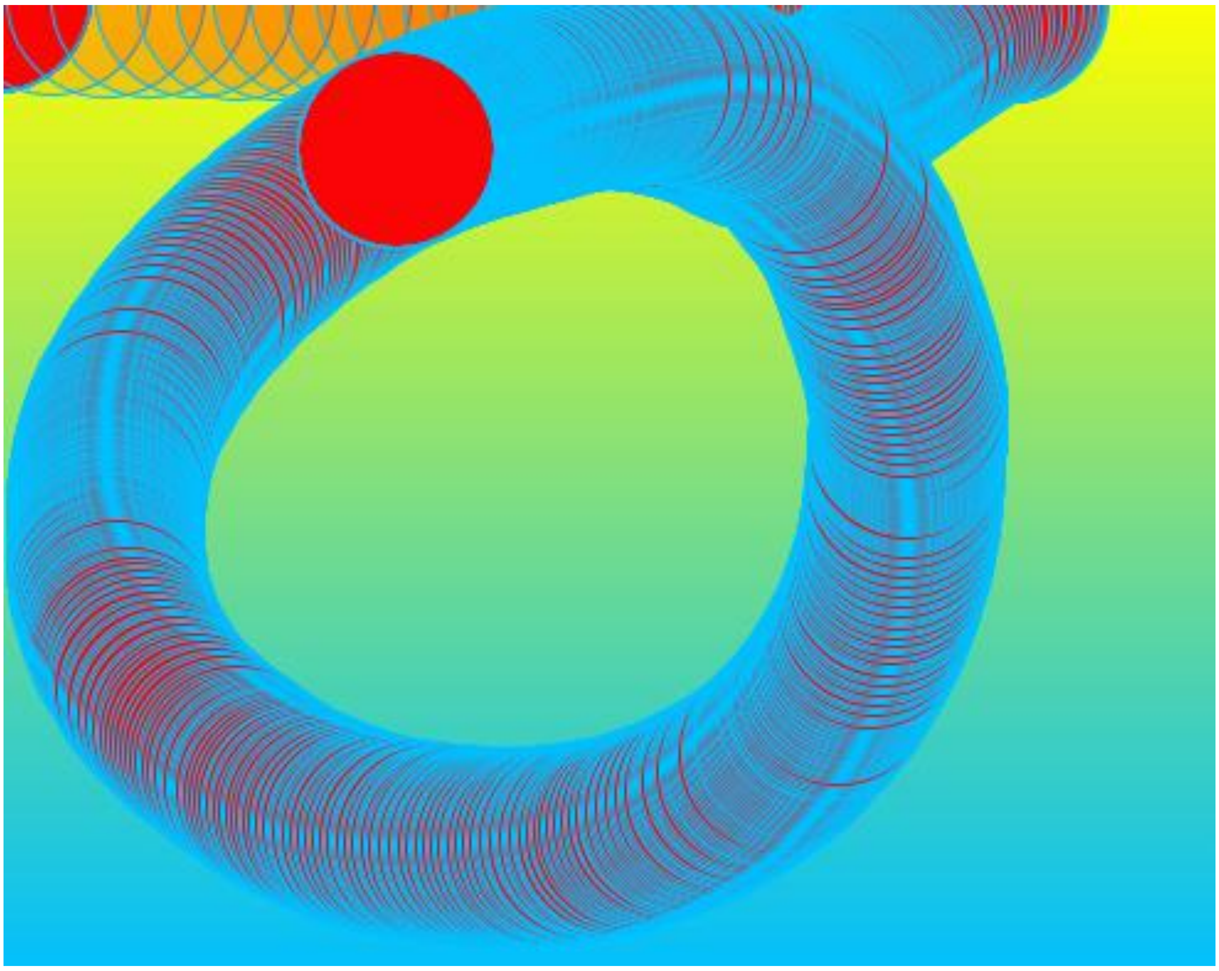
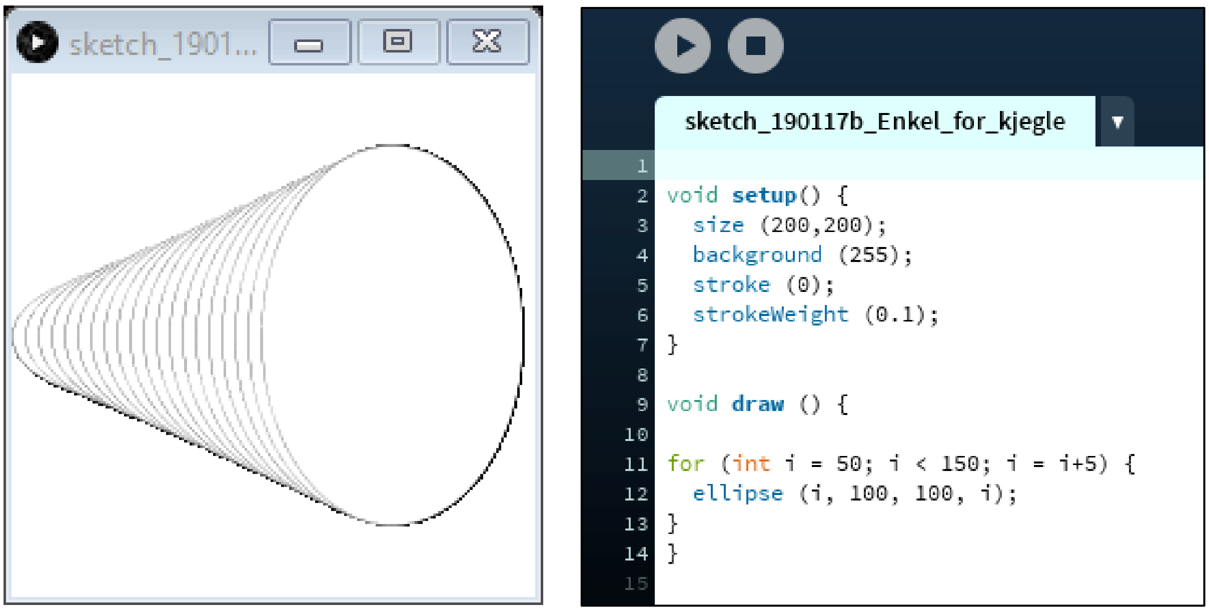


FIGURE 4. Colour transitions in the background and a circle controlled by the mouse (the background is not in the loop, so the circle leaves a track).

Stage 3: Back to basics: Going deeper into a function in the programming language

The last experiments may look 'fancy', but they were mainly products that stemmed from modifying and playing with existing codes to mix colours in different ways to try different colour transitions and to play with movement and interactivity. Here, tinkering (Berland, 2016) was the main approach. A motivating element during this stage was to play with something advanced. The main author employed this approach instead of continuing to learn programming systematically (as in the approach in stage one).

The original web resources and the textbook (Reas & Fry, 2006) began with exercises where one drew lines in order to understand the fundamental basics. This was achieved early on in stage one but, as the aim was to learn fast, this approach was abandoned in stage two. By stage three, going back to the basics was essential for further development. It was crucial to look closer at the *for* structure. The logic behind *for* is explained like this: *For* (*i* = init, test, update) (Reas & Fry, 2006). This means that the variable *i* is initiated (e.g. set to a specific value), then tested (stop if this value is higher or lower than a certain number) and then updated (if the test was positive, update with a specified addition). With this structure, it is possible to let a shape copy itself in a specific rhythm, thus making it possible to create patterns. Apparently complex images can be made with just a few lines of code. By this point we were getting closer to formulating the essence of how programming can be an incredibly useful tool in certain situations; its effectiveness. The next example is the first independent exploration of this function, written from scratch and without looking to references. It shows a small ellipse, copied and enlarged in a specific rhythm.



```

void setup() {
  size (200,200);
  background (255);
  stroke (0);
  strokeWeight (0.1);
}

void draw () {

for (int i = 50; i < 150; i = i+5) {
  ellipse (i, 100, 100, i);
}
}
```

FIGURE 5. Duplicating an ellipse with a few lines of code.

Figure 5 is an example of an effective way to create different patterns with a few lines of code. This was a milestone in terms of the learning process. It was interesting to write such codes and to experiment and tinker with them, creating new patterns – as opposed to tinkering with advanced code scripts that were found online. After some testing in regard to how to repeat lines according to the *for* structure, the ability to create a grid with a little twist was discovered (Figure 6).

At this point, the motivation that stemmed from gaining new knowledge and examining the possibilities became important. This entailed not being so concerned with ambitiously creating fancy interactive images (they could wait). This new mode of learning was fun. It was a decisive moment in the learning process. This resulted in several patterns, exploring different possibilities with one or two variables in the *Processing* code.

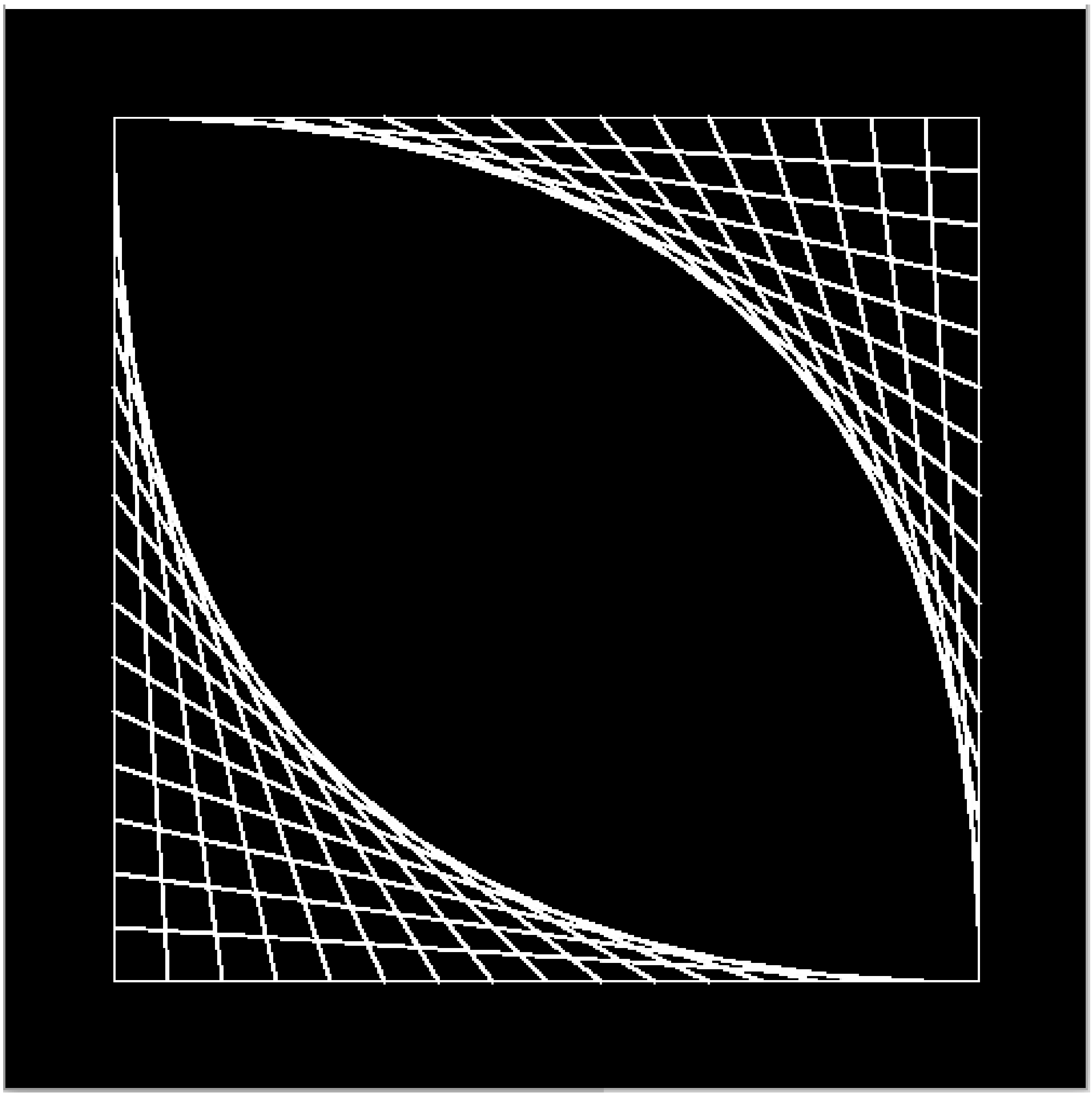


FIGURE 6. A grid with a twist, achieved after experimenting with the for structure.

After making patterns with one variable, the main author began testing out two variables. The next pattern (Figure 7) is made with two variables: It is a circle placed in a fixed position on the *x* axis. The first variable makes the circle copy itself along the *y* axis at a given rhythm. The other variable makes the circle occur in different sizes, starting at a given number and increasing until it reaches its size limitation (which is defined in the code).

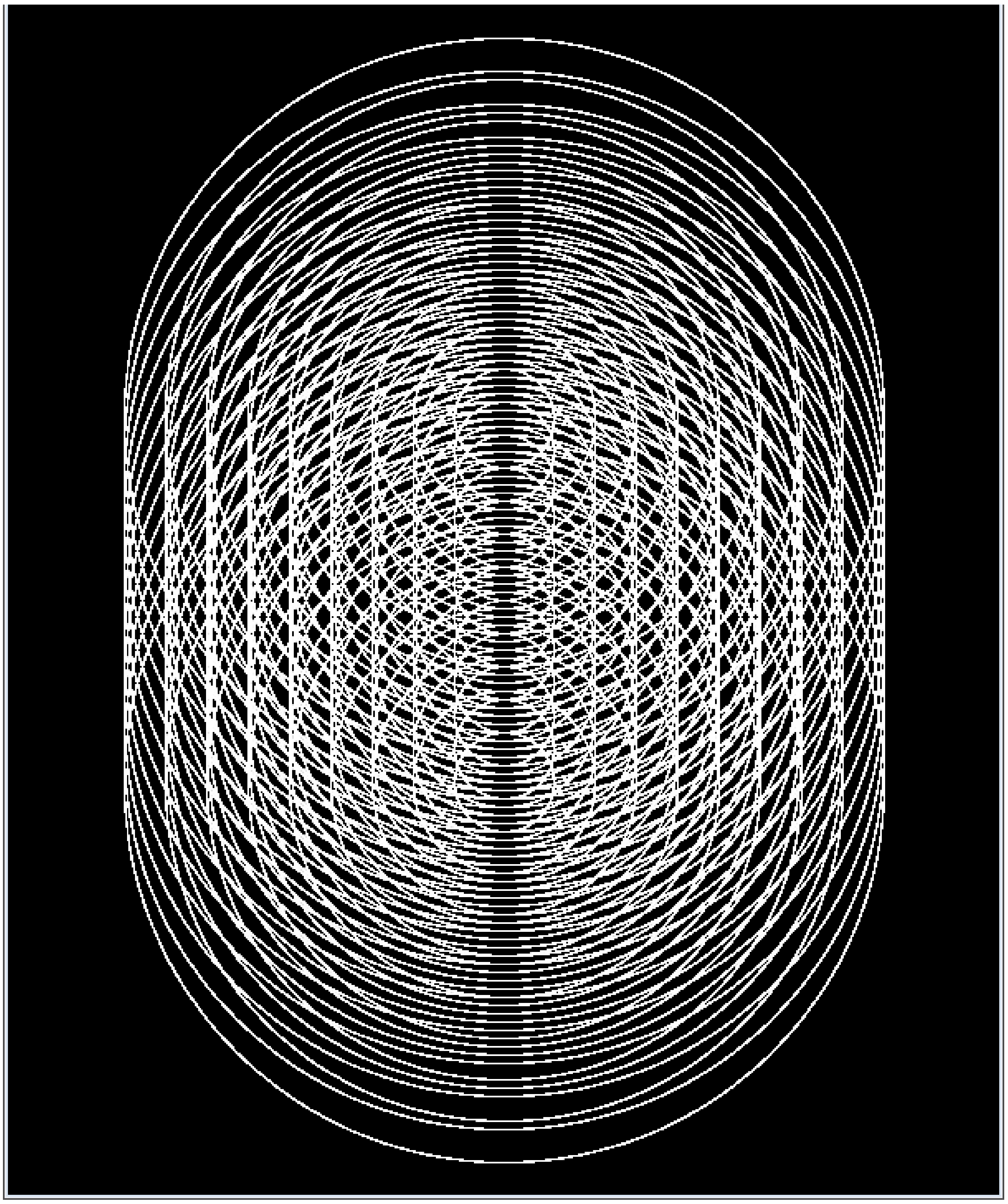


FIGURE 7. Testing two variables in the for structure: An ellipse duplicating its position and size.

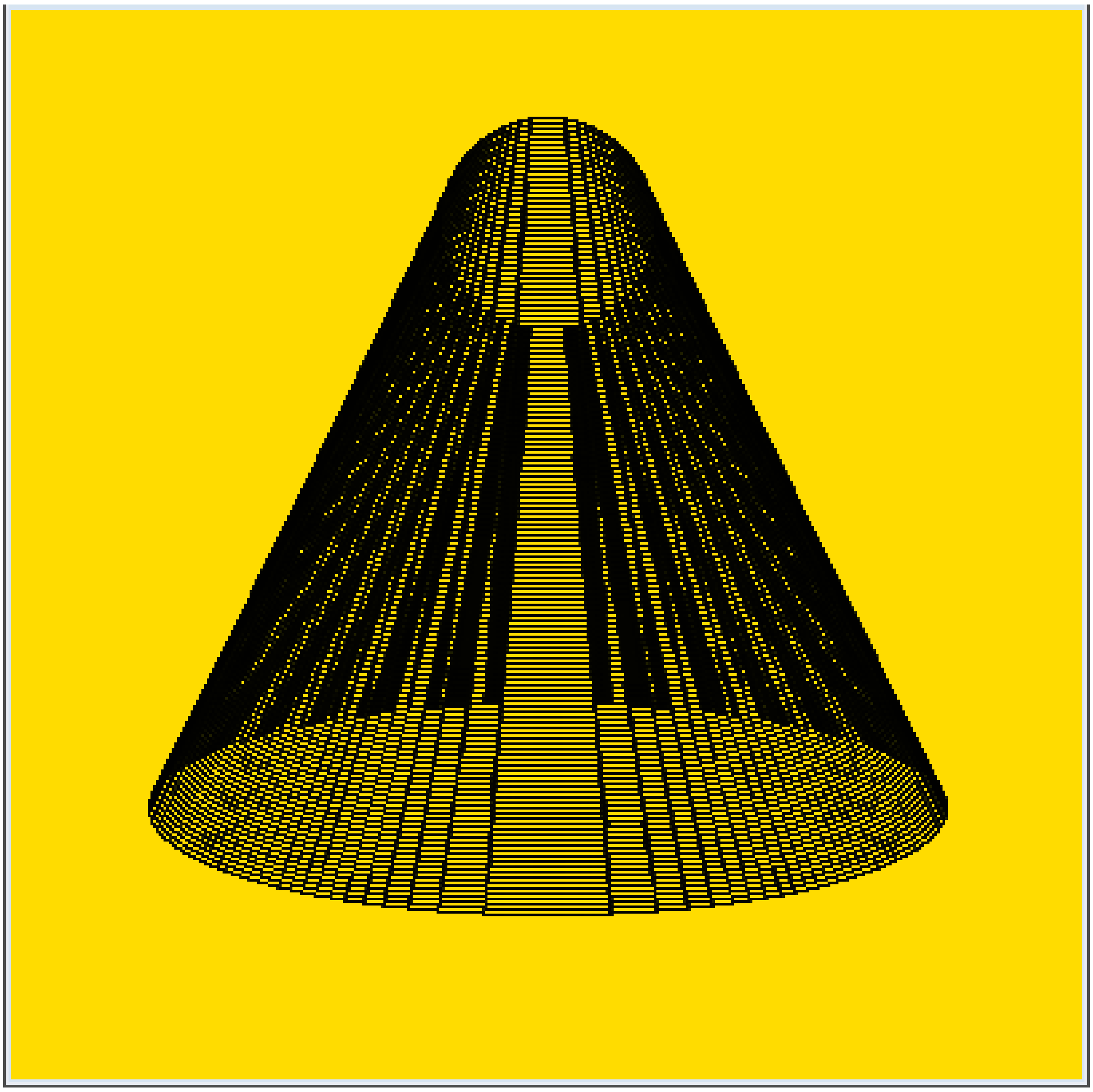


FIGURE 8. Based on the first cone created with the for command, this is another way to make the ellipses multiply both the x position and the y part of the shape. The yellowish background is there for the variation (the exact colour is RGB 255,220,0).

Figure 8 has two variables (*i* and *j*), both of them increasing at a specified rhythm. The cone is a result of a variable *y* position, increasing with a specified value until it stops at another specified value. The shape has a fixed *x* radial but the *y* radial increases gradually until it stops at a specified point.

Figure 9 resulted from going back and tinkering with the code from Figure 7, without necessarily predicting the exact outcome. The goal was to play with one's own code which was written from scratch, taking it further, working intuitively and seeing the visual results right away rather than trying to understand everything in advance. This was combined with the additive RGB colour mixing from stages one and two. Some of the meeting lines create new colours as a result of additive colour mixing (CMY and white).

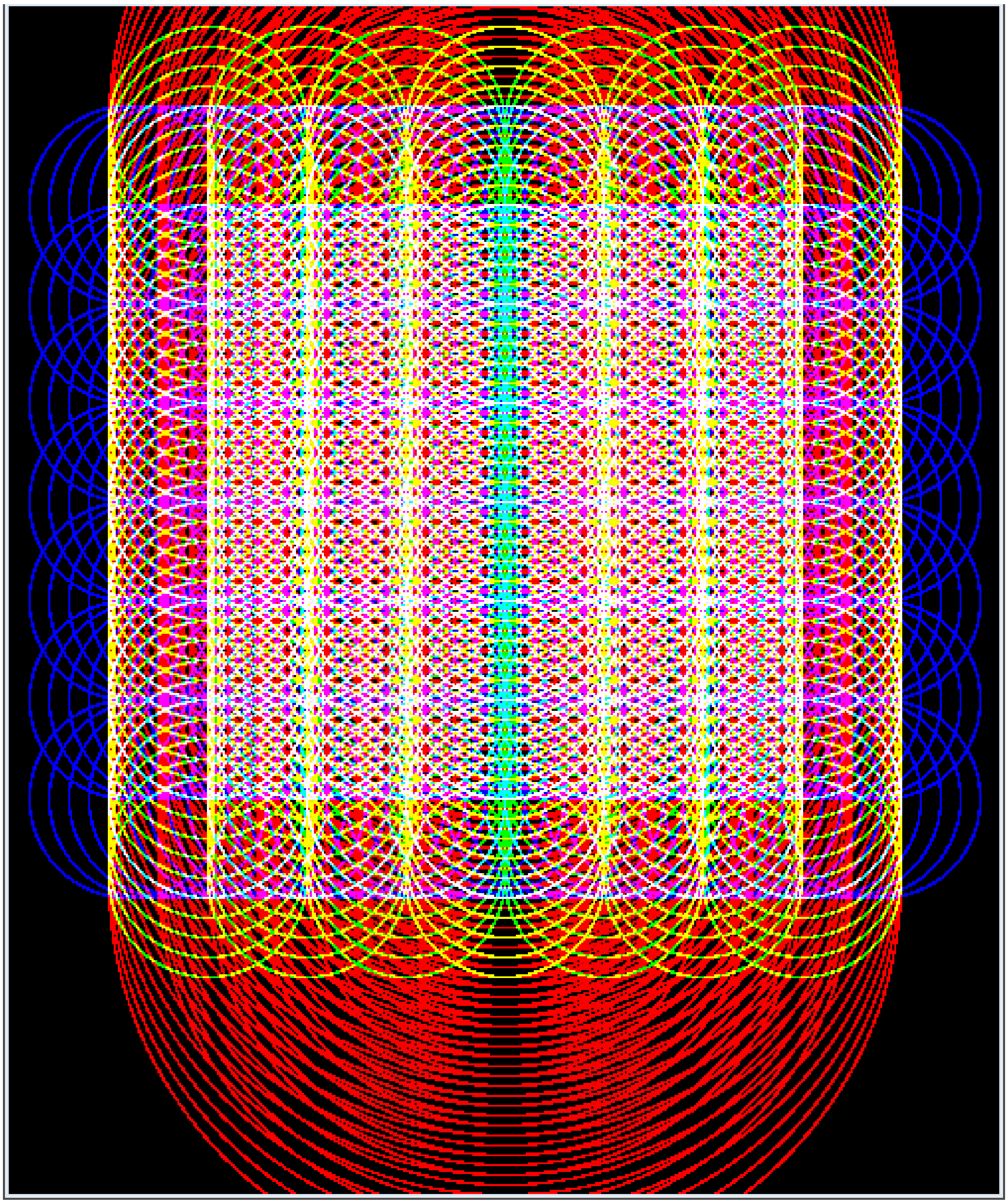


FIGURE 9. Result of playing with the codes in Figure 7, combining it with the colour-mixings from stages one and two.

By this part of stage three, the dialogue between the creative tinkering and the learning process became interwoven with new ideas that arose from the new findings. Practical exercises and reflections on actions (Schön, 1991) drove the working process forward. The *for* structure is a complex function to a beginner but it gradually became simple with practice, making it possible to create patterns. Within the third stage the focus had been to explore this one function, examining its possibilities through many different tests. They started out as simple patterns written from scratch, then both playful tinkering and structured analysis became involved in order to modify the patterns. An effective way to make new things happen in the patterns is to tinker with them without predicting the exact results but while still being able to understand the results after the changes occur on screen. In other words, the process contains a series of alterations between writing codes, seeing the results, taking them further with intuitive tinkering and then analysing the results later. Afterwards it is important to transfer the knowledge gained from the outcomes into new versions, creating new codes analytically. The learning process is not over: It would be ideal to deeply explore another function, advancing further by conducting a new stage.

DISCUSSION

This paper offers a perspective on programming within art, design and craft education with a special emphasis on tinkering as a learning process. The case reflected in this paper has shown that tinkering has been a crucial element throughout the learning process in terms of advancing the main author’s knowledge of programming. After using textbooks in stage one, tinkering with existing codes in stage two made it possible to freely play, unconcerned with the systematic learning of advanced programming. This boosted the motivation to enter stage three, developing new codes based on newly achieved knowledge. As a part of stage three, the patterns created were further developed through ‘happy accidents’ that stemmed from tinkering. These results were then analysed and refined into new patterns.

When first discovering *Processing* and upon hearing of its potential and opportunities, the main author had high ambitions in terms of what he could create with this software. It held a variety of new possibilities (compared to *Photoshop* or *Illustrator*) and it made it possible to combine digital imagery with interactivity: It may even be possible to learn how to combine images on a screen with a microcontroller kit with censors to create images that respond to physical stimuli. This project started with high ambitions and an even higher level of motivation. However, it also started with a lack of skills and knowledge in regard to programming. Despite this, after learning the basics, it became interesting to play and tinker with existing codes to investigate the software’s advanced possibilities. After a period of tweaking and tinkering with several code scripts that were found online, new visual effects were made but without necessarily consolidating how to write similar codes from scratch – motivation decreased as the high ambitions began to seem unreachable. Still, it was interesting to play with these codes, modifying them or creating errors that only left a blank screen.

As Berland and colleagues (2013) describes, tinkering is about messing around and meeting the material’s feedback mechanisms. *Processing* gives you instant feedback on whether the code works or not. Either you have created something visual or the program tells you what part of the code needs to be fixed. If you cannot fix it, you may delete the code line (or hide it by writing two dashes in front of the lines). After looking for interesting codes that could be modified through small systematic steps, through playing around until new things happened or until they failed, a part of these codes began to seem slightly familiar as it had been present in several of the projects: The *for* structure. After a period of reading about this structure, the learning process reached a new stage. It had made a small and yet important leap. Now it was possible to see more clearly how some of the more advanced code scripts were built and even how they could be written from scratch. The motivations of the main author increased slightly, and the ambitions were adjusted to a more achievable level. This does not mean the learning process is over; many more phases such as this are needed to achieve proficiency at *Processing*. But at this stage, creating images with code is great fun and because the main author is a teacher of digital media at university level, this enthusiasm may reach (and hopefully influence) his university students.

After working on the third stage, after denouncing the main author’s original high ambitions (or at least after putting them on hold), the learning process had to return to a phase that was similar to the very first steps of stage one; the online resources and textbooks (Reas & Fry, n.d., 2006). It was a step back to the basics. But stage two was necessary in order to get there; it was necessary to play with codes and to have fun with the higher ambitions before it felt natural to take this step backwards. After stage one, many of the structures and functions (such as the *for* structure) described in the textbooks seemed incomprehensible. Between stages one and two, looking for existing codes seemed more motivating.

This paper does not claim that computer coding should always be present within art, design and craft education, but it is an attempt to look at how and why it might be relevant to use programming to create digital images. There are many more possibilities that arise from using *Processing* and other such programming languages. This paper gives some examples of how programming images can create other kinds of outcomes than what you can achieve in known imaging software such as *Adobe Photoshop* or *Illustrator*. The learning process described in this paper may serve as an example of how to use programming to create images and how to learn computer coding step by step. However, further research on the topic is needed.

When programming becomes a part of the curricula in lower and upper secondary schools in Norway, creating images with computer coding might become increasingly relevant in art, design and craft subjects. Why should we learn this and at what level of difficulty should an art, design and craft teacher be teaching programming? They are not IT teachers and the students are not attending an IT class. Nevertheless, programming can be used for many different things and it opens up new possibilities in terms of creating visual digital images. You can generate patterns, iterate with small changes in code scripts or even make something interactive.

This paper has shed light on the learning process. Tinkering has proven to be a good way to learn because it combines a playful approach with the instant feedback that *Processing* provides from the materials. This may apply to different programming languages suited for different children at different ages. On the website, Lær kidsa koding (English: Teach the Kids How to Code) (Lær kidsa koding, n.d.), there are different online resources for *Scratch*, *Processing* and other such software, with step-by-step recipes spanning across varying difficulty levels, from kindergarten to upper secondary and beyond. Many of the instructions end with tasks where the users are encouraged to find out the next steps on their own and take the codes to new levels. A good thing about tinkering with codes is that it is not 'dangerous' to fail. Failing is a fruitful and often necessary part of the learning process. After the initial playful steps, one might have learned something new and become ready to write codes from scratch. Based on the experiences drawn from this project, tinkering is regarded as a fruitful way to introduce programming to both teachers and children in art, design and craft subjects. As the new curricula will be implemented in 2020, further research on this topic is crucial. This includes classroom studies at both primary, lower secondary and upper secondary schools, as well as teacher-training programs.

REFERENCES

Antonsen, R. (2018). Card Shuffling Visualizations. In *Proceedings of Bridges 2018: Mathematics, Art, Music, Architecture, Education, Culture* (pp. 451–454). http://archive.bridgesmathart.org/2018/bridges2018-451.html

Berland, M. (2016). *Making, tinkering and computational literacy*. In K. Peppler, E. R. Halverson, & Kafai, Y. B. (Eds.), *Makeology: Makers as learners* (Vol. 2, pp. 196–205). Routledge. https://doi.org/10.4324/9781315726496-12

Berland, M., Martin, T., Benton, T., Smith, C. P., & Davis, D. (2013). Using learning analytics to understand the learning pathways of novice programmers. *Journal of the Learning Sciences*, *22*(4), 564 – 599. https://doi.org/10.1080/10508406.2013.836655

Bocconi, S., Chioccariello, A. and Earp, J. (2018). *The Nordic approach to introducing Computational Thinking and programming in compulsory education.* (Report prepared for the Nordic@BETT2018 Steering Group). https://doi.org/10.17471/54007

diSessa, A. A. (2001). *Changing Minds : Computers, learning and literacy.* MIT Press. https://doi.org/10.7551/mitpress/1786.001.0001

European Schoolnet. (2017). *Country Report on ICT in Education – Norway*. http://www.eun.org/it/resources/country-reports

Kuehni, R. G. (2004). *Color: An introduction to practice and principles*. https://doi.org/10.1002/0471687448

Ludvigsen, S. (2014). *Elevenes læring i fremtidens skole – Et kunnskapsgrunnlag : Utredning fra et utvalg oppnevnt ved kongelig resolusjon 21. juni 2013.* Kunnskapsdepartementet.[Pupils' learning in the future school – A knowledge base : Report from the committee appointed by Royal Resolution on 21 June 2013]. https://www.regjeringen.no/no/dokumenter/NOU-2014-7/id766593/

Lær kidsa koding [Teach the Kids How to Code]. (n.d.). *Oppgaver* [Tasks]. https://oppgaver.kidsakoder.no

NOU 2015: 8. (2015). *The School of the Future — Renewal of subjects and competences.* Ministry of Education and Research.https://www.regjeringen.no/en/dokumenter/nou-2015-8/id2417001/

Reas, C., & Fry, B. (2006). *Processing: A programming handbook for visual designers and artists*. MIT Press.

Reas, C., & Fry, B. (2009). Processing: Programming for designers and artists. *Design Management Institute Review, 20*(1), 52–58. https://doi.org/10.1111/j.1948-7169.2009.tb00225.x

Reas, C., & Fry, B. (n.d.). *Processing*. https://processing.org

Regjeringen [The Government]. (2017). *Åpner for koding som valgfag ved alle skoler*. [*Coding as an elective in all schools*]. https://www.regjeringen.no/no/aktuelt/fra-hosten-kan-alle-ungdomskoler-tilby-programmering-som-valgfag-til-elevene/id2552808/

Regjeringen [The Government]. (2018). *Fornyer innholdet i skolen* [Press release]. [Renewing the content in schools]. https://www.regjeringen.no/no/aktuelt/fornyer-innholdet-i-skolen/id2606028/?expand=factbox2606073

Scardamalia, M., & Bereiter, C. (2010). A brief history of knowledge building. *Canadian Journal of Learning and Technology, 36*(1). https://doi.org/10.21432/T2859M

Schön, D. (1991). *The reflective practitioner: How professionals think in action.* Avebury. (Original work published 1983)

Sennett, R. (2008). *The craftsman*. Yale University Press.

Utdanningsdirektoratet. (2018). *Yrkesfaglige utdanningsprogram fra 2020* [Vocational education programs from 2020]. https://www.udir.no/laring-og-trivsel/lareplanverket/forsok-og-pagaende-arbeid/ny-tilbudsstruktur-og-nye-lareplaner-pa-yrkesfag/ny-tilbudsstruktur-i-fag--og-yrkesoplaringen/

[1] First published as Haakonsen, P & Fauske, L.B. (2019). Learning to create images with computer code. *Conference Proceedings of the Academy for Design Innovation Management, 2*(1). The article is republished with permission.