Draft 1, Thursday

**ABSTRACT:** TODO!

## 1. Introduction

A solid understanding of Data Structures and Algorithms (DSA) is an important skill for any student in the field of Computer Science, since they underpin many of the fundamentals of Computer Science. However, as is the case with many conceptually demanding topics, students can find learning DSA challenging early on in their studies[1] due to their inability to correlate DSA concepts with real-world objects and problems. Furthermore, the complexity of DSA means that some students may struggle to engage with the topic, thus requiring more motivation if they are to successfully learn it.

Educational tools have often been proposed that either take advantage of Game-based Learning (GBL) or Algorithm Visualisation (AV) in order to help students better learn concepts in computer science.

Game-based Learning is the use of games with traditional game elements (such as level progression or animation) in order to teach or practice a particular topic[2]. We find that GBL is been quite commonly used in the some fields of computer science, especially programming, however there are currently few examples of games that directly teach basic DSA. Those games that do exist to teach DSA usually revolve around more advanced DSA theory, such as algorithmic complexity. While learning those concepts are also important, there is a lack of games that help students learn about the practical application of DSA to computer science problems.

Another commonly studied educational tool is the concept of Algorithm Visualisation (AV). With AV, we try to abstract away the implementation of a given algorithm or data structure (in terms of, say, memory locations and pointers) and instead present them as static or dynamic diagrams or even analogies. Unfortunately while AVs are commonly seen in DSA teaching, most of these AVs are of the static, box-and-line diagram type[3]. When compared to dynamic visualisations that use analogies, static AVs may be less effective at helping students correlate DSA with problems and implementations.

We therefore propose an educational tool, DeCode, that integrates both GBL and AV in ways that aren't commonly seen in the field of DSA. By using both GBL and AV approaches, we hope to address both issues with learning DSA - the high complexity (with support from AV), and lack of student motivation (with support from GBL)

DeCode is a game-based learning tool for teaching introductory data structures such as Arrays, Lists, Stacks, and Queues alongside basic algorithms that use said data structures. DeCode takes advantage of game-based level design and a 2.5D, animated depiction of data structures. The animation uses the metaphor of cars and parking spaces to better engage the student and help map key DSA concepts to a real-world analogy.

1

The research, implementation process and evaluation of DeCode will be discussed in this report, along with conclusions and possible future work.

## 2.    Research Intent and Questions

Our research intent revolves around determining whether our implementation of Algorithm Visualisation with Game-based Learning in the same tool offers improved educational value for students new to learning DSA. We therefore want to address the following research questions;

1. Does DeCode help improve students understanding of DSA concepts?

2. Does DeCode offer additional educational value on top of existing teaching methods or tools (such as box-and-line visualisations of DSA)?

3. Is DeCode enjoyable and engaging for students to use (and thus students are more likely to remain motivated)?

## 3.    Literature Review

### 3.1. Algorithm Visualisation Techniques

One of the first techniques used to improve instruction of DSAs was Algorithm Visualisations. While these were originally limited to drawings on paper and blackboard, these eventually developed into software-generated interactive visualisations as early as 1984, with BALSA[4] as an early example of such. As development progressed, more user friendly visualisation tools were created, with most being freely distributed online as open source software, for use by educators around the world.

Despite this work however, the most common examples of AVs are still Static AVs (whether on a computer or paper), which do not allow the user to interact with the visualisation while using it, or allow analogies to be displayed. Recent research has shown that improved interactivity, explanations[5] and the ability for students to construct their own visualisations can help significantly improve the usefulness of AVs to students, especially those that struggle with traditional learning methods[6].

### 3.2. Existing AV Work

One of the most prominent algorithm visualisation tools available now is VisuAlgo.net[7], developed at the National University of Singapore. It offers a clean, modern, 2D visualisation of various data structures, and full interactivity for the user to perform operations on the data structure, and see what happens. The disadvantage of this approach is that the visualisations don't naturally lend to the construction of analogies.

While most AV tools we identified were designed purely for teaching and demonstration, the TRAKLA2[8] tool was developed to be used as a student assessment tool. Even though it uses similar, traditional representations of DSA as

VisuAlgo, TRAKLA adds the ability for students to view algorithm or DS questions. Students are then able to answer the questions by manipulating the visual representation, and have their answer marked automatically. Students are also able to view and browse through (i.e. step back and forward) the model solution for their question.

### 3.3. Existing GBL Work

Despite a large volume of existing work on both Game Based Learning and Algorithm Visualisation, there's a surprisingly low number of papers that implement both GBL and AV to teach DSA. One of the few papers we identified was Park and Ahmed[9] in which they design a video game using the Unity game engine that visualises various data structures. The game uses real world analogies in order to visualise these in a 3D environment, such as showing the stack data structure as a stack of crates. The player can perform operations on these data structures in order to complete game objectives. Vegh[10] presents another similar game that's centred around algorithms rather than data structures. The game uses a real word analogy of crates to demonstrate sorting algorithms (in this case, sort by mass). The game also logs the actions of players, allowing educators to investigate them and determine the student's line of thought.

### 3.4. Evaluating GBL

One of the most important steps of building an educational game is the evaluation - we need to know whether our game offers a positive user experience for students, while contributing to learning. It would also help if different educational games can be compared to one another to assess their relative effectiveness. Savi et al presents MEEGA[11][12], the Model for the Evaluation of Educational GAmes aimed to tackle this. MEEGA is designed to offer a common evaluation model for educational games that can be compared to each other. Its questionnaire model is derived from literature and is designed to measure three dimensions - motivation, user experience and learning. These encapsulate the dimensions of attention, relevance, confidence, satisfaction, immersion, social interaction, challenge, fun, competence and short/long term learning.

### 3.5. Other Education Research

Most tools that teach basic DSA usually don't cover some intricacies of working with them, which can result in students being confused on concepts that might otherwise be regarded as basic. Izu, et al.[13] presents an example of students struggling with array manipulation and loop iterations where students are unable to recognise which direction they should loop in, in order to shift an array to the left or right. They conclude that this is the result of an unintentional oversight in course materials favouring upward loops.

Meanwhile, there has also been research into correlations between students ability to explain and trace code and their abilities to write code. Teague and Corney's work[14] presents a study of such a relationship where the ability of students to explain a simple code block that swaps two elements is correlated with their ability to explain a simple sorting algorithm

that relies on swapping out-of-order elements (similar to BubbleSort).

## 4. Goals

In general, we had three broad goals during the development of DeCode;

1. Teach introductory DSA concepts

2. Be an engaging tool for the user

3. Offer a highly visually-appealing visualisation

Additionally, from our literature review, we were also able to identify ideas from multiple pieces of existing work that we would like to integrate into DeCode.

- An *interactive* algorithm visualisation with explanations, since they've been shown to be more useful to students[5].

- Ask students DSA questions, and allow them to answer them by manipulating the visualisation, such as in TRAKLA2[8].

- Use a game environment, with game objects as real-world analogies to better demonstrate DSes, such as in Park and Ahmed[9].

- Log students interactions with the tool, to allow educators to better understand student's line of thought, similar to Vegh[10].

- Explicitly introduce loop directions and array shifting, since Izu, et al[13] showed that students are commonly confused by this.

- Measure students abilities to explain and trace code instead of just their performance in performing DS operations, in order to see if we can reproduce the relationships between the two that Teague and Corney demonstrated in their work[14].

- Evaluate using MEEGA[12], so that we're using a common evaluation model for our game, in order to make it easier to compare our results with other results.

## 5. Analogy/Metaphor

Using an analogy in our visualisation can allow us to make DeCode more visually appealing, while also improving educational value[9][5]. However, coming up with an analogy that works for all of our use cases proved to be quite difficult - if something cannot be adequately explained using your analogy, then it quickly falls apart. The analogy should also be relatable for almost all users of the application and not be overly complex. Eventually, we settled on the idea of cars and carparks as our analogy.
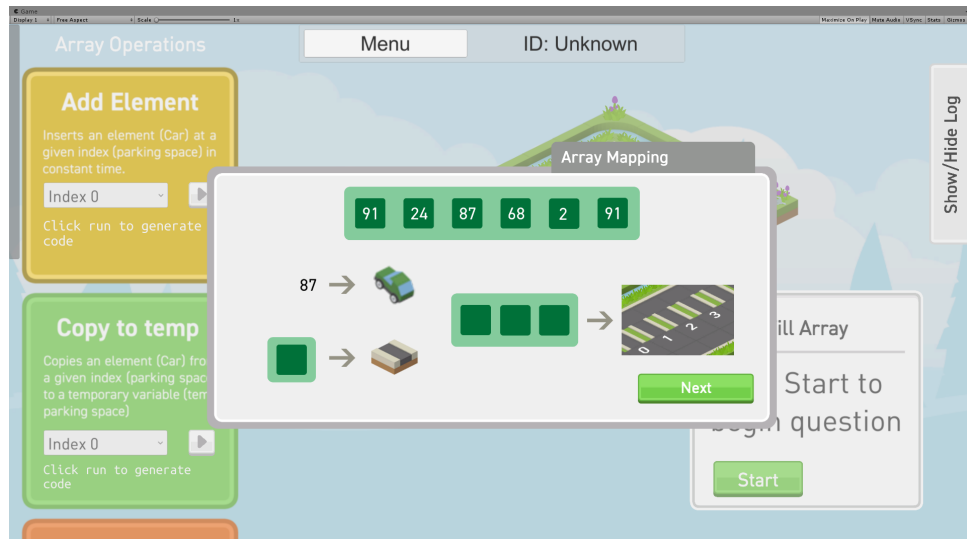
4

Figure 1: Help screen explaining our analogy

Cars would represent the values that we would want to store in our data structure, with different types/colours of cars representing different data values that we would like to store. Parking spaces represent locations in memory where these values can be stored, and contiguous blocks of memory (such as arrays) are represented as carparks. To store a value into a memory location, we use the analogy of parking a car into a parking space. The parking spaces are labelled with their index where appropriate.

To help users get better accustomed to our analogy, we show a help screen (Figure 1)at the start that links the analogy to a more classic box-and-line diagram visualisation of data structures.

## 5.1. Limitations

One of our primary goals with this analogy was to have minimal complexity, which meant keeping it to only two main elements (cars and parks). Unfortunately, this meant that there were a few things that ended up being inadequately explained by our analogy.

Firstly, the user is able to view the contents of the whole carpark at a time, whereas in reality it takes time to check the contents of each memory location. This made some algorithms such as sorting algorithms more awkward to explain since in reality comparing various indexes makes up a significant portion of the running time of such algorithms. Initially we intended to add some sort of character to the game (such as a driver) to represent the current state of execution - so, as an example, the driver would walk around the carpark and look at cars in order determine their value, then get in one and drive it when array elements are being moved/copied. However, we found that this overly complicated the visualisation, and actually made it harder for users to keep track of what's going on. We therefore decided to go without this, even if it did mean our analogy was less accurate.

Figure 2: 4 seasons for the 4 levels in DeCode

The second issue was trying to display pointers with our analogy - we simply couldn't come up with something that would fit well that represented a pointer without being overly complex. Signs and other things pointing to a specific parking spot (memory location) would be too complex for students to grasp. Ultimately, we decided to omit most data structures that use pointers, such as Linked Lists since we would be unable to represent them satisfactorily. The only DS that had a pointer was our optimized implementation of Queue, for which we used a red flag to mark the parking space pointed to by a pointer. This only worked since we only had one pointer (having multiple, different coloured flags would be confusing for something like a Linked List).

The final issue that we identified was how we would deal with copying of values in the data structure. For example, copying the value from index 0 to index 1 in an array would mean *duplicating* the car that stores the value in index 0 and driving it to index 1, while leaving the original in index 0. This breaks the analogy's link to the real world since one cannot simply duplicate a car in reality. In the end we kept this in the game since we were unable to come up with a way to represent this more realistically. Even if this one part isn't realistic, we feel students will still be able to understand the intent of the analogy.

## 6.    Design and Technology

### 6.1. Design

One of the main goals of DeCode was to design a highly visually appealing visualisation, in order to draw users in and differentiate our visualisation from other, more standard, 2D visualisations. One of the best ways to achieve a high quality visualisation would be to use custom 3D models for our analogies, however we this would've been impractical given the

Figure 3: Left: Sample assets from Kenney's pack, Right: Our main menu UI

time constraints we were under. Instead, we elected to use existing city assets from Kenney's Isometric City Pack[15]. This asset pack offered most of the assets that we needed for our analogy (city buildings, streets, parking spaces and cars) with a consistent design across them. The assets that weren't available (such as labelled trucks for sorting and indexed parking spaces) were created manually by us, with care taken to ensure they match the design of the Kenney pack.

To visually separate each of the 4 data structures we were introducing, each data structure has its own seasonal themed variation on the normal assets. Figure 2 shows the design of DeCode across all four seasons (data structures).

UI design was also done by us, with the goal of trying to match the design of the Kenney pack as much as possible. A comparison can be see in Figure 3. Another primary consideration in our UI design was to ensure all UI elements would be usable on a touch device, should a user choose to test our game on such a device. This meant things like array index selection would need to be designed as drop down boxes instead of text boxes, since popping up a keyboard results in a poor touch experience. We also made sure that all buttons were of sufficient size for touch interaction.

### 6.2. Technology

Due to time and cost constraints when developing DeCode, achieving the highest possible technical quality was not a priority in our game. Instead, our choice of technology was largely determined by existing familiarity and cost. With that in mind, we chose to use the Unity game engine[16] combined with the Ultimate Isometric Toolkit[17] library - this offered us most of the tools we needed to design our game, with strong existing documentation and 2.5D game support. We were also already familiar with Unity, so we needed less time to get up to speed with the tooling. Unity also supports WebGL deployment, which means DeCode can run in-browser without users having to download anything. Additionally, both of these tools are free (Unity, for non-commercial use) or low cost (Ultimate Isometric Toolkit).

## 7. Data Structures

We chose 4 data structures to cover in our tool based on their importance in the ACM CS2013 curriculum's section on Fundamental Data Structures[18] and our ability to visualise it using our analogy. As mentioned earlier, we opted to skip

Figure 4: Cars in the middle of being copied to a larger internal array

data structures that require significant use of pointers since we are unable to represent them using our analogy.

## 7.1. Arrays

Arrays are depicted as a simple carpark of a fixed size, with an extra park representing a temporary variable. Users are asked to fill the empty array and swap values around - this introduces the user to most of the array operations, including add, copying to/from temp and copying from one index to another. Following on, the user is asked to rotate the array using the operations they were just introduced to. We chose to do this since prior work[13] has shown that students are commonly confused by these operations. Finally, we introduce the user to BubbleSort - a common simple sorting algorithm that puts to use all of the operations that they have learned thus far. BubbleSort is introduced in two steps - firstly, users are given an animated demonstration of the algorithm completely with explanations, after which they are asked to step through each step of the algorithm by manipulating the visualisation. Throughout this, the user is asked basic multi-choice questions about what they just did, except in a psuedocode form. This encourages the user to link their knowledge in the visualisation with code.

## 7.2. ArrayLists

ArrayLists (also known as Vectors in some languages) are introduced in two parts - first, we expose the user to the benefits of a List - namely, dynamic sizing that allows us to add as many elements as we see fit. The user is asked to add elements to the ArrayList, and the internal array is shown as expanding every time a new element is added. This is visualised by having multiple rows of parking spaces next to each other (see Figure 4), with each row 1 larger than the last. When a new element is added that exceeds the size of the current row, all the cars are moved to the next one, thus representing a possible naïve implementation of ArrayList. Throughout this process, the user is asked questions on how the performance of this implementation could be improved, in order to get them thinking about the next part.

Figure 5: Annotated view of the gameplay features

In the second part, we introduce a more optimized implementation of ArrayList to the user, in order to give them an idea of how ArrayLists are used in the real world. This done using the same array expanding visualisation as before, except we now show how the implementation doubles the size of the internal array each time.

### 7.3. Queues

Since Queues are an abstract data type (ADT), we elected to show it in two parts - first, we show the user the interface of such an ADT, where the implementation is a "black box" and the user can only perform enqueue and dequeue operations on said box. We visualise this as a parking garage with one entrance (for enqueues) and one exit (for dequeues). After the user is familiar with these operations, we allow the user to "see inside" the parking garage, thus revealing the implementation to them. Similar to ArrayLists, we first introduce them to a naïve implementation of a Queue (basic array), then proceed to introduce a circular array implementation, where we use a pointer (red flag) to arbitrarily define the start of the array.

### 7.4. Stacks

Stacks, being another ADT, are also introduced in a similar way to Queues - first a "black box" view where the user can only use the single entrance for push/pop, and then later a "see inside" view where the array-based implementation is revealed to the player.

## 8. Gameplay Features / User Interface

The majority of the game has a interface similar to that shown in Figure 5. The annotated elements do the following...

1. Shows the user their current game ID, used to track their progress for the evaluation (more in the Evaluation section).
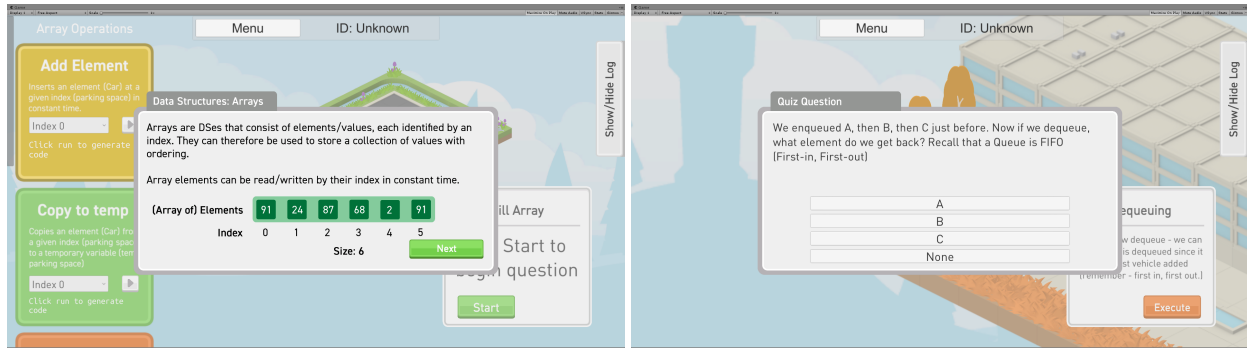
Figure 6: The help and quiz pop-up screens

2. The code view allows the user to see a pseudocode representation of the operations that they have done, so that they can better relate the visualisation with code.

3. Tasks for the user are introduced here. Users are able to *check* if their answer is correct, or *reset* in order to start again.

4. Operations supported by the current data structure are shown here, and users can select the index(es) to perform the given operation on. Generally, not all operations are shown at once since they are gradually introduced to the user

5. The visualisation is shown in the centre of the screen. When the task is introducing something to the user, the visualisation automatically zooms in to focus on what's being shown.

In addition to these elements, DeCode also shows help screens that give users a general introduction to a given data structure, and has quiz screens to ask users specific multi-choice questions, both of which are shown in Figure 6.

## 9. Evaluation

### 9.1. Design

In evaluating DeCode, we aimed to answer two main questions, which are related to our research intent - do students enjoy using DeCode, and, if so, is it actually educational?. In order to answer the first question, we rely on a questionnaire answered by students since they have a accurate self-judgement on whether they enjoyed the game or not. For the second question, we rely on a combination of questionnaire answers and in-game data collection, since it's been identified that students have a limited ability to self-assess their learning[19].

#### 9.1.1.   In-Game

As students play through our game, data is collected about how long they take to play through each level, what operations they use to perform each task, and how many attempts they take to answer the various multi-choice questions that are displayed. While these statistics are useful in allowing us to know how students are thinking about questions (by looking

| Question | Strongly Agree | Some-what Agree | Nei-ther | Some-what Disagree | Strongly Disagree |
|---|---|---|---|---|---|
| The app was effective in communicating Data Structure concepts | 53% | 27% | 16% | 0% | 3% |
| I found visualization to be a better approach to learning Data Structures compared to a purely code-based approach | 53% | 17% | 17% | 3% | 10% |
| Overall, I enjoyed using DeCode as a learning tool | 47% | 30% | 10% | 10% | 3% |

Table 1: Questionnaire questions around enjoyment

at their order of operations) or what questions they're struggling on, they don't directly answer our questions about whether DeCode actually offers educational value.

In order to help us determine whether students have actually learned anything, we added additional questions into the game where students are asked about similar concepts before and after the game introduces the concept to them. As an example, we asked students to identify the purpose of a piece of swapping code (derived from Teague's work[14]) before we introduce the concept to them in the game. We then ask the same question again, except with slightly different naming, afterwards. We intend for this to show possible improvement in knowledge from the student being introduced to the concept in-game. We were unfortunately unable to ask too many of these questions, as we feared it would make the game feel like a test. These questions do, however, introduce a few threats to validity, which we will discuss later.

*9.1.2.   Questionnaire*

The primary objective of the questionnaire is to collect data on user's experiences using DeCode, and specifically whether it improves students' motivation by being enjoyable and engaging to use. The original intention was to design our questionnaire around MEEGA[11][12] (more in Section 3.4), which is designed to measure the user experience and educational value of educational games. However when designing our questionnaire we found that MEEGA (and its sequel, MEEGA2) would take too long for students to answer, and thus result in us getting a reduced response rate to our questionnaire. This is because MEEGA is designed for studies where students must complete the study as part of their course, whereas our study involves students volunteering their own time to participate.

In the end, our questionnaire was based around a subset of MEEGA, with most of the questions pertaining to how the students felt about the user experience, and some questions about specific features in the game to gauge their individual usefulness (e.g. the code view or the MCQs).

**9.2. Results and Discussion**

With human ethics and HoD approval, we were able to test DeCode across two different cohorts of students - second-year Mechatronics and Computer Systems engineering students who haven't been formally taught DSA yet, and Software engineering students who had been formally taught DSA in the previous semester. Students were invited to try DeCode

and complete the questionnaire in their own time, in exchange for a chance to enter a raffle for a small prize.

In total, approximately 300 students were offered the opportunity to try DeCode. In the end, approximately 70 attempts were made at DeCode, of which 50 completed at least a few questions/tasks. 30 students answered our questionnaire.

### 9.2.1. Do students enjoy using DeCode?

In the free-form feedback in our questionnaire, students often noted that they enjoyed the game, especially the visual design/graphics. An example of some of the feedback we received;

- "The graphics were very cute. The quizzes were helpful too."

- "I enjoyed the visual analogy of cars in a car park. The graphics were very cute and I liked how it was like an old school game."

- "I enjoyed the visualisation of many of the ideas I have already been taught, it clarified how things actually worked (eg. with pop and push)"

In fact, of the 20 free-form comments we received, 16 of them noted that they enjoyed the game.

When it came to the likert scale questions, we had 3 that were relevant to measuring students' enjoyment of DeCode, which are shown in Table 1. Notably, we 77% of students found that they enjoyed using DeCode as a learning tool. In general, we don't have any real reason to doubt these students' evaluation of their own enjoyment, so it's pretty safe to make the conclusion that, Yes, students enjoy using DeCode.

We were able to identify two possible minor threats to validity for this conclusion: Firstly, our sample size of 30 students is relatively small ( 10% of total number of students the study was offered to), even if 30 is often claimed to be the magic cutoff for a statistically significant sample. The second potential threat to validity is selection bias - since students made their own decision as to whether to try DeCode or not, it's possible that the students who chose to take part in our study are more interested in educational games in general, thus giving us higher evaluation scores compared to had we randomly picked students from the cohort (more on this in Section 9.3.4).

### 9.2.2. Is DeCode educational?

The likert scale questions related to educational value are shown in Table 2. From this, we can see 77% of students answered positively to the question of "The app helped improve my understanding of Data Structure concepts", and 70% to the question of "I found visualization to be a better approach to learning Data Structures compared to a purely code-based approach". Interestingly, if we restrict ourselves to just looking at students who self-reported as having a very limited understanding of Data Structures, we find that these numbers go up to 83% and 84% respectively. This is

| Question | Strongly Agree | Some- what Agree | Nei- ther | Some- what Disagree | Strongly Disagree |
|---|---|---|---|---|---|
| The app helped improve my understanding of Data Structure concepts | 30% (33%) | 47% (50%) | 7% (17%) | 10% (0%) | 3% (0%) |
| Analogies used in the app helped me better understand the operation/usage of Data Structures | 50% (50%) | 20% (33%) | 20% (17%) | 7% (0%) | 3% (0%) |
| The content covered by the app helped me in studying for the course | 13% (20%) | 10% (20%) | 40% (60%) | 10% (0%) | 7% (0%) |
| I found visualization to be a better approach to learning Data Structures compared to a purely code-based approach | 53% (67%) | 17% (17%) | 17% (17%) | 3% (0%) | 10% (0%) |

*Note: Values in brackets are from students who self-reported as having a very limited understanding of Data Structures.*

Table 2: Questionnaire questions around educational value

| Question | Student Improved | Student Worsened | No Change or Perfect |
|---|---|---|---|
| Identifying swapped elements in pseudocode before and after playing an array swapping level | 6 | 5 | 20 |
| Identify possible performance improvements in an ArrayList implementation, after playing the ArrayList level | 7 | 5 | 3 |
| Identify possible performance improvements in a Stack implementation, after being shown a visualisation of the Stack implementation | 4 | 2 | 10 |

*"Student improved" is defined as answered question with less attempts than when the similar question was asked before the level, "Student worsened" is the opposite*

Table 3: Data collected from in-game MCQs before and after certain levels

promising, since that is the group of students that we're trying to target with DeCode self-assess DeCode as useful for their learning.

Additionally we received quite a few pieces of free-form feedback from students that praised DeCode for teaching them something new. An example of some of this feedback;

- "From my learning of algorithm, it was a bit hard to get my head around the implementation of a data structure. So I particularly like the explanation of the array implementation of the queue. It is very easy to understand."

- "I found list most effective(ly taught)! probably because it was the one I was least confident in my knowledge of, and I feel like the visual solidified my understanding of the concept."

- "I personally found that how queues were taught was very good. I had never heard of LIFO or FIFO but it was very handy."

A common criticism of educational tool evaluations, however, is that students are unable to accurately self-assess their learning(TODO: Citation Needed), especially when they are comparing different learning methods. Therefore, we also investigated data collected in-game to try and determine whether students have improved. As mentioned before, questions were asked before and after certain levels in an attempt to gauge whether students improved after playing the level, the

results for which are shown in Table 3. Unfortunately this data is very inconclusive - while it technically shows that more students improved than worsened after playing the game, the delta is too small to make any meaningful conclusion.

We assume the reason for this inconclusive data is because our pre and post questions were not similar enough, so other variations in students' abilities could affect their accuracy. Questions were not made too similar to avoid students rote-learning the answer, since they were presented to the student in quick succession.

The likert scale data collected here suffers from a similar threat to validity as defined above - namely the small sample size and a possible selection bias towards students who have a higher interest in educational games.

### 9.3. Other findings

In addition to answering our evaluation questions, the data collected in our questionnaire and in-game statistics were useful in making a few other observations. . .

#### 9.3.1. Common complaints

In our questionnaire, we collected free-form comments from students, some of which included suggestions for improvements or complaints. Here we collate some commonly shared complaints;

- Instructions in the beginning are not clear, especially around how to use the DS operations

- More competent students found the animations to be too slow, since they already knew what was happening

- Levels should be ordered from shortest to longest, instead of vice-versa

#### 9.3.2. Feature Usefulness

Our questionnaire also asked questions about the usefulness of various features in DeCode. We found that 70% responded positively in regards to usefulness about analogies, 50% about the code view, 60% about the user interface and 77% about the (MCQ) quiz questions. The feedback about the code view and quiz questions was surprising, since we didn't expect it to be so low and high respectively.

Our best guess is that students didn't discover the code view since it was only visible as a tab on the side of the screen, and there was no motivation to open/use it. One could finish the entire game without using it.

Quiz questions were a late addition to the game in order to collect more data for research purposes. They weren't explicitly designed with educational value in mind. We feel students likely found quiz questions to be helpful since they offered a different perspective when compared to the visualisation-based tasks in the rest of the game. They could've also been more interesting for more advanced students, since the questions tended to cover wider topics (such as performance and advanced implementation details) compared to the visualisation tasks (with that said, we are unable to prove this relationship with the data we have about self-reported ability level). It may be worth adding more questions in a future

version of DeCode, as this version included a limited number of questions since we feared turning the game into more of a test.

We were pleased to see that students generally responded positively about the analogies and user interface, since they were a primary focus during the design and development of DeCode.

### 9.3.3. *Array Rotation*

As mentioned in our goals, we have a task specifically on array shifting, since Izu, et al[13]'s work shows students would commonly rotate the wrong way. This consisted 3 parts - a task in the arrays level where students were asked to rotate an array to the right by one element, plus a pre and post multi-choice question (MCQ) on basic array manipulation (identifying what a given piece of code does - in this case, swapping two elements).

Looking at the task, out of the 30 students who attempted the level, 26 rotated right by one element (the "correct" way), whereas 4 rotated to the left (the "wrong" way). Of those who rotated to the right, about half decided to start at index 0 and half started at index 4. Of the students who rotated to the right, two students required more than 2 attempts to complete the task ($< 10\%$), whereas 3 of the 4 students who rotated left required more than two attempts, with one requiring 4 attempts.

As for the questions, the pre-question had an average number of attempts of 1.32, whereas the post-question was 1.36. The average number of attempts for the four students who rotated left were 2.0 and 1.5 for pre and post respectively. While it was interesting to see that the students that used the wrong rotation in the task also struggled with the questions, this sample size is too small to draw any conclusions.

A final observation is that 46 students started the rotation task, but only 30 completed it - the highest quit rate of any task. Unfortunately we don't really know why these students quit (only two of the 16 students who quit completed the questionnaire, and they didn't give any clue there), but it would be reasonable to assume that they found the task too difficult. It's also interesting to note that, despite quitting, most of these students were able to answer the pre question correctly.

While we simply don't have enough samples to draw any conclusions that back up Izu, et al.'s claims, it was nevertheless interesting to see that students appeared to struggle on a section we didn't expect them to.

### 9.3.4. *Study Cohort*

Of the 30 students who answered the questionnaire, 17 were from the Mechatronics and Computer Systems course and 13 were from the Software Engineering course. Of the Software Engineering students, 10/13 of them stated they already understood how some Data Structures work, and have implemented them in the past. Curiously, 9/17 of the Mechatronics and Computer Systems students gave that answer too. This contradicts what we were expecting, since, at least as part of

their University education, they shouldn't have been introduced to data structures or implemented data structures yet.

This seems to back our concern that students who particpated in our study tended to have existing interests/knowledge of data structures. This is a threat to validity as it's likely to be not representative of the cohort of all students, and also they are not our target group. We were unable to exclude these students since if we did, there would only be 11 students left. This is unfortunately a limitation of our study design since it was impractical for us to force students to complete this.

## 10.    Future Work

### 10.1. Game

While our evaluation didn't identify any pressing issues that needed to be resolved with DeCode, it did identify a variety of smaller issues such as lack of instruction and slow animations. Fixing these issues would help refine DeCode into a more polished game, and would significantly improve the user experience.

Another major upgrade would be to add new levels/data structures to DeCode to cover all of the Fundamental Data Structures outlined in CS2013[18]. This may be quite difficult as it would require overcoming some limitations in our current analogy (more in Section 5.1)

### 10.2. Evaluation

Due to time and resource constraints, we were regrettably unable to perform an evaluation on the scale that we would've liked. In the future, it would be good to perform an evaluation with a larger sample size (perhaps an entire class studying DSA), and collect better, more concrete, data on whether students learning outcomes are improved by using DeCode. This could be done in a few ways, such as implementing a pre and post-test, or analysing the pass rates of students who did and did not use DeCode, which was done by Mathrani et al[20] in a similar study. A full MEEGA[11] evaluation could also be introduced in order to allow for easier comparisons to be made between DeCode and other educational tools that have been evaluated with MEEGA.

## 11.    Conclusion

//TODO

## References

[1] B. Pérez-Sánchez and P. Morais, "Learning data structures—same difficulties in different countries?" *IEEE Revista Iberoamericana de Tecnologias del Aprendizaje*, vol. 11, no. 4, pp. 242–247, Nov 2016.

[2] K. Kuk, D. Jovanovic, D. Jokanovic, P. Spalevic, M. Caric, and S. Panic, "Using a game-based learning model as a new teaching strategy for computer engineering," *Turkish Journal of Electrical Engineering and Computer Sciences*, vol. 20, pp. 1312–1331, 12 2012.

[3] M. Esponda-Argüero, "Techniques for visualizing data structures in algorithmic animations," *Information Visualization*, vol. 9, no. 1, pp. 31–46, Mar. 2010. [Online]. Available: http://dx.doi.org.ezproxy.auckland.ac.nz/10.1057/ivs.2008.26

[4] M. H. Brown and R. Sedgewick, "A system for algorithm animation," *SIGGRAPH Comput. Graph.*, vol. 18, no. 3, pp. 177–186, Jan. 1984. [Online]. Available: http://doi.acm.org/10.1145/964965.808596

[5] L. Végh and V. Stoffova, "Algorithm animations for teaching and learning the main ideas of basic sortings," *Informatics in Education*, vol. 16, pp. 121–140, 01 2017.

[6] J. Stasko, A. Badre, and C. Lewis, "Do algorithm animations assist learning?: An empirical study and analysis," in *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*, ser. CHI '93. New York, NY, USA: ACM, 1993, pp. 61–66. [Online]. Available: http://doi.acm.org/10.1145/169059.169078

[7] "Visualgo.net." [Online]. Available: https://visualgo.net/en

[8] L. Malmi, V. Karavirta, A. Korhonen, J. Nikander, O. Seppälä, and P. Silvasti, "Visual algorithm simulation exercise system with automatic assessment: Trakla2," *Informatics in Education*, vol. 3, pp. 267–288, 10 2004.

[9] B. Park and D. T. Ahmed, "Abstracting learning methods for stack and queue data structures in video games," in *2017 International Conference on Computational Science and Computational Intelligence (CSCI)*, Dec 2017, pp. 1051–1054.

[10] L. Vegh, "Using interactive game-based animations for teaching and learning sorting algorithms," 04 2016, pp. 565–570.

[11] R. Savi, C. G. v. Wangenheim, and A. F. Borgatto, "A model for the evaluation of educational games for teaching software engineering," in *2011 25th Brazilian Symposium on Software Engineering*, Sep. 2011, pp. 194–203.

[12] G. Petri, C. Gresse von Wangenheim, and A. Borgatto, "Meega+: A method for the evaluation of educational games for computing education," 07 2018.

[13] C. Izu, C. Pope, and A. Weerasinghe, "Up or down?: An insight into programmer's acquisition of iteration skills," 02 2019, pp. 941–947.

[14] D. Teague, M. Corney, A. Ahadi, and R. Lister, "Swapping as the "hello world" of relational reasoning: Replications, reflections and extensions," in *Proceedings of the Fourteenth Australasian Computing Education Conference - Volume 123*, ser. ACE '12. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2012, pp. 87–94. [Online]. Available: http://dl.acm.org/citation.cfm?id=2483716.2483727

[15] [Online]. Available: https://www.kenney.nl/assets

[16] U. Technologies, "Unity." [Online]. Available: https://unity.com/

[17] ——, "Isometric toolkit." [Online]. Available: https://assetstore.unity.com/packages/tools/sprite-management/isometric-toolkit-33032

[18] A. f. C. M. A. Joint Task Force on Computing Curricula and I. C. Society, *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. New York, NY, USA: ACM, 2013, 999133.

[19] P. Eduardo Battistella, C. Gresse von Wangenheim, A. Von Wangenheim, and J. Martina, "Design and large-scale evaluation of educational games for teaching sorting algorithms," *Informatics in Education*, vol. 16, 11 2017.

[20] A. Mathrani, S. Christian, and A. Ponder-Sutton, "Playit: Game based learning approach for teaching programming concepts," *Educational Technology & Society*, vol. 19, pp. 5–17, 04 2016.