Teaching Programming Fundamentals to Modern University Students

Anton Bogdanovych and Tomas Trescak
School of Computing, Engineering and Mathematics, Western Sydney University, NSW, Australia

Keywords: Programming Education, Learning by Doing, Clara, Gamification.

Abstract: In this paper we investigate how teaching programming to the modern generation of students, "digital natives" who grew up with Google and Facebook and do not know the world before the Internet, can be improved through a highly visual game-like approach. Many programming teachers report that modern programming students have short attention span, lack concentration and have poor motivation to learn programming. We show how we were able to improve the motivation of students and their marks by changing the study program so that the entire entry-level programming course (Programming Fundamentals) is being taught using a visual set of in-class examples and assignments. The paper presents a set of successful teaching patterns that helped to convert one of the most hated subjects in our school into a subject that many students loved and were able to master. The corresponding statistics suggests that one of the key achievements of our approach is a dramatic change in students' motivation to learn programming, which has resulted a significant improvement in their overall results and was noticeable in the follow-up subjects.

1 INTRODUCTION

In recent years many programming teachers have noticed that teaching introductory level programming courses is associated with great difficulties for students (Bergin and Reilly, 2005). Educators are concerned with high failure and drop-out rates, poor academic performance and lack of motivation to study by modern students. Some researchers believe that the reason for this change in students' attitude to programming is a global phenomenon and is not only programming related.

In our digital age technology surrounds students from early childhood; they play video games, watch movies and use the Internet more frequently than reading books or going to the library (Prensky, 2007). As the result, the information becomes "cheap" and readily available, so educators have to compete for student attention with too many distractions introduced by technology as well as with other online material (Small and Vorgan, 2008).

Indeed, the amount of quality educational content on the Internet is increasing at an astonishing rate. Platforms like Coursera[1], provide open online access (sometimes for free) to courses from the world's best universities. Many universities themselves e.g. Massachusetts Institute of Technology[2] and Stanford[3] choose to upload their course materials online (including lectures videos, tutorials and solutions to exercises) and allow anyone to access them. This presents a great opportunity for students and, at the same time, creates a huge problem of how to manage the learning process in this "ocean" of information - vast, disorganised, and continually in a state of flux between updating and stagnating. The problem is further complicated by the new generation of students, who have grown up with Google and Facebook and who do not know the world before the Internet. The way these "digital natives" learn and how their brain processes information is believed to be different to the "pre-Internet" generations (Vassileva, 2008). Easily operating with multiple and diverse information streams at the same time, "digital natives" have learned to frequently multitask, but as the result of this multitasking they are believed to process information in a more superficial manner (Vassileva, 2008). In general, educators observe that contemporary students are less efficient in their school work, have shorter span of attention and tend to struggle with in-depth analysis (Small and Vorgan, 2008). This is not a reflection of being less capable to learn, but rather being disengaged with traditional instruction

[1] https://www.coursera.org/

[2] http://mit.edu

[3] http://stanford.edu

Bogdanovych, A. and Trescak, T.
Teaching Programming Fundamentals to Modern University Students.
In *Proceedings of the 8th International Conference on Computer Supported Education (CSEDU 2016) - Volume 2*, pages 308-317
ISBN: 978-989-758-179-3

(Prensky, 2007). Constant interaction with technology made modern students comfortable with multiple information streams; they prefer inductive reasoning, want frequent and quick interactions with content, and have good visual literacy skills, but this is not something that is easy to offer through traditional instruction (Prensky, 2007).

Despite the changing nature of the students, modern education often fails to adapt to their new learning habits and capabilities. In a traditional classroom the changing demands of digital natives are often ignored as teachers expect the same levels of focus, attention and self-motivation as from the pre-Internet students and are resilient to change their style of delivery and to engage with new technologies. We believe that this is one of the key problems that has to be addressed in order to improve the situation in teaching programming. Our key hypothesis is that modern educators have to adapt their teaching practices with a particular focus to improve the motivation of modern students. We strongly believe that in the world, where information is cheap, the role of educational institutions and teachers has to shift from simply being knowledge providers toward motivators and mentors (Palfreyman, 2008).

The advancement of video games and internet technology shows a huge drift towards entertainment in modern societies. This growth in the entertainment industry has certainly affected education as well as many other aspects of our lives (Pine II and Gilmore, 1998). Students demand engaging learning experiences in traditional learning environments and existing literature suggests video games as one of the best mediums to provide such experiences to students (Vassileva, 2008). The unique advantage of these games is the player's engagement: (Prensky, 2007) demonstrates it as the ability to keep people in their seats for hours and hours, actively trying to achieve new goals, determined to overcome their failures and immersed as part of this interactive entertainment. While digital natives are being criticised for their lack of concentration and poor motivation, at the same time they show these very characteristics when playing video games. In this work we will investigate the range of factors that make playing video games so effective and how to integrate these factors into the classroom when teaching programming fundamentals to university students.

The remainder of the paper is structured as follows. In section 2 we analyse the problem and attempt to understand the reasons causing it. Section 3 outlines existing research on teaching programming to digital natives with a particular focus on using game-based approaches. Section 4 outlines and provides motivation for the changes in teaching Programming Fundamentals in our school. Section 5 presents the research methodology and other details of the study that was conducted in our school. Section 6 outlines the results of this study and Section 7 presents further discussion of results and concluding remarks.

2 PROBLEM STATEMENT

In order to understand how teaching introductory programming courses could be improved, we first have to formulate the key problems that need to be addressed, understand the reasons for these problems and produce research hypotheses that can be tested in regards to successfully developing an appropriate solution.

2.1 Problems

To understand the problems associated with teaching introductory programming, we have conducted an extensive analysis of academic feedback in this respect. At the level of the dean of our school the problem with programming has been raised as an alarming phenomenon and the dean has initiated a regular discussion group among academics involved in teaching programming, as well as academics whose subjects require programming as a pre-requisite. Below is the summary of key problems raised by the academics:

- Lack of engagement and motivation to study programming
- Poor academic performance
- High dropout and failure rates
- Poor programming skills after the course

In order to obtain a more complete picture it was also important to understand the perspective of students. The student perspective was analysed by a number of qualitative and quantitative surveys that were conducted by two lecturers in their subjects, among the students who have completed Programming Fundamentals. The analysis of student survey responses showed that many students simply developed a strong sense of dislike to programming in general and could not really explain the rationale behind these strong emotional reactions. Apart from the emotional component, the key concerns of the students can be summarised as follows:

- The subject is too difficult
- Poor engagement in the classroom
- Fighting errors and compiling code

- Writing code and sticking to the confusing syntax of the programming language
- The textbook is hard to understand
- The lectures are hard to understand
- Not clear why learning programming is important
- Boring and difficult assignments

2.2 Reasons

Having the feedback from lecturers and students was important in understanding the key concerns of both sides, but even more important was to understand the deep roots of these problems that could be responsible for the identified problems. In an attempt to understand the reasons behind these problems we have initiated informal talks with academics and students, where the aim of establishing the high-level reasons for problems with programming was declared and an attempt to understand these was made through a brainstorming session in each group (a group of academics and a group of students). The discussion with students was the least productive one as students kept repeating the problems and were unable to offer something constructive beyond blaming programming to be too hard and useless, having bad teachers and the course being too difficult. Talking with academics proved to be more productive and the following set of reasons was formulated there:

- Digital natives require different instruction practices
- Low entry requirements for students starting at university
- Most modern students have to work and have little time for assignments

The initial analysis of problems in teaching programming and underlying reasons for those showed that further analysis was needed to understand how to improve the situation. So next we discuss our findings in the relevant literature and summarise successful practices in teaching programming to modern students.

As educators, we realise that from the list of the identified reasons for the problems associated with teaching programming we will not be able to offer a solution to addressing low entry requirements, which is the job of university management rather than educators. We will also not be able to change the working patterns of the students. Therefore, the key focus of our further analysis is on improving the instruction practices, so that they are more suitable for digital natives and improving student motivation is the prime focus of our work.

3 BACKGROUND

When looking at developing better teaching practices for educating digital natives some educators claim that the focus of modern universities has to change from providing information to providing mentoring services (Palfreyman, 2008). The times when universities have actually possessed the knowledge and could easily restrict the outsiders from accessing it are long gone. Now most of the necessary information is freely available and is only one click away, so the role of modern universities has to change from simply giving the information to students (hoping that they are motivated enough to absorb it) to selecting the relevant subset of information and making sure they are focusing on the right things and are progressing in the right direction, which essentially is mentoring rather than lecturing. The lack of such mentoring services is believed to cause education to become less “higher” and more “tertiary” as it shifts from a pedagogical emphasis on liberal education to a skills/curriculum dominated agenda (Palfreyman, 2008). Such shift in education, in the authors’ view, fails to properly prepare graduates for rapidly changing economic conditions and related employment opportunities (Palfreyman, 2008).

Offering personalised mentoring and tutoring services is not something that is only relevant to digital natives and not something that has to be fully re-invented for the modern age. Building education around mentoring was one of the key features of better world universities in the late 1800s and is often named as the key reason for high quality and price of the degree provided by these institutions (Palfreyman, 2008). Each student back then was provided with a personal tutor whose key role was in mediating students’ access to knowledge. One of the fundamental works on assessing the tutorial system in (Moore, 1968) claims that the tutor is not teacher in the usual sense: it is not his job to convey information, but the role of the tutor is to act as a constructive critic and help in sorting out the knowledge of the student. The usefulness of inquisitive nature of the tutor is supported by the French philosopher Peter Abelard, who said “the key to wisdom is this — constant and frequent questioning... for by doubting we are led to question and by questioning we arrive at the truth” (Gaskin, 1998).

While we can rely on the teaching practices of the past, we should not ignore the new demands of the modern age. Universities can no longer afford having small classes and offering personalised mentoring services to each of the students. The fact that the vast majority of the students are digital natives also forces

us to integrate corrections in the teaching practices. Apart from mediating access to knowledge and questioning, one of the key demands of our times is to help students with their motivation to learn. There is strong evidence suggesting that intrinsic motivation to study programming has a strong correlation with academic performance (Bergin and Reilly, 2005).

As per findings from (Prensky, 2007), a good way of improving motivation of digital natives is to use highly visual interfaces and employ game-like assignments and lecture examples. Graphical Frameworks in teaching Programming Fundamentals have become an effective motivational mechanism (Moskal et al., 2004). This approach is growing in popularity, especially in primary school programming classes. Systems like ALICE (Dann et al., 2011) and Scratch (Resnick et al., 2009) have been successfully used in primary and secondary schools, but they are often abandoned by university educators for their simplicity, specific appearance and due to the fact that they teach students framework specific languages that cannot be used beyond these tools.

Another framework that is being used at some universities is Karel (Pattis, 1981). Karel represents a Java plugin that can be used together with classical Java editors and, essentially, teaches pure Java rather than some other framework specific language. The Karel approach is to teach programming in a discretised environment, where students control a robot using a small set of standard commands: move(), turnLeft(), putBeeper(), removeBeeper(). Figure 1 shows an example of a typical interface of a Karel assignment. An extensive explanation of the Karel framework is presented in (Pattis, 1981).

In most Karel-based courses students use this interface for a short period of time (e.g. three weeks like in Stanford) and then move on to pure Java. In our study we use a similar approach to Karel, but extend it to the entire course. Karel is rather outdated and its black-and-white graphics lack modern appeal. A more advanced framework is Greenfoot (Koelling, 2010). We found Greenfoot to be suitable for our purposes, but we have decided to further enhance it with gamification and automatic student feedback features, so we have developed our own framework called “Clara”. All assignments worked both in Greenfoot and Clara and students had a choice to select one of the two frameworks for working on their assignments.

Improving academic performance through motivation was one of our key goals, but another goal was to improve the situation with high dropout and failure rates. These, apparently, are not easy to fix. While high motivation is certainly connected with dropout and failure rates, introductory programming requires dedication and a lot of work and high dropout and failure rates are a norm. A study in (Bennedsen and Caspersen, 2007) shows that indeed the failure rates are significant (33% on average from 80 institutions who participated in the study). The authors also mention that they do not consider 33% being a very high number as their study has also shown that the number of people enrolling vs the number of people graduating from a university is around 27%.

4 SUMMARY OF CHANGES

Based on the analysis of the data obtained from staff and students, as well as using the results of the literature review we have decided to introduce a number of changes in the introductory programming unit in our school (Programming Fundamentals). In order for us to be able to measure the results of the introduced changes we made a decision to maintain a high degree of similarity in terms of lecture material, the degree of difficulty of assignments and tests as well as the structure of the new course and the original course. In this way if there is indeed any difference in the course outcome it can be attributed to specific changes we made rather than the new structure of the course or its difficulty level. Next, we explain the key changes that were introduced in the new Programming Fundamentals course.

4.1 Highly Visual Programming Examples

One of the key novelties of our approach was to make all examples shown in lectures and assignments being highly visual. We wanted to benefit from the success of highly visual systems like ALICE (Dann et al., 2011) and Scratch (Resnick et al., 2009), but after discussions with other academics and potential employers, conversations with students and the analysis of the job market we have decided to keep a strong focus on predominantly teaching a popular programming language and minimise teaching other things that are not directly related to this language. In case of Scratch and ALICE, while they offer highly visual and intuitive frameworks, a lot of the knowledge gained as the result is the knowledge of the framework itself, as well as the knowledge of specific commands that are only used in those environments. Furthermore, our analysis of the teaching resources offered with Scratch suggests that this framework is more suitable for primary and secondary school and is not a good fit for teaching at university. As the result, the approach

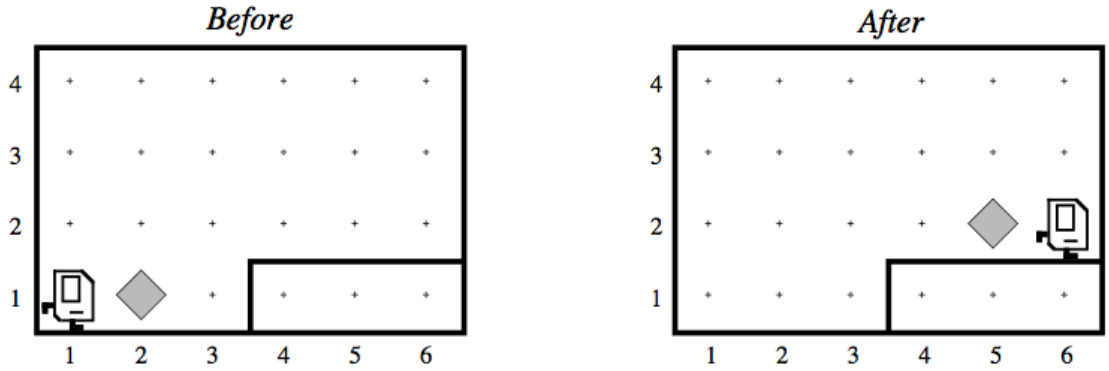

Figure 1: An example of a Karel exercise. Picture source (Pattis, 1981).

we took is similar with that of Karel the Robot used at Stanford (Pattis, 1981). The framework we use (Clara) is similar to Karel, but has a modern graphical interface that is more appealing and also includes various gamification elements. Unlike most Karel based courses our entire course was taught using the visual Clara approach.

Figure 2 shows an example of one of the practical assignments as well as the interface being used there. On the right hand side of the figure there is a practical assignment showing the main character (ladybug Clara) located in a discrete environment. Here Clara must find her way out of a maze. The exit is marked by a leaf, which after finding she has to eat. On the left hand side there is a programming solution for this problem. The code is written in pure Java and uses a small set of additional Clara specific commands (move(), turnLeft(), turnRight, onLeaf(), removeLeaf()).

Using the Clara approach we were able to translate the entire programming course to be based on problems with Clara. All programming assignments and lecture examples are Clara problems.

4.2 Problem-Oriented Lectures

From our previous teaching experience we know that students respond well to stories that touch upon things that interest them. So, to better engage with students and capture their attention we built all lecture material around interesting and challenging practical problems, which can be translated into terms of a ladybug manipulating leaves and moving around a discrete environment. Each lecture would start with a story or a video focused on some interesting internet phenomenon, a popular hobby topic or just some fun story from a TV series. After discussing the story further lecture material was presented as new knowledge that could help to solve the problem discussed through the story in the beginning of the class. Most of such stories that we used were from the domains of gaming and entertainment. Such problems ranged from building a robot vacuum cleaner to implementing the friendship algorithm from the Big Bang Theory[4].

4.3 Learning by Doing and Interactivity in the Classroom

One of the important elements of the digital natives is their predisposition to learning by doing (Schank et al., 1999). To address this need each of the lectures was structured so that a bigger problem presented in the beginning of the class was broken down into smaller sub-problems and these smaller sub-problems were interactively solved during the lecture using the new knowledge that was learned in the class. Solving the sub-problems in the class represented a moderated process, where the lecturer presented the problem and asked the students for help with continuing to solve it. The moderation helped to make sure that the problem is being solved in a focussed manner. Apart from simply asking students for help and typing their proposed commands, we have also experimented with more engaging forms of interaction. One of such forms is to translate the process of problem solving into interactive theatre. In this interactive theatre the students enacted the elements of Clara's world (e.g. each student enacting a leaf, a tree or Clara) and other students controlled the changes in the simulated environment by proposing commands to be enacted.

4.4 Generating Flow and Making Programming Fun

A good way to motivate students is to generate flow, which is the state associated with a high degree of excitement. One of the conditions that allows for experiencing the flow state in the classroom is to structure

[4]http://the-big-bang-theory.com

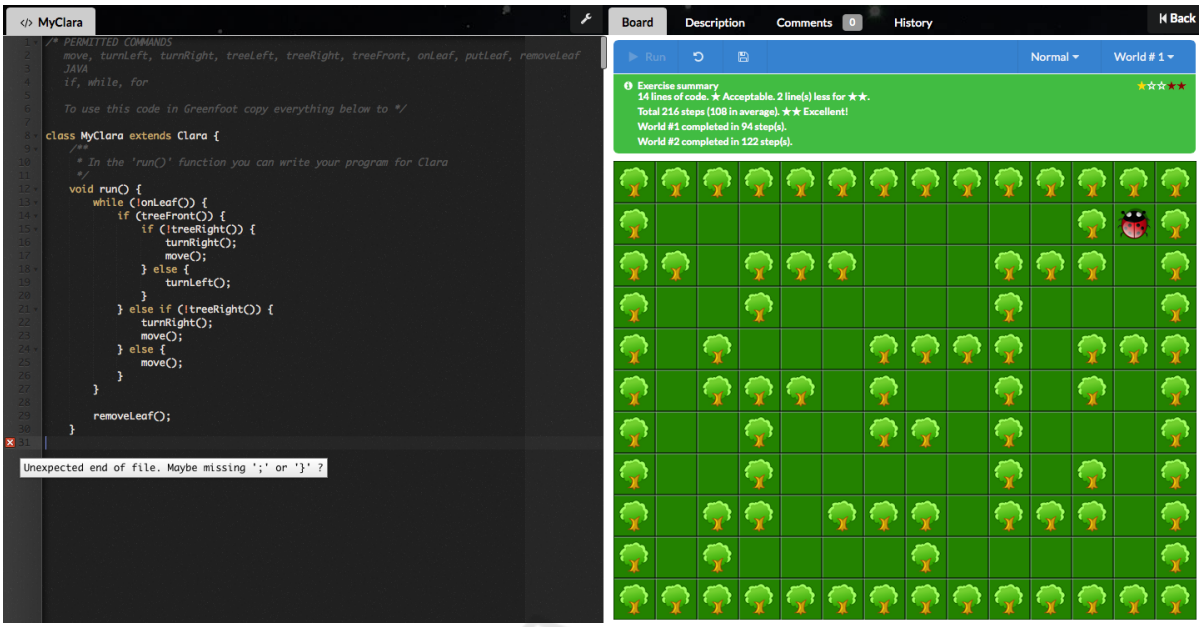

Figure 2: Clara's World: An online assignment in Programming Fundamentals.

the practical assignments so that they incrementally increase in difficulty and the difficulty of each new problem is such that the amount of skills required to solve it is slightly above the current level of skills of the person solving it (Csikszentmihalyi, 2002).

Flow is associated and is considered to be much easier to achieve in video games as games environments often represent puzzles or problems to solve and can be easily structured by the aforementioned principle of incrementally increasing the difficulty of the problem together with the growing level of player skills. An important work that documents this phenomenon in video games is (Koster, 2013). This work does not explicitly explore the concept of flow, but investigates the principles that make video games fun and enjoyable. One such principle (that can potentially address the issue of high dropout rates) is that players will quit playing the game if it is too hard to win or if it is too easy to win. So, the increase in difficulty should be carefully tested on an average player to make the game fun (Koster, 2013).

Both lectures and practical assignments were carefully designed to potentially be fun and generate the flow. To be able to do so, we have modelled an average student and made practical assignments and examples being solved during the lectures incrementally more difficult, so that the skills required for solving each problems are slightly above the current set of skills of an average student.

While we had clear cases of reaching the state of flow by many students that we were able to witness in the tutorial rooms, this approach is also associated with the drawback that students who for some reason missed some of the assignments showed much lower engagement thereafter due to the growing gap between their skills and skills required to solve the problem.

4.5 Maintaining Motivation via Gamification and Social Experience

Apart from generating flow, having highly visual assignments and facilitating learning by doing (Victor, 2012) presents a number of other recommendations for building programming frameworks that could improve student motivation. Providing quick (or even automatic) feedback in case of errors, reflecting on the quality of the student code are some of the factors that are claimed to be responsible for better understanding of the programming concepts and improved motivation to learn programming. Additionally, (Prensky, 2007) states that offering a competition element to the coding and using gamification principles are also known to be associated with an improved motivation to study.

Therefore, in our online Clara framework[5] we have incorporated some of these features. The Clara framework employs game-based learning and gamification approaches to motivate students to learn, continuously improve their solutions and possibly design their own in their spare time (Figure 2 depicts the online editor). We studied and evaluated effects and

[5]http://pf.scem.uws.edu.au

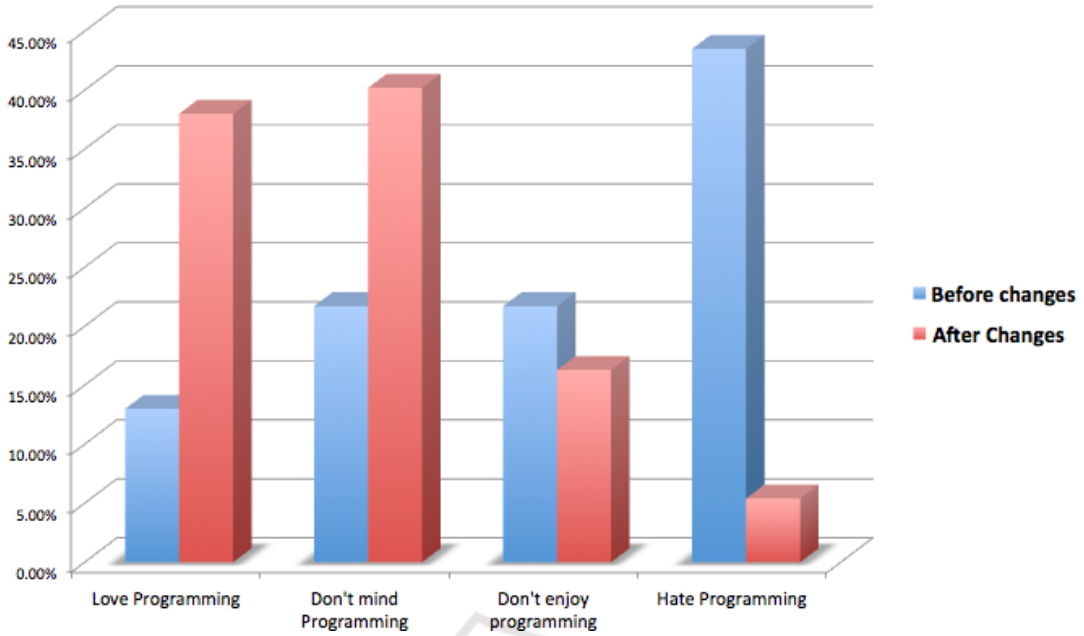

Figure 3: Programming attitude.

significance of various gamification approaches. To achieve our goal, we decided to employ *star ratings*, *badges* and *challenges with user created content*.

To motivate students to submit better code we used the approach popular in games like Angry Birds[6]. Each submitted solution received up to five stars for the shortest and most effective code. Five stars were awarded when Clara used the least amount of steps to complete the solutions. Completed solution received hints on how much the solution needs to be improved to receive a better star rating. Results were published in leaderboards and often were the target of heated discussions between students fighting for the best result. Anecdotal evidence obtained from the tutors suggests that students who have used the Clara framework were less eager to share their solutions with their colleagues and they preferred to not to reveal their solutions but to guide other students to come up with their own.

Another form of using gamification was to award badges to the best performing students with the most efficient code, as well as for the first five solutions turned in the given week. This approach encouraged students not to procrastinate, and to complete their solutions as fast as possible. Badges could be "stolen" by creating a better solution than the badge holder. In this case, badge holder would be notified to try to get the badge back by revisiting and improving the solution, effectively improving student engagement.

Challenges with user generated content aimed at highly motivated students that completed their assignments fast and wanted to improve their expertise by creating their own set of problems, publishing them online and challenging their colleagues. Often, such students created problems that were then used in the class to explain a specific topic, which further encouraged students to actively participate in the educational process.

Our online system also improved communication between students and tutors, where solutions could be discussed remotely, not only in class. In this scenario, both student and tutor connect to the same page, which is automatically updated, allowing for collaborative coding and presentation of results.

5 USER STUDY

In order to find a solution for improving the situation with teaching Programming Fundamentals in our university we have combined the knowledge obtained from academics and students together with the findings from the literature. As the result, we have formulated the following research hypotheses:

5.1 Research Hypotheses

- **H1:** Supplying visual and animated game-like examples throughout the course would help to increase interest in the subject and motivation.

[6]https://www.angrybirds.com

- **H2:** Generating flow is associated with increased motivation.
- **H3:** Student academic performance will improve as the result of increased motivation.
- **H4:** Adding Gamification and social elements would further increase student engagement and their academic performance.

5.2 Research Methodology

The introductory programming unit at our university is composed of the two hour lecture and two hour laboratory session every week over one semester. In general, most students in Australia do not study programming before entering university. The majority of students in the Programming Fundamentals course (58.7% of students in Autumn 2013) had no prior programming experience.

Performance in this module is based on continuous practical assessments (30% of the overall mark), major assignment (10% of the overall mark), mid-semester test (10% of the overall mark) and the final examination (50% of the overall mark).

A number of studies were carried out in the academic years 2013-2015. Students enrolled in the first year "Programming Fundamentals" course in our department voluntarily participated in this study.

The study aimed at testing H1-H3 has been carried out in Autumn 2013. Overall 458 students were enrolled in this course in Programming Fundamentals in Autumn 2013. Only 92 students have decided to complete the study survey. This survey was conducted after the final lecture, when all the practical assignments have been completed and only the final exam was pending.

Testing H4 was carried out throughout 2014-2015 with 160 students in total. The students had a choice to employ either the Greenfoot framework (90 students) or our experimental Clara framework (70 students) that featured gamification and social elements. The performance of the students was evaluated to see whether the choice of the Clara framework would have any impact in regards to H4. An additional qualitative survey with overall 90 students has been conducted to understand particular benefits of using the Clara framework.

6 RESULTS

The outcomes of the study confirm the validity of hypotheses H1-H4. The evidence obtained in favour of H1-H2 comes from a survey that was conducted in Autumn 2013 where students explicitly report on visual aspects of the course contributing to their engagement and better understanding of the course material. Some students explicitly report about them reaching the state of flow and programming becoming "addictively enjoyable" as the result of this.

6.1 Impact of Visual Delivery and Assignments Designed to Generate the Flow

The results of our study show that the attitude toward programming has dramatically changed after the introduced changes. From the students who were interviewed after completing the previous version of the course 43.5% have reported that they hate programming and only 13% said that they love it. In an online survey that was completed by the students enrolled in the new version of the course (after sitting through all lectures and finishing all their assignments) indicates that only 5.4% of the students claim to "hate programming" and 38% said that they love programming. Figure 3 shows the comparison between student attitude toward programming in the previous course (labelled as "Before changes") and the course after we've introduced the changes to the study program (labelled as "After changes").

6.2 Changes in the Academic Performance

Due to maintaining a similar level of difficulty in the assignments and in the final exam across the new course and the old course we were able to measure the impact of the introduced changes on the academic performance of the students (testing the H3). Ensuring the similarity across practical assignments was a difficult task, as the nature of assignments has changed from printing numbers on the screen to moving a lady bug and eating leaves, but we ensured that each practical problem in the new course requires a comparable amount of time for an average student to solve it and targets the same topics of the course. Maintaining a similar level of difficulty in the exam was achieved by reusing the same exam questions that were used in previous years of teaching Programming Fundamentals. The new exam paper was a compilation of exam questions from multiple past exam papers, but all questions had to be adopted to Java from C++. It is important to note that despite teaching the entire course in the visual framework all exam questions were pure Java questions and did not feature Clara. This was done so that we could achieve better

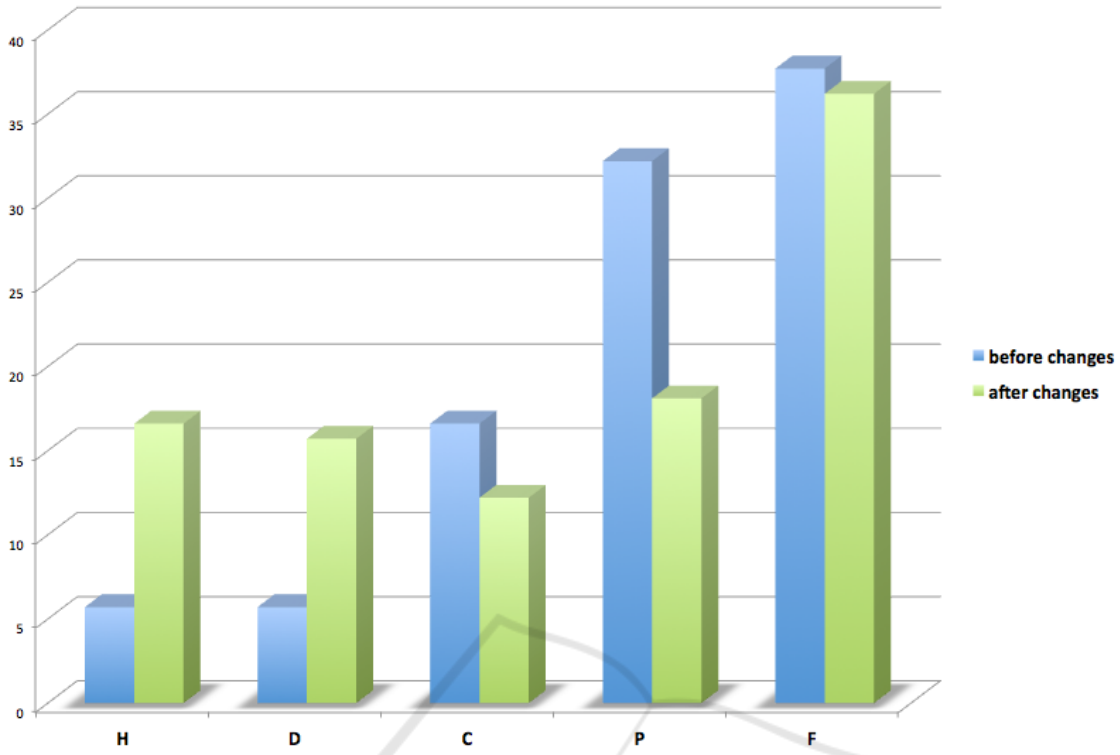

Figure 4: Marks comparison.

comparison. Given that in order to pass the subject students had to pass the final exam we can claim that students graduating from this unit had pure Java skills and not only Java in the context of Clara.

One of the key achievement that we would like to report is the difference in the grade distribution that we connect to the difference in the student motivation. Figure 4 features a comparison between two versions of the Programming Fundamentals course with clear indication of the percentage of students receiving the corresponding mark in the course "before changes" and "after changes". The marks are coded as follows: "HD" stands for High Distinction, "D" stands for Distinction, "C" stands for Credit, "P" stands for Pass and "F" stands for Fail.

Figure 4 shows that in the previous version of the course the majority of the students (32.2%) received the pass mark and only a small minority (5.7%) received High Distinction and Distinction. In contrast. after introducing the changes the mark distribution looks rather flat. Many more people have achieved higher marks with 16.6% receiving High Distinction and 15.7% receiving Distinction. The number of students receiving Pass and Credit has significantly declined. Our explanation of this phenomenon is that in the updated version of the course the students have shown an improvement in their level of motivation to study programming and, hence, they were much more keen to obtain higher grades.

6.3 Impact of Gamification Elements

The impact of the gamification elements (H4) introduced in the Clara framework was measured using quantitative and qualitative approach. The qualitative measure came in the form of surveys, given to ten groups of students in three semesters. During this survey, students responded to questions regarding their experience with both original solution (i.e. Greenfoot) and the online system. The results show that students prefer the innovative online solution and find the reward system highly motivating, making them revisit and improve their solutions. This was happening very sparsely with the original system. Students have particularly enjoyed the automated feedback in the form of detecting their errors as they type, helping to identify infinite loops etc. Students reported that such automated feedback improved their submitted code.

The quantitative measure evaluates the quality of the student code. On average, students who used the gamified system wrote almost 20% shorter code and 17% more effective solutions (taking less steps to complete). Moreover, no online solution has ended in an infinite loop, while almost 8% of offline solutions did. One of the greater successes of the online system is that 0% of students failed the unit for not achieving

required points from practicals. Points received from practicals were 16% higher than average and completion of the unit is 15% higher than average.

7 CONCLUSION

We showed how by basing the lectures and all assignments of the Programming Fundamentals course around game-like visual examples structured so that an average student is likely to encounter the state of "flow", we were able to dramatically improve students' motivation to learn and interest in the subject. To make it a fair comparison the level of difficulty of the practical assignments as well as the difficulty of questions in the final exam was made similar to those that were used in the Programming Fundamentals course prior to introducing the aforementioned changes. The results show that while maintaining a comparable level of difficulty of the assignments, the grade distribution has shifted towards High Distinction and Distinction and the student feedback has become much more positive. Moreover, despite learning in a highly visual framework and only with visual animated examples the students were able to successfully learn pure Java. The exam questions were deliberately designed to include no Clara related questions and were made comparable with those from previous years. The students showed good performance on the exam and experienced no difficulty with switching to console programming in pure Java as reported by the lecturers from the follow-up courses.

Additionally, we have obtained evidence suggesting that the inclusion of such gamification elements as awarding badges, employing ratings and introducing challenges helped students to produce better coding solutions and had a positive impact on their academic performance (when compared with students who were not exposed to these features).

REFERENCES

Bennedsen, J. and Caspersen, M. E. (2007). Failure rates in introductory programming. *SIGCSE Bull.*, 39(2):32–36.

Bergin, S. and Reilly, R. (2005). The influence of motivation and comfort-level on learning to program. In *Proceedings of the 17th Annual Workshop of the Psychology of Programming Interest Group*, pages 293–304, University of Sussex, Brighton.

Csikszentmihalyi, M. (2002). *Flow: the classic work on how to achieve happiness*. The Random House Group Ltd, London, UK, 2 edition.

Dann, W. P., Cooper, S., and Pausch, R. (2011). *Learning to Program with Alice (W/ CD ROM)*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition.

Deterding, S., Sicart, M., Nacke, L., O'Hara, K., and Dixon, D. (2011). Gamification. using game-design elements in non-gaming contexts. In *CHI'11 Extended Abstracts on Human Factors in Computing Systems*, pages 2425–2428. ACM.

Gaskin, R. (1998). The Philosophy of Peter Abelard by John Marenbon. Cambridge University Press. *Philosophy*, 73(2):305–324.

Koelling, M. (2010). *Introduction to Programming with Greenfoot: Object-Oriented Programming in Java with Games and Simulations*. Prentice Hall, University of Kent.

Koster, R. (2013). *Theory of fun for game design*." O'Reilly Media, Inc.".

Moore, W. G. (1968). *The tutorial system and its future, by Will G. Moore*. Pergamon Press Oxford, New York,, [1st ed.] edition.

Moskal, B., Lurie, D., and Cooper, S. (2004). Evaluating the effectiveness of a new instructional approach. In *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '04, pages 75–79, New York, NY, USA. ACM.

Palfreyman, D. (2008). *The Oxford tutorial : 'Thanks, you taught me how to think'*, volume 2nd. OxCHEPS, Oxford.

Pattis, R. E. (1981). *Karel the Robot: A Gentle Introduction to the Art of Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition.

Pine II, B. J. and Gilmore, J. H. (1998). Welcome to the experience economy. *Harvard Business Review*, pages 97–105.

Prensky, M. (2007). *Digital Game-Based Learning*. Paragon House.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., and Kafai, Y. (2009). Scratch: Programming for all. *Commun. ACM*, 52(11):60–67.

Schank, R. C., Berman, T. R., and Macpherson, K. A. (1999). Learning by doing. *Instructional-design theories and models: A new paradigm of instructional theory*, 2:161–181.

Small, G. W. and Vorgan, G. (2008). *iBrain : surviving the technological alteration of the modern mind*. Collins Living, 1 edition.

Vassileva, J. (2008). Toward social learning environments. *IEEE Trans. Learn. Technol.*, 1(4):199–214.

Victor, B. (2012). Learnable programming : designing a programming system for understanding programs.