First Programming Language - Java or Snap? A Short Course Perspective

Mark Noone
Computer Science Department
Maynooth University
Maynooth, Ireland
mark.noone@mu.ie

Aidan Mooney
Computer Science Department
Maynooth University
Maynooth, Ireland
aidan.mooney@mu.ie

***Abstract* — A question often asked and rarely answered effectively in the Computer Science Education field is "What is the best First Programming Language"? We find ourselves asking this due in part to the low retention rates in third level introductory programming courses. With the ever-increasing requirements for Computer Science graduates in industry, and the introduction of programming courses in second level schools worldwide, now is the time to answer this question with confidence. If we can set younger students on the right educational path early on, we should see better performance at third level. This paper discusses the implementation of two identical introductory 8-week short courses, one based in Java and one based in Snap. These courses were taught to Transition Year students in Ireland and data was collected on how they performed and around their opinions of the languages. The goal was to determine if there is any significant difference in the difficulty to learn either course. If a difference is present, then there may be elements of the language itself causing difficulty given that the courses were identical. From the results of this phase of the study, we can make some initial recommendations about favorable First Programming Language choices.**

Keywords — Java, Snap, Computer Science, Education, First programming language, Visual programming language, Text-based programming language, Short course, curriculum.

I. INTRODUCTION

The role of the First Programming Language (FPL) is a very important one. In fact, Gupta [1] believes that the choice of FPL is such a big decision that it will have a "profound impact" on future learning in Computer Science. This paper will examine the development and first phase of teaching of two short courses; one in the Snap programming language and one in the Java programming language. These courses were created so as to be as equivalent as possible in content and different only in syntax and verbosity.

Previous research by Noone & Mooney [2] into a programming course in both Java and Snap has helped to inform this larger scale study. The study described in this paper is part of a larger project. The initial phases of this project, and the work undertaken thus far will be described in Section II.

The research questions for the overall project are to determine if there is a best approach to teaching an FPL, what the language choice should be and if a visual or textual approach provides the best results.

The courses developed are both eight-week courses and cover the following threshold concepts:

- Introduction to the Language
- Variables and Operators
- Selection
- Loops
- Strings and User Input
- Arrays (Omitted in this iteration of the course delivery)

Week seven and week eight respectively of the delivery of these courses allow time for revision of the material that was covered and the taking of a short examination question to test what was learned by the students.

Success in the courses will be determined using a number of metrics. A survey was administered during the first week of the course to collect some information relating to student opinions of programming. In the final week, a second survey was administered to determine how students found the course, if they enjoyed it etc. The examination question was also graded to determine student performance.

Finally, all of this information will be analysed with cross-comparisons between the Java results and Snap results to help draw some conclusions in relation to the research questions.

II. BACKGROUND

A. Systematic Literature Review

Noone & Mooney [3] first started examining this topic in 2016 with a Systematic Literature Review. This paper was based around finding literature to answer the questions "Are there any benefits of learning a visual programming language (VPL) over a traditional text-based programming language (TPL)?" and "Does the choice of FPL make a difference? What languages are the best ones to teach?". The main goal of the review was to determine the best languages (one visual and one textual) to create courses in.

John and Pat Hume Scholarship 2016, Maynooth University.

10th Annual International Conference on
Computer Science Education: Innovation and Technology (CSEIT 2019)

ISSN: 2251-2195
DOI: 10.5176/2251-2195_CSEIT19.148

In terms of VPL choice, the blocks-based approach seemed to resonate the best with students [4]. Within this category of VPL's, the most commonly used language was found to be Scratch with 40,951,154 projects shared when last checked in April 2019 [5].

An important aspect of these findings that we wanted to examine is the belief that VPL's are for younger students and TPL's are best for every other scenario. Cheung, Ngai, Chan & Lau [6] believe that a gap exists somewhere between the ages of 11 and 13; younger than that and a student will likely prefer a VPL while above that age TPL's may be preferred.

This coupled with the high usage rate of Scratch encouraged us to use Snap as our VPL of choice. Snap is a clone of Scratch that allows for the creation of custom code blocks. This means a course can be created that is identical to one in Scratch, while leaving room for future work in the area. Specifically, we aim to create a "hybrid" programming environment somewhere between Snap and Java. Weintrop [7] examined a form of this with promising results. This plan will be discussed more in Section VII.

On the TPL side, the TIOBE index [8] continues to show Java as the most widely used programming language in industry. According to Davies, Polack-Wahl & Anewalt [9], it is the most used programming language for CS1 courses in the USA. It also attained a high score (14/17) on Manilla & de Raadt's [10] educational language benchmarking test. For these reasons; along with our own familiarity of teaching it as a CS1; we chose Java as the TPL of choice.

Interestingly, it was shown by Noone & Mooney [3] that the actual choice of language isn't as important as one might think. It is the content that is key. That is why by developing these courses with equivalent content we can get a good idea of what issues the languages themselves may be posing.

B. Summer Camp Short Course

With the languages chosen, a pilot session was designed in 2017 by Noone & Mooney [2]. This pilot study involved the development and delivery of a 90-minute lesson in both Snap and Java. These lessons were delivered at a summer camp in our department for 10 – 18 year old students. Both lessons were delivered on the same day to the same cohort and were similar in content. They covered similar threshold concepts to those that were described for this curriculum (See Section I).

The goals of this phase of the research were to determine if the chosen languages and course delivery style were appropriate, and to gather some initial data on which course was deemed more "difficult" based on language choice alone. One of the major findings was that Java was considered significantly more difficult than Snap ($p = 6.8E\text{-}10$, $p<0.05$). It was also noteworthy that 41.6% of 13 – 15-year-old students preferred the visual approach while 62.5% of 16+ year-old students preferred the textual approach. This once again aligns with Cheung, Ngai, Chan & Lau's findings [6].

It is important to note that the findings of the Summer Camp study while noteworthy were performed on a small scale. The students were not expected to have any retained knowledge at the end of the sessions, it was merely about exposure to the concepts. The main purpose of the summer camp pilot study was to inform the body of work described in this paper and to aid in the creation of a large scale, multi-week curriculum in both languages.

III. Course Development

In May 2018, work began on developing the eight-week courses in both Snap and Java based on all findings from the summer camp short courses. For each session mentioned in Section I, the material was created in parallel for both languages to ensure that they were equivalent. The material created for each week included a class plan, teaching slides, in class practice questions, a homework sheet and a homework answer sheet. Two surveys were also drafted; one as a pre course survey which collected basic data and opinions on Computer Science and one as a post course survey which would collect opinions on the lessons, the courses and Computer Science in general again.

The questions asked both in class and as part of the homework sheets aligned closely with the normal CS1 material taught in our department but simplified for the target audience. A sample of a question asked in both courses during the "Selection" session is shown in *Fig. 1*.

> Create a program that checks if a person is able to vote. You will need to create an int that holds a value to represent a person's age. Print a statement that says "They can vote" if they're of age and a statement that says "They cannot vote" if they are too young.

Fig. 1. Example homework question (Java and Snap)

Once the course development was completed in June 2018, the material was given to some colleagues for equivalence testing. This involved reading the course materials in parallel and deciding if they were equivalent, as we needed others to verify our thoughts and decisions. Results of this testing were favorable, and some feedback was also obtained which led to further iterative development of the curriculum materials.

As well as the six major topics, the week 7 and week 8 material also had to be developed. Week 7 was a revision week which allowed the students to reflect on all of the concepts they had touched on over the previous weeks. This material contained worksheets with multiple short questions (one per topic) and two longer questions (using elements from all weeks) to pick from. Week 8 was an examination so a question and marking scheme needed to be written up for this session. The material for both courses can be seen on Padlet [1, 2].

IV. Course Delivery

During July and August of 2018, contact was made with a number of schools under the PACT initiative [11] to offer them

[1] https://padlet.com/mark_noone1/java
[2] https://padlet.com/mark_noone1/snap

the option of hosting one of the courses in the first term of the 2018 – 2019 academic year. These courses would be delivered in Transition Year, which is a one-year programme that forms the first year of a three-year senior cycle in many Irish secondary schools. An agreement was made with two chosen schools, with the following make up:

- School #1 – Mixed gender public school, four Transition Year classes, 15-16-year-old students, two classes would undertake the Java course and two classes would undertake the Snap course, approximately 40 students for each course.
- School #2 - All girls private school, one Transition Year class, 15-16-year-old students, approximately 20 students total, would undertake the Java course.

All five classes began in mid-October 2018 with the intention to run until the final week before Christmas for a total of eight weeks. There were some challenges with this which ended up pushing the last two sessions to late January 2019 after a month-long break from material, but all material was still delivered to all students.

One exception to this is that the "Arrays" sessions were not delivered to any class. This was due to the fact that Strings took longer for the students to comprehend and practice than originally expected. As such, Strings and User Input (originally a single week session) was expanded out to two weeks and arrays had to be dropped from the course. The revision and final exam were adjusted to account for this.

The first session began with an introduction to the programming environment of choice (JCreator for Java, Snap UI for Snap). The students were also given an overview of what the rest of the course would entail. Finally, they were asked to complete a short survey on their opinions of CS and what they expected from the course. Data was collected anonymously with each student being given an ID number to link their data with the survey in the final week.

All intermediate sessions began with a recap of what was covered in the previous week and a run through of the homework questions that were assigned. The new material was then delivered.

Week seven gave the students control to look over any material they previously struggled with. They were able to ask questions or simply work away on questions. The final session began with the post course survey and the course completed with a final examination.

V. OUTCOMES - EFFICACY

The final examination was delivered in week eight of the course delivery. Students were given 45 minutes to answer a short question which used an element of each week's material. The questions posed in Java can be seen in *Fig. 2* and the question posed in Snap can be seen in *Fig. 3.*

Once the examination was completed, a photograph was taken of each student attempt, saved as their ID number. After the course was completed, these attempts were then graded based on a marking scheme.

Write a Java program which:

1. Creates a String
2. Gets the value for this String using user input (use any sentence when testing)
3. Using a loop and selection, go through this String and only print every even positioned character. For example:

   Hello World -> HloWrd (0, 2, 4, 6, 8, 10)

Fig. 2. Java examination quesion

Write a Snap program which:

1. Creates a String variable
2. Gets the value for this String using user input (use any sentence when testing)
3. Using a loop and selection, go through this String and only print every even positioned character. For example:

   Hello World -> el ol (2, 4, 6, 8, 10)

Fig. 3. Snap examination quesion

The marking schemes gave either 0 (no attempt made), 5 (decent attempt made at using the element) or 10 (good or perfect use of the element) for each of ten different requirements to give a total of 100 marks. The ten marking elements for each language's examination are given in *Table I.*

TABLE I. MARKING SCHEMES FOR JAVA AND SNAP

Marks (/100)	***JAVA***	***SNAP***
10	Use of import	Use of a "When click" block
10	Use of class	Use of the "Answer" element
10	Use of a main method	Use of an update statement
10	Use of a String variable	Use of a variable with text
10	Use of the Scanner	Use of "Ask and Wait" input
10	Use of a loop	Use of a loop
10	Use of an if statement	Use of an if statement
10	Use of modulus	Use of modulus
10	Use of a print statement	Use of "Say" block to print
10	Use of charAt	Use of "letter X of" block

The average results from these tests were as follows:

- Snap – 34.4/100
- Java – 22.8/100
- Java in school #1 – 22.8/100
- Java in school #2 – 42.5/100

From these results, we were able to discern some important information. First of all, the overall grades were quite low. There any many possible factors for this including the limited timeframe for practice, the general difficulty of learning programming [12], the fact that the last two sessions (revision and examination) needed to be delivered after the winter holidays, and in school #1's case the lack of a permanent teacher present in the room. Additionally, it is worth noting that no marks would be awarded to students towards their end of year marks for Transition Year so a lack of motivation may be present.

For the null hypothesis that the Java examination would not be more difficult than the Snap examination; the results show that this is possibly the case. This was shown through the usage of a two tailed t-test which was the metric used for all further data analysis in this paper. A minimum p-value of 0.05 will be required to reject any null hypotheses. In this instance, we had $p = 0.3420$ ($p > 0.05$). This is not a statistically significant enough result to say if one exam was more challenging than the other. If this were true, it would imply that Java is not actually implicitly harder to learn than Snap is.

Interestingly though, when we break this down further and assume that the two schools were both significantly different environments, the results look a little different. We can affirm this by comparing the Java results of school #1 with school #2. When we do, we see a statistically significant difference in outcomes with $p = 0.0009$ ($p < 0.01$). This means that there's a greater than 99% chance that the differing environments between the schools had a significant effect on the study.

With this in mind, we can now compare the outcomes of the Java examination and the Snap examination in school #1 (which was also the school with the larger dataset with two classes each). We found that the Snap grades were significantly higher with $p = 0.01239$ ($p < 0.05$). This tells us that there's a very high probability that the Java examination was in fact more difficult to perform well in, and we can reject the null hypothesis for school #1.

This would align with Noone & Mooney's [2] previous findings that Java was a significantly harder language to learn. This is the key point of the study and requires further examination with additional cohorts of students in order to verify the findings.

VI. Outcomes – Survey Analysis

The examination results were only one metric analysed as part of this process. As previously mentioned, two surveys were also administered; the first one prior to the first session of the course which collected opinions on what they might expect from the course, if they might consider studying Computer Science at University and other collected survey data.

The second survey was administered right before the final examination to determine the easiest and hardest sessions from the student's vantage, their overall enjoyment of and difficulties with the course and once again whether they would consider studying Computer Science at University.

One important point to note as we delve into some of the results from these surveys is that the cohort who were there for the pre-survey did not exactly match the cohort who undertook the post-survey. There is much overlap but due to absences on both days, there are some who only sat one of the two surveys.

In *Fig. 4*, the opinions of the Java students on studying Computer Science at University can be seen and in *Fig. 5* the opinions of the Snap students on studying Computer Science at University can be seen.

In terms of the Java students, we found that with $p = 0.2349$ for the change in opinions between the pre course survey and the post course survey that there was no significant decrease. This implies that the course itself did not negatively affect their views of programming and Computer Science. In terms of the Snap students, with $p = 0.1795$ we can make the same conclusion.

What these results did show however is that some of the students were able to make their mind up over the duration of the course. This is an important thing given the current dropout rates in Computer Science course at University [13]. It is vital that students are aware of what Computer Science is before committing to attending third level and having exposure to courses like these ones will ensure this.

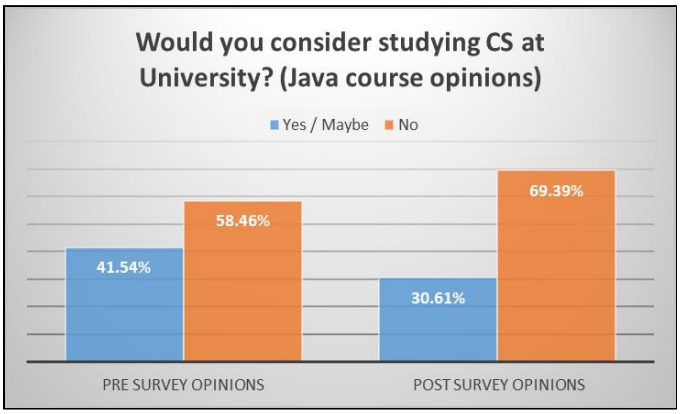

Fig. 4. Computer Science at University opinions – Java Course

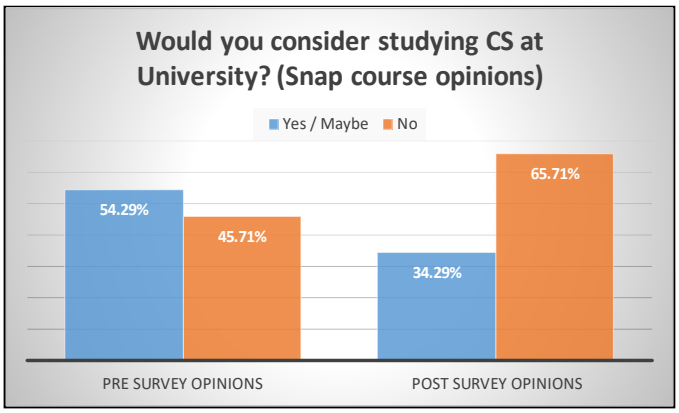

Fig. 5. Computer Science at University opinions – Snap Course

In *Fig. 6*, *Fig. 7*, *Fig. 8* and *Fig. 9*, the outcomes are shown for which course sessions the students found the easiest and most difficult. In *Fig. 6* and *Fig. 8* it is interesting to note that loops were considered the hardest to learn in Java, but not as many deemed it the hardest in Snap. Instead, Strings and User Input seemed to be the most difficult Snap session (which was still a highly chosen session in Java). In *Fig. 7* and *Fig. 9* it was clear that the introductory sessions eased students into each course and were overwhelmingly considered the easiest. This is much in line with what we would have expected.

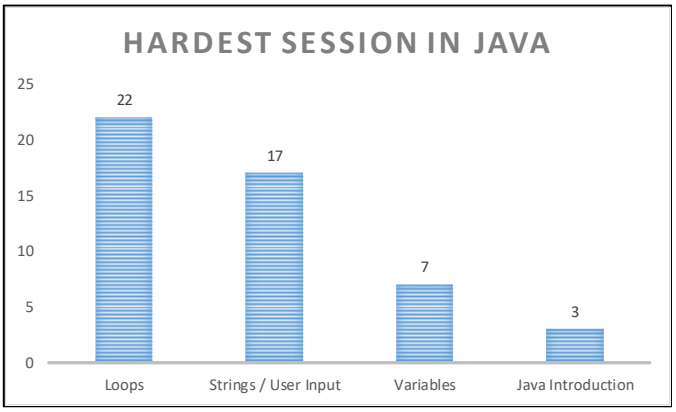

Fig. 6. Hardest Session to Learn in Java

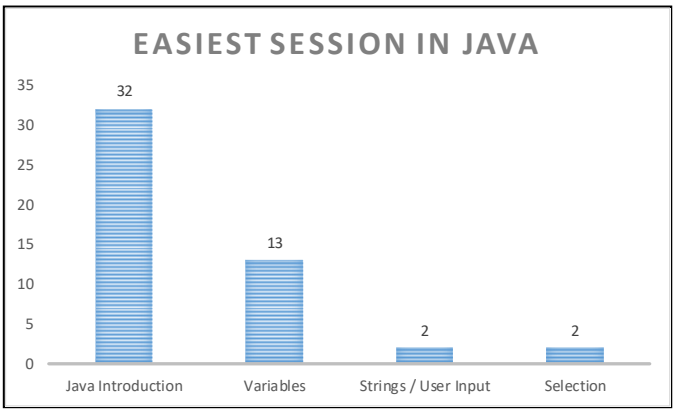

Fig. 7. Easiest Session to Learn in Java

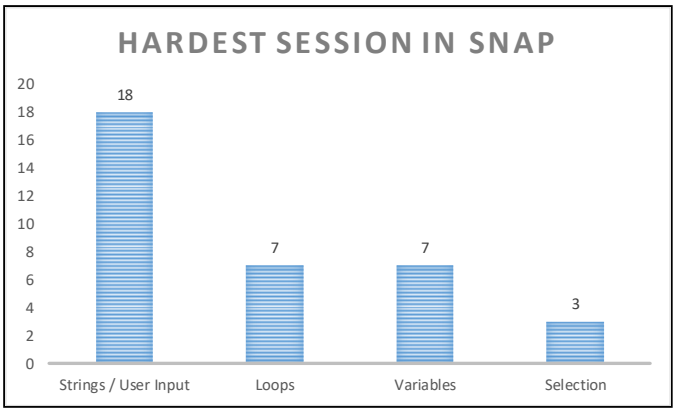

Fig. 8. Hardest Session to Learn in Snap

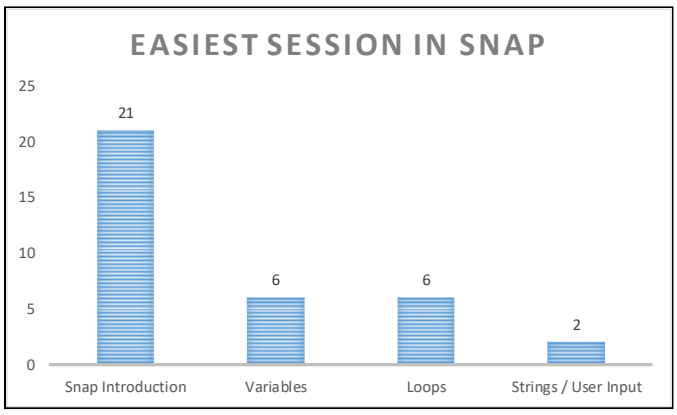

Fig. 9. Easiest Session to Learn in Snap

The two other key questions in the post course survey were "How much did you enjoy the course? (1= Not at all, 5 = It was OK, 10 = I loved it)" and "How difficult did you find the course? (1 star = easy, 5 stars = difficult)".

In terms of enjoyment, the average rating for Snap was 5.6 and the average rating for Java was 5.4694. The difference between the results for both courses was not significant with $p = 0.8027$. In other words, there was no difference at all in enjoyment levels of the courses. This is logical since the courses had identical content so enjoyment levels would be expected to be similar.

When it came to difficulty though, the results were more interesting. Java was rated 3.51 / 5 on average for difficulty. Snap however was only rated a 2.63 / 5. The courses were as identical in content as possible. Comparing the rated difficulties, we have $p = 0.0002$ which is significant at $p < 0.01$. This means that students found Snap to be much easier than Java, which again aligns with the previous findings of Noone & Mooney [2].

If we look even deeper at school #1 only (as they would have a similar cohort of students, they rated Java 3.69 / 5 in terms of difficulty) the result only gets more significant with $p = 0.0001$. This leaves very little room for the results to be chance meaning that Java is clearly more difficult than Snap. The only way we could be more certain is if the exact same students rated each language which is planned for a future phase.

This is once again key. It further exposes the fact that Java is more challenging to learn. This can only mean that some element of Java (verbosity, overhead or something else entirely) caused it to be more difficult to learn given the nature of the study with identical courses. If this is the case, why do we not teach CS1 in a VPL all of the time? Of course, this study can only make this assertion for students aged 15 – 16 years old. This is a younger cohort than we would see at University level. Further work will need to be done in order to verify if this is also the case for older (and younger) students. As it stands, it may be an isolated result for this age group or this specific cohort of students.

We also examined whether gender made a difference to the difficulty the students perceived with either course. We found that there was no measurable difference in the difficulty rating

for girls against boys in the Java course ($p = 0.5514$) or the Snap course ($p = 0.5033$). These results mean that within the same language groupings, gender did not have an effect on how students perceived the course.

Due to the low number of participants who took both surveys in certain cases (Males taking Java course $n = 8$, females taking Snap course $n = 9$) the results were inconclusive as to whether Java was more difficult than Snap amongst only males and only females respectively. The data did seem to be trending towards the overall conclusion that Java was more challenging, however. Females rated Java 3.36 / 5 on average for difficulty compared to their average Snap difficulty rating of 2.89 / 5. Likewise, males rated Java 3.63 / 5 on average compared to Snap 2.54 / 5.

Finally, we note some interesting comments that the students of the course made. These comments were made under the post-course survey question "Do you have any other comments or suggestions?". All feedback is valuable, and these comments will help shape future iterations of the courses.

- "Nope but maybe have a school teacher in the classroom" – From a student in the Snap course. This is very important given the difference between school #1 (no teacher present) and school #2.
- "I found that when I was doing the tasks with instructions it was easy, but when I had to do them on my own, I found it to be much more difficult." – From a student in the Java course. This is a common issue that faces many students in CS1 / CS2 courses.
- "It was a useful experience to get under my belt! I'm not sure if programming is what I want for my future, but it was good to get to know the basics and try it out." – From a student in the Java course. Programming courses in schools are important to help students decide early on if it would interest them as a degree choice.

VII. Future Work and Conclusions

The results from this study show promise. At least with the 15 – 16 age group, it does seem that Snap was easier to learn than Java was. We aim to further our analysis of this in a few ways. First and foremost, the courses presented in this paper will be refined and redelivered to more schools to see if the results can be replicated (in particular, we will prioritise school types we haven't yet delivered curricula to, i.e. private all boys school, private mixed gender schools or public single gender schools). We also endeavour to deliver the exact same courses to different age groups (younger children, CS1 University students, adult learners) to see if the results are the same or if the type of student affects the outcomes. Similarly, we would like to deliver both courses to the same group of students to see if they feel one was more difficult.

There were some challenges in the course delivery that need to be ironed out for the next phase. First and foremost, a permanent teacher from the school should always be present in the classroom to help keep student focus. Secondly, a long break is not recommended within the course delivery and may have skewed the results negatively. Finally, more time for practising would be preferred as some students didn't seem to take in the concepts.

As briefly mentioned in Section II, we are currently beginning work on developing a "hybrid" Java language. The concept of a hybrid programming language has been examined by multiple educators in recent years with promising results. In particular, Weintrop [7], Poole [14] and Harken [15] have looked at the concept in different ways. Weintrop [7] has created one of these environments with some promising early results. Harken [15] has commented that "browsability" is a key feature of visual blocks-based environments that reduced the complexity of learning them.

Our hypothesis is that a hybrid, drag and drop implementation of the Java programming language will reduce the difficulty of learning Java. We have already proven in this study that Snap was certainly easier, for 15 – 16 year old students, than Java. Using Snap's "build your own blocks" feature, we will be able to create custom blocks to match the syntax and semantics of Java keywords and code sections. With this created, we can create a third version of the course using the hybrid environment.

We hope to then see that Java is harder than Java Hybrid, and Java Hybrid is harder than Snap in terms of learning difficulty. We infer that this could be the case due to the browsability and blocks-based environment having enough of an effect to ease the complexity of Java's verbosity without trivialising the learning of the language. From looking at results of similar studies, we do not believe that it would make learning Java easy enough to put it on level ground with Snap and other purely visual blocks-based languages, however.

In the coming months, when this work is completed, we aim to run a short test (one session) of the efficacy of each of the three languages in a Summer Camp setting. If the anecdotal evidence from this seems promising, it will be ramped up and delivered in parallel to the Snap and Java courses in the next phase of the project.

Overall, the results of this study show promise for considering other forms of FPL rather than just a text-based approach. All approaches have their merits but with increasing drop-out rates [13] and student difficult with traditional CS1 courses as shown by Watson & Li [16], considering other approaches might just be the key to improving retention.

Acknowledgment

This work was assisted through the support of funding received from the John and Pat Hume scholarship, Maynooth University in 2016.

References

[1] D. Gupta, "What is a good first programming language?," *Crossroads*, vol. 10, pp. 1–7, 2004.

[2] M. Noone and A. Mooney, "First Programming Language : Visual or Textual ?," in *International Conference on Engaging Pedagogy (ICEP)*, 2017.

[3] M. Noone and A. Mooney, "Visual and Textual Programming Languages: A Systematic Review of the Literature," *J. Comput. Educ.*, vol. 5, no. 2, pp. 149–174, 2018.

[4] S. Sandoval-Reyes, P. Galicia-Galicia, and I. Gutierrez-Sanchez, "Visual Learning Environments for Computer Programming," *2011 IEEE Electron. Robot. Automot. Mech. Conf.*, pp. 439–444, 2011.

[5] MIT Media Lab, "Scratch Statistics," 2019. [Online]. Available: https://scratch.mit.edu/statistics/. [Accessed: 25-Apr-2019].

[6] J. Cheung, G. Ngai, S. Chan, and W. Lau, "Filling the gap in programming instruction: a text-enhanced graphical programming environment for junior high students," *ACM SIGCSE Bull.*, vol. 41, pp. 276–280, 2009.

[7] D. Weintrop, "Blocks , Text , and the Space Between," *2015 IEEE Symp. Vis. Lang. Human-Centric Comput.*, no. C, pp. 301–302, 2015.

[8] TIOBE, "TIOBE Index," 2019. [Online]. Available: https://www.tiobe.com/tiobe-index/. [Accessed: 01-May-2019].

[9] S. Davies, J. A. Polack-Wahl, and K. Anewalt, "A snapshot of current practices in teaching the introductory programming sequence," *42nd ACM Tech. Symp.*, p. 625, 2011.

[10] Linda Mannila, Michael de Raadt, "An objective comparison of languages for teaching introductory programming," *ACM, New York, NY, USA*, vol. 276, pp. 32–37, 2006.

[11] A. Mooney, J. Duffin, T. Naughton, R. Monahan, J. Power, and P. Maguire, "PACT : An initiative to introduce computational thinking to second-level education in Ireland."

[12] E. Lahtinen, K. Ala-Mutka, and H.-M. Järvinen, "A study of the difficulties of novice programmers," *ACM SIGCSE Bull.*, vol. 37, no. 3, pp. 14–18, 2005.

[13] C. O'Brien, J. Humphreys, and N. Ide McAuliffe, "Concern over drop-out rates in Computer Science courses," *Irish Times*, 2016. [Online]. Available: http://www.irishtimes.com/news/education/concern-over-drop-out-rates-in-computer-science-courses-1.2491751. [Accessed: 02-May-2019].

[14] M. Poole, "Design of a blocks-based environment for introductory programming in Python," *Proc. - 2015 IEEE Blocks Beyond Work. Blocks Beyond 2015*, pp. 31–34, 2015.

[15] A. H. Harken, "To block or not to block? That is the question," *J. Thorac. Cardiovasc. Surg.*, vol. 149, no. 4, pp. 1040–1041, 2015.

[16] C. Watson and F. Li, "Failure rates in introductory programming revisited," *Proc. 2014 Conf. Innov. Technol. Comput. Sci. Educ.*, pp. 39–44, 2014.