The TuringLab Programming Environment
An Online Python Programming Environment for Challenge based Learning

Henry Miskin[1,*] and Anandha Gopalan[2]
[1]*TuringLab Ltd, London, U.K.*
[2]*Department of Computing, Imperial College London, 180 Queens Gate, London, SW7 2RH, U.K.*

Keywords: Computer Programming, Information Technology, Online Learning, Python Programming.

Abstract: Computing has recently been introduced as a core subject in British schools, meaning that children need to learn computer programming. Teachers have to be prepared to be able to deliver the new curriculum, but many of them do not feel confident teaching it as they have no formal background in Computer Science. Also, when learning to programme, children need the correct environment and support to succeed. This paper presents TuringLab, an environment to assist teachers in delivering the practical elements of the computing curriculum, while also proving to be engaging and challenging for the children. Teachers can create programming challenges for their pupils and see how they are progressing (or struggling) during completion of the challenges. Students can undertake challenges in an engaging environment which displays a graphical output of their code and assists in understanding errors they may encounter. TuringLab has been used to teach children how to programme at a number of volunteer-led coding clubs. Children engaged well with TuringLab, and the volunteers, who acted as teachers in these sessions, found TuringLab an extremely valuable educational tool.

1 INTRODUCTION

Technology is transforming the world in which we live and children of the future need to be provided with the knowledge and skills to keep up with this change. The ability to understand the underlying functionality of technology has become a required skill of the modern world (Stergioulas and Drenoyianni, 2011).

The importance of computing knowledge is reflected in the British Curriculum, as computing has recently been introduced as a core subject in both primary and secondary education (Gove, 2014). Children from the age of 5 need to learn to programme (Cellan-Jones, 2014) and beyond the age of 11, children are required to learn a syntax based programming language (Department for Education, 2013).

Teachers have to be prepared to teach the new computing curriculum, which is a daunting prospect as many teachers do not have a formal background in computer science. Students must be taught both the fundamentals of computing science and computer programming: the practical element of computing (Jones, 2015). A recent study found that although teachers are very enthused by the new curriculum, many do not feel confident in teaching it (Computing at School, 2015).

To assist teachers in delivering the new computing curriculum, there are a number of resources available for teaching the fundamental concepts of computing (OurICT, 2015). Additionally, many websites exist that can be used to help children learn to programme; they are well-developed, interactive and initially engaging. The tutorials provided on such sites incrementally build skills to develop programming concepts (EdSurge, 2015). Fully online courses have been found to achieve positive results (Flanagan, 2013), however, they are limited in how effectively they maintain the engagement of children, as is shown by high attrition rates (Zheng et al., 2015).

Current systems such as those listed above seldom offer collaboration or interaction, with the result that each child operates as a discrete unit. Furthermore, with highly structured tutorials, they provide very little ability for teachers to influence the content which the children consume. As a result, the current tools are not always applicable to a classroom environment and cannot be used for bespoke or cross curricular projects. The shortcomings of online courses present

*http://www.turinglab.co.uk

Miskin, H. and Gopalan, A.
The TuringLab Programming Environment - An Online Python Programming Environment for Challenge based Learning.
In *Proceedings of the 8th International Conference on Computer Supported Education (CSEDU 2016) - Volume 1*, pages 103-113
ISBN: 978-989-758-179-3

a demand for a tool that assists teachers in delivering the new computing curriculum while maintaining teacher involvement and control. The ideal learning environment will provide the intersection between the interactivity of online courses and the adaptability and engagement of physical teaching. By reducing the work required for teachers to set programing tasks and provide feedback to children, such systems allow teachers to spend more time providing differentiation and support.

This paper presents TuringLab, an online programming environment which assists in delivering the new computing curriculum. This application has been used to teach children Python programming at a number of volunteer-led programming sessions. Iterative feedback from children testing and using this software has enabled the current version to be fine-tuned to the needs of real classroom situations. Children returned week on week to the sessions and also continued to use TuringLab outside of structured sessions. Feedback collected from children was positive: they enjoyed using TuringLab and found the challenges engaging and enjoyable to complete.

2 BACKGROUND

The merits of TuringLab are dependent on its working harmoniously with current teaching practices, while optimising the mechanisms through which students best learn technical topics. To achieve an educational learning environment, a number of learning theories from existing literature are discussed first (Section 2.1) after which existing systems similar to TuringLab are reviewed (Section 2.2).

2.1 Existing Literature

The TuringLab system is a tool for teaching computing to students. Literature on learning theory and learning to programme was reviewed to consider how children could learn computing most effectively using TuringLab. In this section, existing literature relevant to TuringLab is discussed.

2.1.1 Learning Theory

The three commonly known learning theories: behaviourism, cognitivism and constructivism can be used as a basis for understanding the practices that are applicable to e-learning. The implications of these theories on implementing e-learning courses is reported from a number of sources by (Alzaghoul, 2012). The findings from each of the common learning theories is presented throughout this section.

From a behaviourist approach the material to be learnt needs to be carefully broken down. Additionally, learners need to be told the outcomes of learning and assessed to check that they have met their required outcomes. (Mödritscher, 2006)

Formative assessment which is outlined by (Black and Wiliam, 1998), as a means of tailoring learning activities based on the completion of previous outcomes of learning, allows learning to be tailored to students based on their individual progress and difficulties. Strategies to facilitate assessment for learning are outlined in (Black et al., 2003; Cooper and Adams, 2007; Knight, 2008) including: providing explicit learning objectives to pupils, making use of peer assessment and providing immediate feedback to pupils.

Cognitive school of learning suggests that learning material should be received in the form of sensations before perception and processing. The location and display of information is key to the learning outcomes for an individual while the difficulty of the material should match the cognitive ability of the learner. (Anderson, 2008)

Maintaining the difficulty of material at the correct cognitive level for a learner is referred to as the zone of proximal development, where children are identified as needing the correct scaffolding through challenge completion (Vygotsky, 1978; Wood, 1998). This is further developed by (Brophy, 1999; Jurišević, 2010) who outline the two zones of proximal development: the cognitive and motivation zones, which suggests the need for children's motivation to be maintained. A student's motivation is broken down into attention, relevance, confidence and satisfaction by (Keller and Suzuki, 1988).

Constructivists see learners as active agents in the discovery of knowledge where learners should be in control of their learning. Instructors have to provide good interactive instructions and learning needs to be meaningful and illustrative for the learners. (Alzaghoul, 2012)

Active learning is identified by (Adler, 1982) as a type of learning where the student is the agent of discovery which corresponds closely to the constructivists school of learning. Strategies and environments which most befit active learning are outlined in (Brophy, 1987; Grabinger and Dunlap, 1995); all advocate authentic, meaningful, collaborative and achievable tasks which call upon higher order cognition, cross curricular knowledge and a focus on the task at hand as opposed to assessment.

The role of technology in learning is discussed by

(Huffaker and Calvert, 2003; Pearlman, 2009; Walser, 2011), who describe how technology has immense capacity to create quality learners who are well placed to solve real-world problems and in doing so develop deeper knowledge. This is achieved through using technology interactively and resourcefully and to seek out knowledge relating to the problem at hand. The flexibility of remote learning in higher education was found to benefit students as it created a better learning philosophy (Twigg, 2002; Gordon, 2014; Arkorful and Abaidoo, 2015).

2.1.2 Learning to Programme

Learning to programme is a complicated task which requires the learner to configure a programming environment, comprehend the syntax and additionally consider how to approach the problem at hand.

Learning to programme can be made more accessible through the use of a visual programming language. Scratch is a popular block based programming environment introduced by (Maloney et al., 2004). Students using Scratch have showed sustained engagement with programming and independently discovered programming concepts (Maloney et al., 2008; Franklin et al., 2013).

The successes of the Scratch system could be due to motivation, simplification and support, which are identified by (Kelleher and Pausch, 2005) as key to making programming more accessible. These findings relate closely to the cognitive school of learning where the students are identified as needing to remain within the zone of proximal development.

Learning to programme is considered from different psychological perspectives by (Hoc and Nguyen-Xuan, 1990; Gomes and Mendes, 2007) who concur that successful learning is the result of feedback from practicing in an environment which assists in the identification and recovery from errors.

Programming can also be simplified by tutorial style interactions that allow students to learn to programme syntax languages by overcoming the barriers for students who are not familiar with the syntax of a language. This behaviourist approach however can be found to provide too much assistance to students completing problems (Deek and McHugh, 1998).

2.1.3 Summary

The existing literature on learning theory and learning to programme provide many requirements for the TuringLab system to achieve an optimal environment in which students can undertake programming challenges.

The behaviourist school of learning defines that the system needs to clearly define learning outcomes and assess against completion of these learning outcomes. This means that the system needs to display the learning outcomes from undertaking a particular challenge. Additionally this also means that a challenge needs to be easily verifiable that it is correct to affirm that students have completed their learning outcomes.

To correspond with recommendations from the cognitivist perspectives, the system needs to again display learning outcomes but additionally suggest challenge difficulty. This allows students to select challenges of appropriate difficulty and for which they have already been introduced to the concepts.

A key requirement identified both from cognitive learning theory and pragmatic experience of learning to programme is the need to make programming accessible. With reference to syntax programming, this outlines the need to assist students in recovering from errors. Additionally, this indicates that challenges need to create meaningful outputs.

From a constructivist theory of learning the requirement to create an active learning environment is presented. This means that students should be able to decide the challenges which they undertake and are supported by interactive instructions in the process. Therefore, challenges need to have some reference material attached to them in order to prevent excessive work for a teacher.

The requirements identified from the existing literature define the overall specifications of the TuringLab system. These requirements along with the aim of TuringLab to work alongside ordinary teaching provide the basis for reviewing and understand existing systems.

2.2 Existing Systems

In this section, existing systems are critically assessed to determine how best to achieve the requirements identified for TuringLab in Section 2.1. The systems considered generally provide teachers with an adaptable environment where custom programming challenges can be set and progress reviewed, however often do not provide an environment which is engaging for children.

Figure 1 shows a broad overview of the existing systems. The dashed circles depict the four major functionalities identified in the existing systems (related to TuringLab). These systems are arranged in a Venn diagram to display each system's relation to a functionality. The labels applied to the functionalities are explained below.

Learning Environment relates to any system

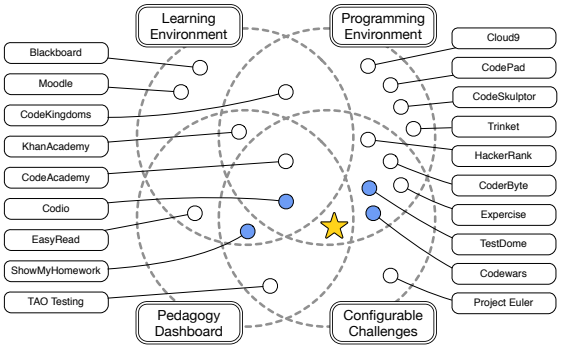

Figure 1: Venn diagram showing related existing systems.

which includes teaching material, be that predefined or teacher created. These systems in general have a large amount of content and are intended for learners to remain within the infrastructure for learning, exercises and testing. Although TuringLab does not aim to include a learning environment, systems with learning environments are important to be considered in order to determine how content is arranged and categorised with the aim of allowing students to select the correct learning material determined by their cognitive ability and learning objectives.

Programming Environment encompasses systems which have the facility to write and run code. Most systems with this feature also have the ability to test code against predefined or user specified test cases. Systems with advanced programming environments are important to review in order to consider how to create meaningful and complex challenges while still supporting the student.

Pedagogy Dashboard defines any system which has a dashboard through which teachers can view metrics on their pupils. These metrics relate to how a user is interacting with TuringLab and what learning progressions are being made through these interactions. To allow teachers to formatively assess their students, a feature rich pedagogy dashboard is important. Allowing teachers to differentiate students means they can be kept at the correct cognitive level.

Configurable Challenges specifies systems which give users the ability to add customised problem definitions. Problem definitions contain a rich text description of the problem and, more often than not, testing criteria to define when a problem is complete. Reviewing approaches to defining and assessing challenges means that TuringLab can have elegant challenge specifications which are not too time consuming for teachers to create.

The star shown in Figure 1 identifies the aim of TuringLab. The shaded circles indicate systems which are of interest to the development of TuringLab. These systems include ShowMyHomework, TestDome, Codewars and Codio. A brief analysis of the beneficial features of each of these systems is outlined below.

ShowMyHomework[1] is a website and app which allows teachers to set homework for students to complete and parents to oversee. Aside from the lack of a programming environment, it is very close to the presented system. Teachers can track the progression of students in completing homework assignments and the system can select problems at an appropriate difficulty level for the students.

TestDome[2] is a web-based tool for assessing individuals on their programming ability. Experienced developers are paid to add authentic challenges of specified complexity. Companies pay to screen candidates using a custom selection of challenges in particular programming languages. Candidates complete programming challenges in a fully featured text editor and test their results.

Codewars[3] is a web-based environment for writing, solving and discussing programming challenges. The site philosophy is collaborative and the aim is for all members to better their programming ability. Their challenges are of varying complexity and topics and are well written. The collaborative aspects of Codewars make it quite engaging.

Codio[4] is a web-based integrated development environment for students to learn and practice programming. Students can enroll in courses tailored to teach different programming languages. Teachers can create custom programming environments and challenges. Student progress and engagement can be analysed through a teacher dashboard. The adaptability of the programming environment and inclusion of a teacher dashboard make this system very versatile and beneficial to teachers.

2.2.1 Summary

Existing systems were reviewed to understand how other systems have implemented requirements for a programming environment. In general the existing systems considered have a common downside in that they do not provide an environment which is fun and engaging for children to use. As the success of the Scratch system is due to children maintaining engagement, a key requirement for TuringLab is to create this engaging environment. This engagement is critical to allow children to discover computing concepts through challenge and exploration.

[1] https://showmyhomework.co.uk/
[2] http://www.testdome.com/
[3] http://www.codewars.com/
[4] https://codio.com/

3 DESIGN

This section details the design process that TuringLab went through to fulfil requirements and features identified in Section 2. A participatory design process was used during development to ensure children enjoyed using the TuringLab environment. This process began with requirement gathering from children at volunteer-run programming clubs before development was commenced and this feedback continued throughout the software development life-cycle.

The current system design is the result of a number of iterations using the lessons from previous stages of the design process to improve TuringLab. Feedback collected from children during the design process was in the form of open ended questions to achieve the greatest insight into how children use the software.

1. **Functional Mockup**
   A functional mockup was created as early as possible. This allowed children to access a number of programming challenges and view the solutions. This functionality was provided on a single page, with the specification of the challenge included in the comments section of the code. Children got confused by the cluttered interface and did not naturally read the comments contained within the code to understand what was required in the challenge.
2. **Alpha Design**
   The alpha design split the selection of challenges and completion of challenges onto separate pages and displayed the challenge specification separately to the code. This version allowed children to write code on their browser and run or test the code on the server. The system was not suitably responsive for children who often ran the challenges many times during completion - both the lack of graphical output and inclusion of code testing made the children lose engagement very quickly. The benefits of testing from an educator's perspective did not outweigh the lack of engagement from children.
3. **Beta Design**
   The beta design moved code execution from the server to the client. This meant TuringLab was far more responsive for the users and outputs to running code could be displayed graphically. In previous trials only a selection of children used TuringLab and they quickly lost engagement. In this version all the children learning Python used TuringLab and maintained engagement and enjoyed the outputs they produced. The graphical outputs seemed to assist children in gaining familiarity with the concepts of iteration and selection. Running code on the client meant challenges were of reduced complexity and were not tested; however, this was worthwhile to maintain child engagement.
4. **Current Design**
   The current design takes much the same form as the beta design; however, provides an interface for teachers to overview the progress which children were making when completing challenges. This means that teachers can see errors and solutions in real time. The volunteers assisting with the coding sessions found these features very beneficial in order to provide the proper scaffolding; however, the greatest benefit was to encourage discussion following a session. Children were asked to run their favorite solutions and then took it in turns to explain how they had achieved the output which was displayed on the large screen.

Each stage of the design process led to large changes to the implementation; however, all changes were driven from the design requirements acquired from using TuringLab with the children. The current design is very capable for children to learn to programme; however, there remain a number of improvements which are detailed in Section 7.

Figure 2 shows the current student interface and how the different screens are linked to one another. This interface is comprised of three main screens: selecting a group, selecting a challenge within a group and completing a challenge. When completing a challenge, children are able to access a help page, run the challenge code and access different versions of the code. The student interface initially allowed children more functionality in how they went about selecting challenges, be that through a group or just from a list of all challenges; however, this was found to be an unnecessary complication for the children.

Figure 3 shows the screen where a student selects which challenge to work on. The content of a challenge are identified by badges where the badges represent programming concepts. The number of badges a challenge has, suggests the difficulty of a challenge. These badges are designed to encourage children to use their meta-cognition when selecting a challenge.

Figure 4 shows the screen when a child has run a challenge. Here the code editor is to the left and provides full syntax highlighting. The output of the code is on the right and provides a graphical output in the form of the Python Turtle[5] graphics and a terminal output which displays print messages and allows

[5]https://docs.python.org/3.4/library/turtle.html

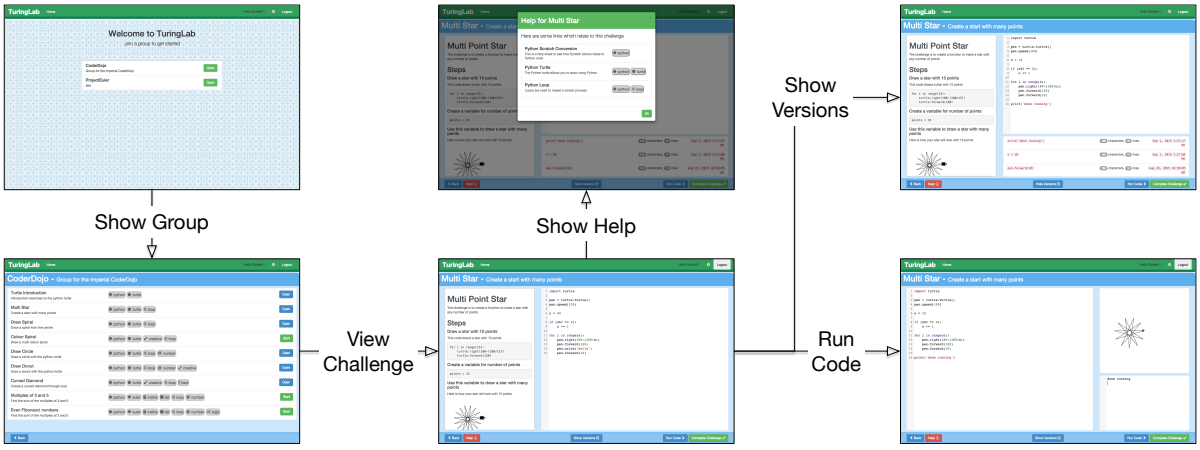

Figure 2: Student interface user interaction flow.

Figure 3: Student challenge selection screen.

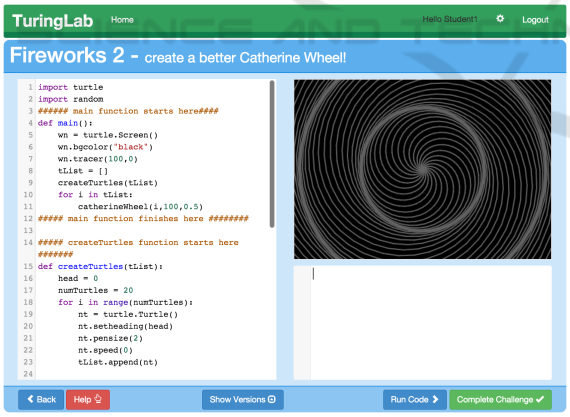

Figure 4: Student running code screen.

children to provide standard input.

4 IMPLEMENTATION

The implementation of TuringLab was completed incrementally alongside the design process. The overall architecture of TuringLab can be seen in Figure 5. This architecture was selected to allow the web server to respond quickly to requests and the worker server to enable computation heavy analysis.

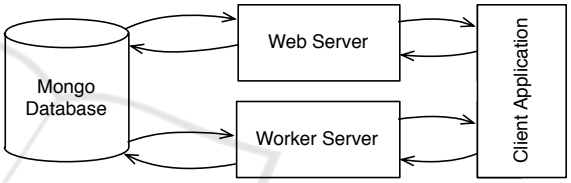

Figure 5: System Architecture.

4.1 Mongo Database

The database is used to store all the persistent data in TuringLab. MongoDB[6] is a non-relational database which was selected due to its flexible storage of JSON data. This allowed the data structure to adapt as and when new system features were added. The database is accessible from both the web and worker server, which allows the functionality of each service to be decoupled while still allowing data to be shared.

4.2 Web Server

The web server serves static files, which make up the client application and provides direct access to the database through RESTful (Representational State Transfer) endpoints. The web server is written in NodeJS[7] using an Express[8] server to provide the routing. The web server can easily be distributed to allow for high load as it maintains no state for client requests.

[6]https://www.mongodb.com/
[7]https://nodejs.org/en/
[8]http://expressjs.com/

4.3 Worker Server

The worker server is written in Python and is primarily responsible for debugging code files. Given the worker server has access to the same data-store as the main web server, it can also be used to process the data relating to a child's progress through challenges. As both functionalities require extensive execution time, having this run on a worker server does not affect the responsiveness of the web server.

4.4 Client Application

The client application renders the pages which users interact with based on data provided by the web server. This application runs completely on the client's web browser and is written in AngularJS[9], a popular JavaScript web framework. The client application has access to both the web server and the worker server through simple endpoints. The database models can be accessed from the client application through the web server and certain computations run on the worker server.

The client application provides a rich code editor through use of the CodeMirror[10] library. Python code is run on the browser through using the Skulpt[11] library, which compiles Python code to JavaScript. This allows the results from the running code to be quickly displayed to users. The outputs to running code can be displayed graphically when using the Python Turtle.

5 EVALUATION

The TuringLab system was designed as an active learning environment which was enjoyable for children to use when learning syntax programing. The success of the system is primary determined by the enjoyment which they gained from using the system. The benefit of the system on the activity of learning to programme can be determined by considering a number of requirements identified in Section 2. Challenge selection must allow students to select challenges with appropriate learning outcomes, of suitable difficulty and with a clear definition of completeness. The system must assist in recovery from errors and allow for meaningful output from the executed code. Furthermore, the system should facilitate reference material to assist children with errors before asking the teacher for help.

[9]https://angularjs.org/

[10]https://codemirror.net/

[11]http://www.skulpt.org/

The system was designed from a user centered approach; children attended volunteer-led programming clubs and made use of TuringLab to learn Python. At a number of the programing clubs children also learnt Scratch [12]. When learning Scratch, an interface was provided to give the children well-scaffolded projects to work on, which recorded their progress in a similar way to the TuringLab system.

From discussions with children during the process of developing TuringLab, it was clear that having a graphical output for their programmes in the form of Python Turtle graphics was very important. This meant children could easily detect errors in their solution through visual comparison and the output to their code was something that they could be proud of. Furthermore, the visual output allowed non technical volunteers to both reason about issues in a child's code and see merit in complex creations.

Feedback on the TuringLab system was collected over the course of 8 programming clubs. These programming clubs were funded by the Department of Computing at Imperial College London and therefore were free to attend and did not require attendees to bring any computer equipment. The first 3 sessions were Python only mixed sessions which had a largely male demographic, the following 5 sessions allowed students to select between Scratch and Python programming but was for girls only.

Over the 5 sessions a total of 122 responses were collected from attendees. Given that a number of individuals were repeat attendees this equated to 70 unique respondents completing feedback on the programming clubs; 21 of these unique respondents reviewed the Scratch system and 49 reviewed the Python system.

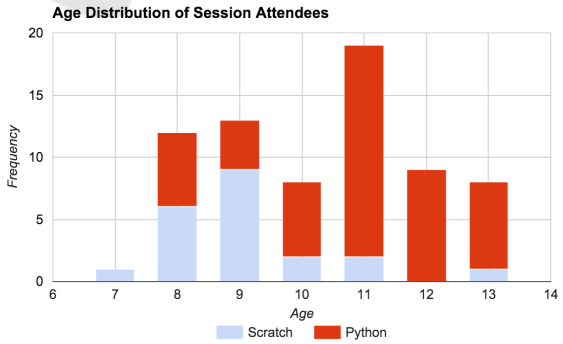

Figure 6: Comparison of programming language choice for attendees of different ages.

Figure 6 shows the number of children learning Python using TuringLab and the number of children

[12]https://scratch.mit.edu/

learning Scratch. It can be seen that generally older children chose to learn Python whereas younger children favoured Scratch. However, children of all ages learnt both Python and Scratch suggesting both languages are appealing to children between the ages of 7 to 13.

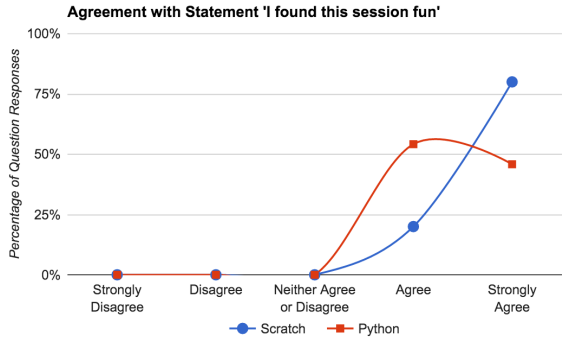

Figure 7: Comparison of agreement with statement 'I found this session fun'.

Throughout the project a focus of the system has been to create an environment which children enjoy using and which maintains their engagement. To find out the level of enjoyment and therefore potential long term engagement, children were questioned on how enjoyable the session was. Figure 7 shows that children found the session slightly more enjoyable when using Scratch than Python but overall enjoyed learning both.

In Figure 7, it can be seen that the programing clubs facilitate a high level of engagement from children; however, as the children were not found to enjoy using the Python system as much as the Scratch system it shows that there are potential areas of improvement. The issues with TuringLab may be due to not having the correct amount of scaffolding to support children through the challenges or that younger children find syntax programming more challenging than block programming.

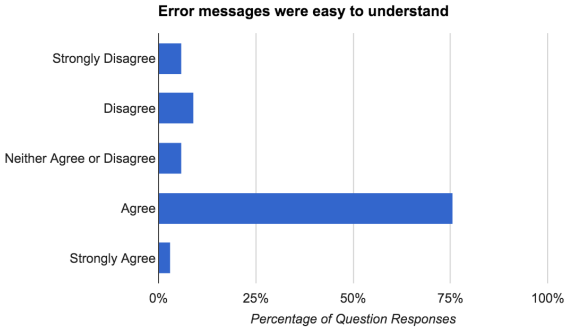

Figure 8: Student agreement with statement 'Error messages were easy to understand'.

Figure 8 shows the extent to which children agree that the error messages were easy to understand. From this figure it can be seen that there are a number of respondents that did not find the error messages easy to understand. Given that children are finding error messages hard to understand, a potential aspect of the system which needs to be improved is to provide more readable error messages. This improvement would not necessarily reduce the error rate but it would allow children to recover from errors more easily through having the correct scaffolding available. Enabling children to recover from errors using feedback from TuringLab, without the need to seek help from a teacher is vital to encourage an active learning environment, where children can work at their own pace on challenges of their choice.

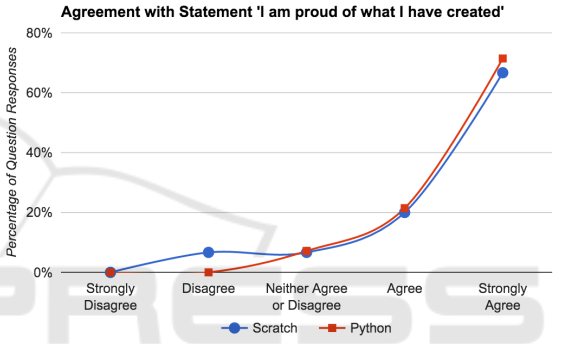

Figure 9: Comparison of agreement with statement 'I am proud of what I have created'.

It can be seen in Figure 9 that in general children are equally proud of what they have created in both Scratch and Python. This result is surprising, since children are able to create games and animation using Scratch whereas with Python they create far less appealing projects (in the time available during the programming clubs), but which are also more challenging. This result suggests that children are proud of completing challenging projects which feel more difficult or meaningful.

Figure 10 shows how easy children found it to select a suitable challenge to undertake. This is important, as being able to select a suitable challenge implies that children are able to make use of the badges which are displayed on challenges. These badges are used to display the learning content of a challenge along with implying the difficulty of the underlying challenge. Given that children are generally in strong agreement, this would suggest badges are an appropriate mechanism to display challenge content. However there remain a number of children who were not able to find a suitable challenge, therefore the badges and challenges require further description.

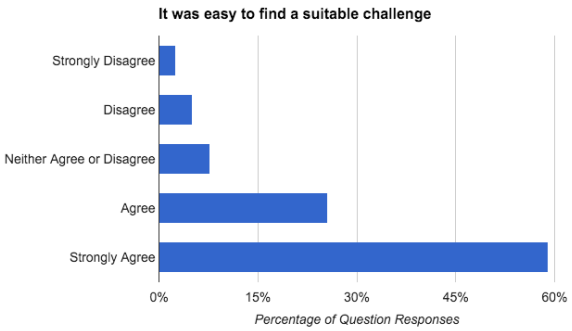

Figure 10: Student agreement with statement 'It was easy to find a suitable challenge'.

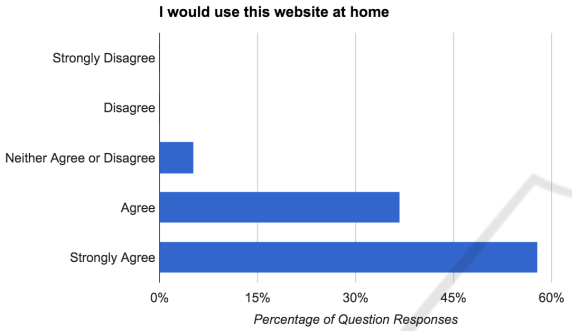

Figure 11: Student agreement with statement 'I would use this website at home'.

Figure 11 shows the responses of the children when asked if they would use TuringLab at home. It can be seen that the overwhelmingly positive response suggests TuringLab provides a programming environment which is fun and engaging for children to use.

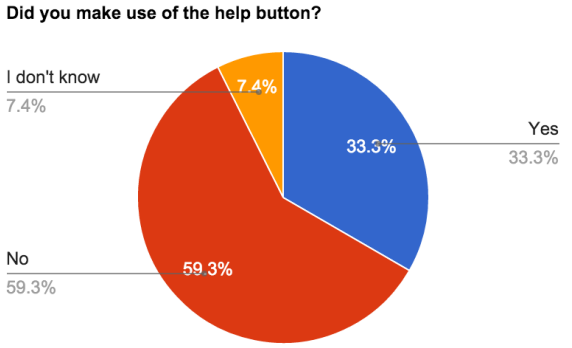

Figure 12: Pie chart showing the number of students who made use of the help interface.

Figure 12 depicts the proportion of respondents who made use of the help button within the TuringLab system. The help button brings up resources related to the current challenge. Resources are automatically added based on the badges which are selected for a particular challenge. Given that a third of children used the help button it suggests that the system nurtures an active learning environment for some of the children. This feedback does not however suggest if the help resources were useful in resolving students issues.

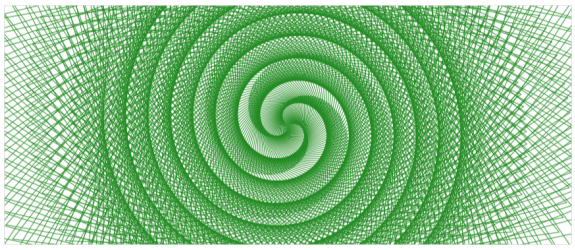

Figure 13: Example of children work created on TuringLab.

Figure 13 shows a spiral created by a child during one of the sessions. The child was very proud of the work completed which could be seen from the passion with which they explained the spiral at the end of the class. The teacher display mode facilitates children discussing their favorite creations during the session.

6 CONCLUSIONS

This paper presented TuringLab, an environment to assist teachers as part of their standard teaching in delivering the practical elements of the computing curriculum through the setting and assessing of programming exercises. TuringLab was designed as a pedagogical system for children to complete programming exercises in a challenge based programming environment which provides support and assistance during their completion of challenges. Teachers are able to perform the core tasks required to assign engaging challenges to collections of children.

Overall the feedback for TuringLab is positive. It is clear children enjoyed using TuringLab and a number of children have returned both to TuringLab outside of the programming sessions and to the sessions the following weeks. Given that the children have agreed the challenges are fun and that they are proud of the results it is clear that the system is providing children with interestingly hard yet suitably enjoyable challenges. This is likely to have been afforded by the ability of TuringLab to display a graphical output from the children code being run.

TuringLab provides a solid and verified base on which to build further. This is a system which children have engaged with and enjoyed using over the course of a number of programming clubs. There are however a number of potential enhancements to TuringLab (detailed in Section 7), which would al-

low TuringLab to provide greater assistance to children when they are seen to struggle and hence facilitate more complex challenges.

7 FUTURE WORK

The errors displayed to children are the default errors from the Skulpt library, following which a separate service analyses the code to ascertain more information on the source of the error. This service currently only checks for correct parenthesis matching, however children have often had difficulties with indentation and capitalisation when writing code. In some cases this can just result in a bad input error from Skulpt which is far from understandable for an individual not experienced with the language.

The system does not allow for multiple files within a challenge, this means any helper functions which are needed when completing the challenge need to be provided in the same file in which the children write their solution. This can often distract children and lead to cluttered files. In future iterations, allowing multiple files per challenge will mean challenges can provide advanced helper functions without confusing or distracting children.

The system currently allows for the use of the Python Turtle. Skulpt could be used to import external libraries for use by children but these have to be individually configured for use with Skulpt. This is the approach taken by the Trinket[13] team to allow components of numpy[14] and matplotlib[15] to be accessible through Skulpt.

To unlock more advanced functionality within the client, external web APIs could be abstracted in Python code. This would allow children to use the API functionality from the code in the browser as if they were calling a simple blocking function in Python. This approach would not give the children practice in using advanced Python libraries but it would allow them to use advanced functionality with potentially interesting results.

REFERENCES

Adler, M. J. (1982). *The Paideia proposal: An education manifesto*. Macmillan.

Alzaghoul, A. F. (2012). The Implication of the Learning Theories on Implementing e-learning Courses. *The Research Bulletin of Jordan ACM*, 11(11):27–30.

Anderson, T. (2008). *The Theory and Practice of Online Learning*. Athabasca University Press.

Arkorful, V. and Abaidoo, N. (2015). The role of e-learning, advantages and disadvantages of its adoption in higher education. *International Journal of Instructional Technology and Distance Learning*, page 29.

Black, P., Harrison, C., Lee, C., Marshall, B., and Wiliam, D. (2003). *Assessment For Learning: Putting it into Practice*. McGraw-Hill Education (UK).

Black, P. and Wiliam, D. (1998). *Inside the black box: Raising standards through classroom assessment*. Granada Learning.

Brophy, J. (1987). Synthesis of Research on Strategies for Motivating Students to Learn. *Educational Leadership*, 5(2):40–48.

Brophy, J. (1999). Toward a model of the value aspects of motivation in education: Developing appreciation for.. *Educational psychologist*, 34(2):75–85.

Cellan-Jones, R. (2014). A computing revolution in schools. http://www.bbc.co.uk/news/technology-29010511. visited 2015-06-04.

Computing at School (2015). Computing Teachers Call For More Training Amidst Concerns Pupils Know More Than Them. http://www.computingatschool.org.uk/index.php?id=current-news&post=quickstart-launch-2. visited 2015-06-04.

Cooper, D. and Adams, K. (2007). *Talk about assessment: Strategies and tools to improve learning*. Thomson/Nelson.

Deek, F. P. and McHugh, J. a. (1998). A Survey and Critical Analysis of Tools for Learning Programming. *Computer Science Education*, 8(2):130–178.

Department for Education (2013). National curriculum in England: computing programmes of study. https://www.gov.uk/government/publications/national-curriculum-in-england-computing-programmes-of-study. visited 2015-06-04.

EdSurge (2015). Teaching Kids to Code. https://www.edsurge.com/research/guides/teaching-kids-to-code. visited 2015-10-14.

Flanagan, J. (2013). 10 places in Britain where you can learn how to write computer code. http://www.theguardian.com/technology/2013/oct/14/learn-how-to-code. visited 2015-08-27.

Franklin, D., Conrad, P., Boe, B., and Nilsen, K. (2013). Assessment of computer science learning in a scratch-based outreach program. *Proceeding of the 44th . . . .*

Gomes, A. and Mendes, A. J. N. (2007). Learning to program-difficulties and solutions. *International Conference on Engineering Education*, pages 1–5.

Gordon, N. (2014). Flexible Pedagogies: technology-enhanced learning.

Gove, M. (2014). Michael Gove speaks about computing and education technology. https://www.gov.uk/government/speeches/michael-gove-speaks-about-computing-and-education-technology. visited 2015-08-27.

[13] https://trinket.io/python

[14] http://www.numpy.org/

[15] http://matplotlib.org/

Grabinger, R. S. and Dunlap, J. C. (1995). Rich environments for active learning: a definition. *Research in Learning Technology*, 3(2).

Hoc, J.-M. and Nguyen-Xuan, A. (1990). Language semantics, mental models and analogy. *Psychology of programming*, 10:139–156.

Huffaker, D. a. and Calvert, S. L. (2003). the New Science of Learning: Active Learning, Metacognition, and Transfer of Knowledge in E-Learning Applications. *Journal of Educational Computing Research*, 29(3):325–334.

Jones, S. P. (2015). Code to Joy. *Times Education Supplement*.

Juriševič, M. (2010). Creativity in the Zone of Proximal Motivational Development. *Facilitating effective student learning through teacher research and innovation*, pages 415–429.

Kelleher, C. and Pausch, R. (2005). Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys (CSUR)*.

Keller, J. M. and Suzuki, K. (1988). Use of the ARCS Motivation Model in courseware design.

Knight, J. (2008). The Assessment for Learning Strategy. Technical report, Depeartment for children, schools and families.

Maloney, J., Burd, L., and Kafai, Y. (2004). Scratch: a sneak preview. *Creating, Connecting and Collaborating through Computing*.

Maloney, J., Peppler, K., and Kafai, Y. (2008). Programming by choice: urban youth learning programming with scratch. *ACM SIGCSE* . . . .

Mödritscher, F. (2006). e-Learning Theories in Practice : A Comparison of three. *Science And Technology*, 0(0):3–18.

OurICT (2015). Ten Resources for Teaching Computer Science. http://www.ourict.co.uk/teaching-computer-science/. visited 2015-08-22.

Pearlman, B. (2009). Making 21st Century Schools Creating Learner-Centered Schoolplaces / Workplaces for a New Culture of Students at Work. *Educational Technology*, 49(5):14–19.

Stergioulas, L. K. and Drenoyianni, H. (2011). *Pursuing Digital Literacy in Compulsory Education*, volume 43 of *New Literacies and Digital Epistemologies*. Peter Lang Publishing Inc.

Twigg, C. A. (2002). *Quality, cost and access: The case for redesign*. Prentice-Hall, New Jersey.

Vygotsky, L. (1978). *Mind in society: The development of higher psychological processes*. Harvard University Press.

Walser, N. (2011). *Spotlight on Technology in Education*. Number 7 in Harvard Education Letter Spotlight. Harvard Educational Publishing Group.

Wood, D. (1998). *How children think and learn: The social context of cognitive development*. Oxford: Blackwell.

Zheng, S., Rosson, M. B., Shih, P. C., and Carroll, J. M. (2015). Understanding Student Motivation, Behaviors and Perceptions in MOOCs. In *ACM Conference on Computer Supported Cooperative Work & Social Computing*.