

Assessing Computational Thinking Process using a Multiple Evaluation Approach

Yasemin Allsop
University College London, UK
y.allso@ucl.ac.uk

ABSTRACT

This study explored the ways that the Computational Thinking (CT) process can be evaluated in a classroom environment. Thirty Children aged 10 - 11 years, from a primary school in London took part in a game-making project using the Scratch and Alice 2.4 applications for eight months. For the focus of this specific paper, data from participant observations, informal conversations, problem-solving sheets, semi-structured interviews and children's completed games were used to make sense of elements of the computational thinking process and approaches to evaluate these elements in a computer game design context. The discussions around what CT consists, highlighted the complex structure of computational thinking and the interaction between the elements of artificial intelligence (AI), computer, cognitive, learning and psychological sciences. This also emphasized the role of metacognition in the Computational Thinking process. These arguments illustrated that it is not possible to evaluate Computational Thinking using only programming constructs, as CT process provides opportunities for developing many other skills and concepts. Therefore, a multiple evaluation approach should be adopted to illustrate the full learning scope of the Computational Thinking Process. Using the support of literature review and the findings of the data analysis I proposed a multiple approach evaluation model where 'computational concepts', 'metacognitive practices', and 'learning behaviours' were discussed as the main elements of the CT process. Additionally, in order to investigate these dimensions within a game-making context, computer game design was also included in this evaluation model.

KEYWORDS

Game programming, Elementary school, Scratch, Alice, Assessment, Computational Thinking, Metacognition, coding

1 Introduction

The recent inclusion of programming concepts in the primary school curriculums of many countries including England raised an interest in teaching children how to code, as a result educators started to explore the methods that they can use for engaging learners with programming activities. Some researchers have suggested that computer game design is a fun and effective way of introducing programming concepts [1, 2], however; the empirical evidence to support this is very limited.

Alongside coding, computational thinking also has increasingly gained attention, which highlighted an important issue: the readiness of teachers for planning, teaching and assessing children's learning when they code. Teachers might be able to describe the terminology relating to computational process or teach coding using lesson plans and instruction sheets that are available online without mastering the concepts, however, this doesn't warrant that they would be able to recognise Computational Thinking (CT) skills and concepts when they evaluate children's work in different subjects. In the following sections I will investigate what computational thinking is and approaches to evaluating CT skills.

2 Defining Computational Thinking

There is no common definition of computational thinking and its characteristics. Papert did not provide a definition for Computational Thinking (CT), however, he came up with the idea of CT by focusing on the procedural thinking that children develop through programming in a LOGO environment [3,4]. Wing [5] pioneered this idea and emphasized that CT is not just about coding, it is a skill set for understanding human behaviour using fundamental concepts from computer science. In 2010 she reintroduced the term computational thinking as "the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent" [6].

A number of studies highlighted CT as a cognitive process [7,8] and some described it as a problem-solving approach [6,9]. The role of metacognition in the CT process is also emphasized [10,11] and how CT is different to other ways of thinking has been explained by focusing on the automation of information [12,13] for computer systems to execute repetitive tasks efficiently. This highlights the link between CT and Artificial Intelligence (AI). AI can be defined as "the ability of the computer systems to learn, think and perform tasks that require complex decision-making" [14]. The core of this repetitive task automation is algorithms and abstractions, which is also key element of CT [15].

Aho explained CT as the thought processes involved in formulating problems so that "their solutions can be represented as computational steps and algorithms" [12, p.832]. From a psychological perspective forming a mental representation of a problem (formulating problems), planning and choosing appropriate strategies for a solution (Formulating solutions), checking

for errors (evaluating) and debugging them, thinking about how to improve work (monitoring) are components of metacognition [16]. Simply described as ‘thinking-about-thinking’ [17], Claxton [18] explains metacognition as a way of supporting people to manage their minds more productively, which enables them to use their resources more effectively.

Lu discussed CT as a “full set of mental tools necessary to effectively use computing to solve complex human problems” [13]. The effective allocation of these mental tools for completing a task requires one’s own knowledge of these tools and knowing how to use them for executing a task, namely metacognition. Several researchers also highlighted the relationship between metacognition and Computational thinking. Resnick [19] suggested that constructive learning environments where learners are given opportunities to design solutions iteratively and reflect on their own learning processes is required for facilitating learning of computational thinking skills. This was also supported by Papert [3, p.25] who argued that creating programs encouraged learners to be more aware of the strategies they used for debugging problems and think about ways of improving them. In a recent paper Kafai and Burke discussed the benefits of constructionist game making and emphasised the learning beyond coding. They claimed that a constructionist game making space supports children to think about their own thinking and learning namely “reflection or metacognition” [11, p.10].

The Barefoot Computing Programme [20] considered computational thinking from the concepts and approaches aspect. They listed tinkering, creating, debugging, persevering, and working collaboratively as the main approaches that pupils apply and develop during the CT process. Brennan and Resnick [10] discussed questioning, connecting and expressing under the term computational perspectives. In a model for computational thinking created by the Somerset e-Learning & information management team [21], making mistakes, perseverance, imagination and collaboration were listed as attitudes that pupils use during the computational thinking process. Furthermore, other studies found that whilst working on their games, pupils had opportunities to apply and develop skills such as collaboration, creativity, communication, critical thinking, tinkering, persevering [22-24]. All these approaches, perspectives and attitudes can be described as learning behaviours since these are the strategies for promoting behaviours that are ‘necessary for learning’ [25, p.53]. Powell and Tod [26] suggested that Learning behaviours reflect pupils’ social, emotional and cognitive development and depends on their prior learning experiences; therefore, the patterns of development would be varied for each pupil.

As discussed above, Computational Thinking process offers wider scope than just learning of programming constructs. Informed by the discussion about what CT consists of using relevant literature, I propose the following definition of computational thinking, which:

- is a cognitive process
- is regulated by metacognitive practices
- involves the application of a series of computational concepts
- includes the utilization of learning behaviours
- aims to design solutions to problems that are susceptible to automation

3 Recent literature on assessing CT Process

In recent years several studies have been conducted to measure the CT skills that children develop when creating their own computer games [27, 10, 28]. Werner, Denner and Campe [27] proposed a three-level assessment model called Game Computational Sophistication (GCS) for measuring children’s computational learning in an ‘Alice’ programming environment. The first level is about coding blocks that are crucial for programming or making games. At the second level, students use coding blocks to create patterns. The next level, a combination of programming constructs and patterns, namely, ‘game mechanics’. Although the clear structure of their game mechanics and pattern model makes it easier for investigating evidence of CT skills in the games that were created by the children, it is crucial to remember that computational thinking includes both concepts and approaches [20]. Therefore, other methods should be used alongside this model to provide a more detailed overview of the CT skills that learners develop when making computer games. It is difficult to gain an insight of the that challenges they faced and / or how they managed their thinking and learning process by just looking at the programming constructions that they used.

Brennan and Resnick [10] proposed a model for measuring CT skills when children develop games in a ‘Scratch’ programming environment. They suggested a framework with three dimensions; computational concepts, computational practices and computational perspectives. Computational concepts include sequences, loops, parallelism, events, conditionals, operators, and data. Computational thinking practices involves focusing on the thinking and learning process, how they planned their games, how they solved problems, which strategies they used and so forth. Although the computational thinking practices were defined in relation to a Scratch Programming environment, it can be applied to activities when using other gaming applications. This dimension can be seen as a metacognition of coding as it involves metacognitive skills such as planning, evaluating, modifying, monitoring, reflecting in other words thinking about thinking.

The final dimension of Brennan and Resnick’s model for measuring CT skills is called the computational perspective and is all about children’s “understandings of themselves, their relationships with others, and the technological world around them” [10]. They suggested three approaches to assess computational concepts, practices, and perspectives; project analysis, artifact-based interviews and design scenarios, all with strengths and limitations. One interesting point about Brennan and Resnick’s model is, computational practices are similar to the metacognitive process which involves focusing on the thinking and learning process, how they planned their games, how they solved problems, which strategies they used and so forth.

4 Towards a multi evaluation approach for assessing CT

The definition of CT I shared in section two and the literature review on assessing Computational thinking skills highlights the complex structure of computational thinking and the interaction between the elements of Artificial Intelligence (AI), computer, cognitive, learning and psychological sciences whilst providing a foundation for defining the multiple aspects that the evaluation of CT skills should include. This multiple means of assessment approach was also supported by Brennan and Resnick [10] who highlighted the necessity of focusing on the process that children go through rather than only their codes. Similarly, Grover [29] after reviewing different assessment approaches to CT suggested that the ‘systems of assessment’ discussed by Conley & Darling-Hammond [30] would provide a more comprehensive view of children’s learning of CT skills. I agree with this view, as it is not possible to use one single method to evaluate the interaction between the elements of computer, cognitive, learning and psychological sciences. Adopting a multiple means of assessment approach would not only provide more in-depth information about children’s understandings of computational concepts, but also gather evidence of children’s individual skills development, especially during pair coding activities. In this context I use the term ‘assessment’ to represent the evaluation of children’s learning rather than as an educational assessment tool.

The terms from my definition of Computational thinking, which represent the interaction between different sciences, were used for evaluating learners’ CT skills from three aspects: ‘computational concepts’, ‘metacognitive practices’, and ‘learning behaviours’. In order to investigate these dimensions within a game-making context, computer game design was also included in the evaluation model. The model is semi-flexible; as it is possible to exclude and replace the game design dimension when evaluating CT in a different context to computer game design for example app development. Figure 1 presents the overview of the Multiple Evaluation Approach to CT skills in a computer game design context.

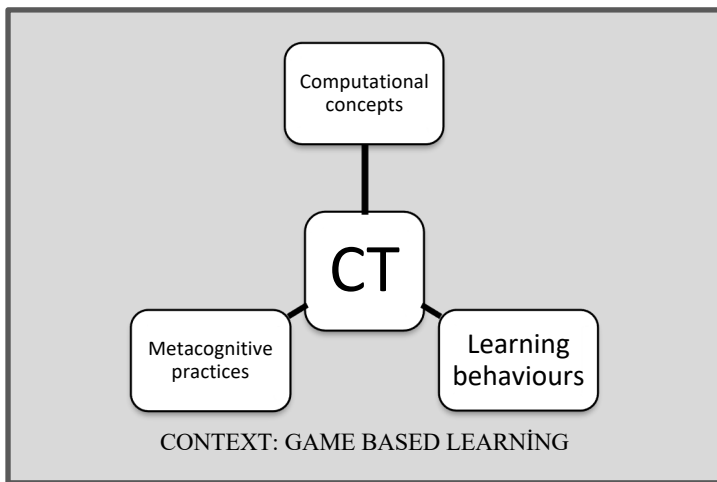


Figure 1: Multiple Evaluation Approach to CT skills in a computer game design context

Computational Concepts

Computational concepts refer to the programming constructs that are commonly used for completing tasks in programming environments such as sequences, loops, conditionals, and variables. Motivated by Werner and colleagues [27] Game Computational Sophistication Analysis Procedure and the Computational thinking concepts by Brennan and Resnick [10], I included sequences, loops, events, parallelism, conditionals, operators, variables and abstraction as the programming constructs that represent computational concepts in this study.

Learning Behaviours

I explain learning behaviour as the strategies, approaches and habits that have been exhibited by children whilst working on a task, which promotes learning. Powell and Tod listed engagement, collaboration, participation, communication, motivation, independent activity, responsibility, disaffection and problems as the main behaviours for learning [26]. In a DfES White Paper about Education and Skills for 14-19 years old pupils enquiry, creative thinking, information processing, reasoning and evaluation were included as learning behaviours [31]. Although he did not mention the term ‘Learning behaviour’, Claxton’s

[32] theories around building learning power seem to focus on similar attributes that schools should focus in order to help children learn. These attributes are; resilience, resourcefulness, reflectiveness, and reciprocity.

Metacognitive practices

I define metacognition as a skills set which enables one to deploy and manage his or her cognitive resources effectively to regulate his or her thinking and learning. Stenberg [33] listed planning, evaluating, monitoring problem-solving activities and allocating cognitive resources appropriately as the main abilities for managing the metacognitive process. Flavell [17] described exploring, setting goals, organizing, planning, self-questioning, choosing and applying, monitoring and managing thinking as metacognitive skills. A number of studies also described planning, monitoring and evaluation as the main metacognitive skills [34-36]. Metacognitive practices can be seen as the trigger and executive control for regulating cognitive activities, which includes planning, evaluation and monitoring.

The core of metacognitive practices is the conversational exchanges that take place between 'others' and 'self'. Vygotsky [37] also mentioned the role of private and inner speech (conversation with self) and Social speech (Conversation with others) in self-regulation by stating that language is not only used for communication, but also for self-regulation through planning and monitoring. Likewise, some other studies also described this conversational exchange with self and others as an instrument for managing planning, monitoring, thinking and learning processes [37-38].

Game Mechanics

The main elements of games were usually explained by game mechanics, which defines how players interact with the game. Lundgren and Björk [39, p.4] explained game mechanics as the rules that players need to employ when they interact with a game. Vincent and colleagues described the mechanics of a game as "the basic components out of which the game is built: the materials, rules, explicit goals, basic moves, and control options available to the players". Hunicke and colleagues [40] noted that mechanics involve actions, behaviours and control mechanisms. This complex structure of game mechanics makes it difficult to create a set of evaluation criteria for pupil created games. Weise [41] suggests that writing game mechanics in a verb form; basically, as actions that are accomplished within the limits of game rules is a technique that can be useful for creating a framework. Werner and colleagues [27] used a similar technique to assess game mechanics in computer games that were created by children using Alice 2. They listed actions such as collecting, shooting, racing, guessing, hitting moving objects, exploration as game mechanics. Additionally, they included puzzles, hidden objects, navigation, levels and avoidance in game mechanics.

5 Methodology

This paper is based on a longitudinal PhD study with a wider focus area and it specifically aims to answer the question 'What is the best approach for assessing CT skills through game making?'. This study was reviewed by Goldsmiths, University of London Institutional Review Board and later also approved by the Manchester Metropolitan University ethics committee. For the purpose of this specific study ethnography as a longitudinal and qualitative method was used to examine the children's Computational Thinking process when making computer games. Ethnography not only enabled me to blend different data collection methods but also provided me with a rich written account of the children's learning process. By being in the classroom for a long period of time to observe what is going on, what children are saying, doing and why; I was able to monitor the changes in children's reasoning over a long period, which provided me with detailed, in depth findings, understanding, and a very personal level of experience.

As Denzin and Lincoln [42] suggested that the use of methodological triangulation can increase the validity of studies, therefore, for the focus of this paper data from participant observations, informal conversations, children's problem solving sheets, in-depth interviews and children's completed games were used to make sense of elements of computational thinking process and approaches to evaluate these elements in computer game design context.

Participant observations and informal conversations

Field notes were collected by examining the language children used for their 'self' explanations and group discussions, the gestures, the context of their relations with teacher, peers and technology in their classroom setting. During observations, I used informal or conversational interviews, which allowed me to discuss issues that arose or question the children on significant events as they occurred. Because they were not formal interviews, this helped the children to feel more comfortable and open in giving their answers. I used a pen and A5 size notebook to keep a record of both my observations and informal conversations.

Semi-structured interviews

Semi-structured interviews were used to unfold the student's 'deeper self' and collect 'authentic data' [43]. I interviewed ten focus children at the end of the project individually. The interviews were recorded using a sound recorder and transcribed. Each interview was around 10-15 minutes long.

Journals and Problem solving sheets

Although I provided children with journals to record their activities, many children found it difficult to decide what to record in their journals; therefore, I decided to provide them with a template to record their activities when solving problems. This problem sheet included questions and a space for the children to draw their problem in case they did not want to write down their comments. Appendix E shows an example of completed problem solving sheet.

Children's completed games

The children's completed games were studied to examine the computational thinking process that the children went through whilst they were working on their games. In total, I analysed 18 games that were created using the Scratch application and 15 games that were created using Alice 2.4. I mainly looked at the programming constructs and game mechanics that the children had used in their code scripts.

The data was collected for a period of eight months, in a primary classroom setting in London. The school is larger than the average primary school with approximately 900 students. The school has a high proportion of pupils from minority ethnic backgrounds. There is also a high proportion of pupils who speak English as an additional language. The proportions of pupils with special educational needs are above average. The students are mainly coming from disadvantaged backgrounds, therefore the proportion of the children eligible for free school meals is high.

Children aged 10-11 (Year six) were included in this study. At the first stage thirty children in a Year six class (16 boys, 14 girls) aged 10-11 used Scratch between September and January for four months, once a week, for an hour as part of their weekly computing session. This totalled 14 hours of coding. All fourteen of the girls decided to work in pairs, only 12 of the boys chose to work in pairs and four of them opted to work alone. At the second stage thirty children in a Year six class (16 boys, 14 girls) aged 10-11 used an Alice application for making games between January and April for four months, once a week, for an hour as part of their weekly computing session. This totalled 14 hours of coding. During the Alice game making project all of the children decided to work in pairs. One reason for this might be that after the introduction session many students mentioned how challenging they found the Alice application in comparison to the Scratch program that they had used in the previous term.

Ethics

Although the school had a generic form signed during the child's registration to allow the school to study the children's work, because this was part of a PhD study, I created an information sheet and a consent form in line with BERA [44] ethical guidelines. A permission letter regarding observing children working on their game designs; interviewing them; studying their written learning log, photos, videos and audio was prepared. I listed the data collection activities on the consent form which included; taking part in the study, being observed by the teacher, keeping a journal, participating in audio recorded interviews, taking part in video recorded group discussions. 14 out of 30 parents returned their consent forms agreeing to permit their children to take part in all data collection activities. All ten of the focus children were selected from this list.

I had the completed parental consent forms, however, I decided to negotiate ongoing consent with the children. As Flewitt [45] suggested, it is difficult to regulate ongoing consent, however, I wanted to make sure that the children were happy to take part in all of the stages of this study. Before the interviews, I asked the children if they were happy to be included.

6 Data analysis

In order to evaluate the thinking and learning process that children go through when making games, their problem solving sheets, completed games, interview data and field notes were analysed for patterns of CT skills and characteristics. Although collecting data using many different methods provided me with a better understanding of emerging themes, I found it challenging to analyse the vast amount of data that I had collected without developing a system. In order to tackle this issue, as suggested by Miles and Huberman [46], I reduced the data into manageable units through constant comparison and coding to answer each focus questions separately. I used the three levels of coding that were proposed by Strauss and Corbin [47]; open, axial and selective coding. I started analysing the data by comparing the findings from different data collection methods. At this stage some categories and themes started to emerge naturally. Next, I started to link the categories by asking questions and making comparisons. At the final stage, I focused on the key themes and investigated similarities and relationships between the categories, which helped me to refine and develop the concepts that reflect the behaviours of the participants.

During the 'open coding process' I read and annotated each interview script, field notes, children's problem solving sheets and their completed game designs. I highlighted and labeled concepts on each data text (Appendix A). For the content analysis all the data that was collected from different sources were coded using a constant comparative method in order to identify the repeating patterns and emerging themes. As I continued to code, themes and categories that were different from previous studies emerged. For example when analyzing the data from the interviews, problem solving sheets and the field notes;

‘Talking to self’, ‘Talking in mind, in the head’ for managing learning were repeated many times. Therefore, conversational exchanges were included as an additional category.

Alongside methodological triangulation, qualitative directed content analysis was used for making sense of the data where the initial coding started with the previous research findings [48]. Drawing concepts from the previous studies was very useful especially at the beginning of the data analysis [49]. For example, prior to forming a framework for evaluating children’s CT skills when making games, Computational Sophistication Framework [25] and Scratch assessment package [10] were explored in depth to understand how children’s learning of CT skills could be evaluated during game making activities.

6.1 Computational Concepts

Motivated by Werner and colleagues [25] Game Computational Sophistication Analysis Procedure and the Computational thinking concepts by Brennan and Resnicks [10], I included sequences, loops, events, parallelism, conditionals, operators, variables and abstraction as the programming constructs that represent computational concepts in this study.

In order to make the analysis process more efficient I created a guide where I described each programming construct, and then showed an example of what it looked like in both the Scratch and Alice environments (Appendix B). This is useful for educators, as it would help them to identify the programming constructs in children’s games. I then used this guide to create case studies; where I exemplified the programming constructs in two games that were created by children using the Scratch and Alice applications. Appendices C and D show the example case studies of games in Scratch and Alice. Finally, I analysed the overall data using a three-step analysis process, as described by Werner and colleagues [25] for analyzing each game that the children created using both Scratch and Alice application:

- In the first stage the code was analyzed to identify the programming constructs that were used.
- The games were then played to check if the programming constructs were executed correctly.
- The final step looked to define whether the code was either built-in or created by the student.

6.1.1 Computational Concepts in Scratch

Out of the 30 children 24 of them worked in pairs and six of them worked alone, therefore 18 games were included in the data analysis. Although at first I used Dr. Scratch (an online application for assessing children’s Scratch projects) created by Moreno-León and Robles [50] to analyse the students’ games, I then manually evaluated each game as I wanted to be able to use specific examples from the students’ game script to explain how well they were able to use the programming constructs rather than the generic ones that were presented by this tool. This was very useful when giving individual feedback to the students by using examples their own game scripts. The programming constructions were graded using a simple point system similar to the Dr. Scratch assessment tool. If the programming constructs were not used within the game the student received zero points, if they were used in a simple form the students received one point and if a more sophisticated programming construct was used, the students received two points.

If the game included all of the programming constructions at a sophisticated level it would have received 16 points in total, which is 100%. Once I had graded each game, I calculated the mean value to define the average level of use for each programming construct in the Scratch environment. This was very useful for identifying the computational concepts that the children were struggling with and those concepts that they were using efficiently. Table 1 displays the mean score for each programming construct.

As it is illustrated on Table 1 the mean score for using an abstraction construct was zero, meaning that no one used custom built functions. The mean score for sequences was 97.2% with a standard deviation of 0.2; this shows that almost all the games included sequences at a complex level. The use of operators was low, at only 33.3% with a standard deviation of 0.7. This means that either the children did not know how to use operators, or it wasn’t necessary for their game design. Parallelism and conditionals were used confidently with a mean score being 77.8% (SD: 0.5) and 75% (SD: 0.6). Loops were used in 16 games and events were used in 17 games. Variables such as timer, score and lives were used by 52.8 % (SD: 0.8) of the games.

	Mean score N: 18	Percentage %	Standard Deviation
Sequences	1.9	97.2	0.2
Loops	1.3	66.7	0.7
Parallelism	1.6	77.8	0.5
Conditionals	1.5	75.0	0.6
Operators	0.7	33.3	0.7
Variables	1.1	52.8	0.8
Events	1.3	66.7	0.6
Abstractions	0.0	0.0	0.0

Table 1: Mean scores for programming constructs

Although pair-coding made it difficult to compare individual students' understanding of CT concepts, single sex pairing made it possible for some gender-based comparisons. There were 14 girls who worked in pairs to create their animations and games. The analysis of the games that were created by the seven pairs of girls showed that, they were all able to use sequences very well, however, they struggled with the application of variables. Only one group of girls was able to use variables to create a game with both a score and a timer. The remainder created simple animations without any variables. The students' prior experience of game playing might have had an impact on this, however, there was no data to support this claim as the students were not asked about their previous experiences of either playing or making games. The gender based comparison for other programming structures did not have any significant patterns, both boys and girls groups had some issues with using operators, conditionals and loops.

The comparison of the six games that were created by individual children along with twelve pair coded games provided me with some information about the impact of pair coding on the students' ability to use programming constructs.

	Pair coded Games			Independently coded games		
	Mean score N: 12	%	Standard Deviation	Mean score N:6	%	Standard Deviation
Sequences	1.9	95.8	0.3	2.0	100.0	0.0
Loops	1.5	75.0	0.6	1.0	50.0	0.6
Parallelism	1.6	79.2	0.5	1.5	75.0	0.5
Conditionals	1.7	83.3	0.5	1.2	58.3	0.7
Operators	0.8	37.5	0.7	0.5	25.0	0.5
Variables	1.3	66.7	0.7	0.5	25.0	0.5
Events	1.5	75.0	0.5	1.0	50.0	0.6
Abstractions	0.0	0.0	0.0	0.0	0.0	0.0

Table 2: Comparing pair coded games and independently created games

As displayed on Table 2, the use of variables in the games that were created by the children who worked alone was only 25%, which rose to 66.7% in the games that were coded by pairs. The students who worked collaboratively used loops, conditionals and events constructs by almost 25% more than the games that were created independently. It is very difficult to describe the factors that might have had an impact on the level of using programming constructions. Students' prior experiences of these constructs and coding with Scratch programs, opportunities for talk and discussion, accessing the work from home can be listed as some of these, however there is no data to support or explain this statement. The use of sequences and parallelism were of a very similar level for both pair coded and independently created games.

6.1.2 Computational Concepts in Alice

Werner et al. [15] listed Alice programming constructs in four levels of difficulty. They placed basic constructions for creating sequences and simple event handlers in level 1; use of built-in functions and more sophisticated event handlers in level two; creating methods, variables, if/else, loop and while statement in level three; parameters, student created functions, list variables, nested if/else statements, more sophisticated sequence and parallelism constructions in level 4. Using their analysis scheme I created a simple rubric that would help me to evaluate the games that students created using Alice application. I first listed the blocks under one and two points to differentiate their difficulty level. This can be seen as level one and two. Then I mapped each block and action in Alice to a programming construction to make it easier to compare the level of use. As discussed by Werner et al [15], both if/else and while statements are based on simple Boolean expressions, therefore they have been listed

under the 1 points section. If the student did not use the programming construct, they received zero points, if they used the simple constructions as listed in Table 3 they received one point, if they used more advanced programming constructs they received two points. In total I analysed fifteen games that were created using Alice application.

	0 point	1 point	2 points
Sequences	Programming construct hasn't been used	Do in order	For all in order
Loops		Loop statement While statement	Nested loop
Parallelism		Do together	For all together
Conditionals		If/Else statement	Nested If/Else
Operators		Mathematical expressions	Relational and Logical Operators
Variables		Non-list variables	List variables
Events		Event with single action	Event with multiple actions
Abstractions		Built in methods Simple event handlers	Student created Methods Sophisticated event handlers
		Built-in functions	Student created functions

Table 3: Scoring system for Alice programming constructs

Similar to the Scratch games analysis if a student used all of the programming constructions at the sophisticated level they would receive 16 points in total, which is 100%. I calculated the mean value for each programming construct to define the average use in the Alice environment. This helped me to see the computational concepts that the children were able to apply at a confident level. Table 4 displays the mean score for each programming construct.

As clearly illustrated by table 4 the mean score for the sequences construct was 100%, meaning everyone was able to create a set of instructions to program the behaviour of an object at a sophisticated level. 80% of the games included more than one event for an object that would happen at the same time (parallelism). The mean score for the uses of operators was 60% with a standard variation of 0.4, showing that the students used this construct more than they used when coding with Scratch. Abstraction was used in 43.3% of the games and loops in 46.7%. Students used conditionals well, as 75% of the games included this programming construct. Variables had the lowest mean score (10%) where only one student used this construction at a complex level, where he created a timer and a score. Another student attempted to create a variable but there was an error so it did not work correctly.

	Mean score N: 15	Percentage %	Standard Variation
Sequences	2.0	100.0	0.0
Loops	0.9	46.7	0.7
Parallelism	1.6	80.0	0.5
Conditionals	1.5	73.3	0.5
Operators	1.2	60.0	0.4
Variables	0.2	10.0	0.5
Events	1.3	66.7	0.5
Abstractions	0.9	43.3	0.6

Table 4: Mean scores for programming constructs

Gender comparison of the games that created using Alice shows that both boys and girls were able to use sequences in their games. Boys were more successful at using almost all the programming constructions including loops, parallelism, conditionals and events. Variables seem to be problematic for both boys and girls, as girls' game did not include any variables, and only one pair of boys were able to include variable in their game. Another pair also tried to use it but didn't work properly. Abstractions were used in 50% of the boys games, in comparison to 35% in girls games. The data used for gender comparison of the games can be seen in table 5.

	Boys			Girls		
	Mean score N: 8	%	Standard Deviation	Mean score N:6	%	Standard Deviation
Sequences	2.0	100	0.0	2.0	100.0	0.0
Loops	1.3	62.5	0.7	0.6	28.6	0.5
Parallelism	1.9	93.75	0.3	1.3	64.3	0.5
Conditionals	1.8	87.5	0.4	1.1	57.1	0.3
Operators	1.3	62.5	0.4	1.1	57.1	0.3
Variables	0.4	18.75	0.7	0.0	0.0	0.0
Events	1.5	75	0.5	1.1	57.1	0.3
Abstractions	1.0	50	0.7	0.7	35.7	0.5

Table 5: gender comparison of children’s games created using Alice

6.2 Metacognitive practices

As discussed previously planning, monitoring, evaluation, self-questioning, choosing and applying were listed as metacognitive skills by many studies [15, 31-34]. Kafai [51, 52] studied children’s game design activities and suggested that game design is a context for children to practice and develop transferable skills such as planning and problem solving.

Planning

This study also found that planning was a skill used by all of the children mainly at the beginning of the activity. The children used different methods and styles to plan their games when using both the Scratch and Storytelling Alice applications. Some children preferred only drawings as a tool to communicate their ideas, some used both text and drawings and a few used only text to present their thoughts. Appendix F shows some examples of the children’s planning sheets. Only four children decided to use the planning sheet that I had prepared for them (Appendix G).

The data from the semi-structured interviews also suggests that coding activities for making games helped the children to use planning skills more often for other activities as well. One student reported this *“I used be like, let’s do this, but never planned for anything. But game design made me to do stuff freely, like independent. And then suddenly my thinking has changed. Now I plan everything out”*.

Monitoring and evaluation

The children were able to identify their errors and explain how they solved them, which can be seen as both a ‘monitoring’ and an ‘evaluating’ skill. During the interviews one student explained that they found their mistake, which was *‘naming the variable wrong and corrected it’*. She also recorded in her journal that both she and her partner learned to *“sort stuff out and how to correct their mistakes in game design”*. Another student reported their problem solving activity as *“We couldn’t change the colour of the tombstone, we went on You Tube and followed the instruction, and we had to put down a lot of methods”*. This shows that this student was able to evaluate his game to identify the error and then exploring ways of finding a solution. This shows that this student was able to evaluate his game to identify the error and then exploring ways of finding a solution. Another focus child recorded their problem solving activity as *“We test it and it doesn’t work...I then figured out that the lollipop must be behind, so we add another net which is behind it. BINGO! It works.”* All these activities can be seen as demonstrations of self-regulating, because had they stopped when they couldn’t solve a problem, the learning would also have stopped. Rather they used different strategies to help themselves to continue to look for solutions for their problems. This involved, testing, evaluating, communicating, working collaboratively, making decisions, experimenting with ideas and selecting strategies.

The participant observations showed that the children constantly tested their game design and checked their codes for errors when it did not work how they expected. They deleted lines of code or added new code blocks to make their designs work; In other words debugged their errors. This constant evaluation activity, continued throughout the design, not just as the learners developed their games but also at the end as a final check up activity. The children also helped each other to evaluate their games by giving one another feedback. They walked around the room, played with their friends’ games and gave verbal feedback. Some students analyzed their game and provided feedback to their ‘self’. For instance one of the focus children looked at his design at the end then started to touch the screen and talk to his ‘self’. He said *‘This works (pointing at a car, good. The sound ‘pop’ doesn’t go with this. Maybe I could use (he clicked on the sound tab and explored different sound effects) this one (chomp)’*.

Conversational exchanges

Another findings of this study showed that children used language as an instrument, in different forms of conversation to make decisions, evaluate and regulate their activities. When the students were asked to record what they asked / talked / thought to themselves on their problem solving sheets, the students shared the questions that they were asked in order to solve a problem, make a prediction, or make a decision before they took an action. For example one student reported this as *"I asked and talked about how are we going to work out to move the robot and the space men"*. Another one wrote, *"I thought to myself how am I going to make the witch move around the screen?"* There were more questions written in this section by children asking about how to complete a specific task and also more broad questions to check if they were doing things correctly.

In his problem-solving sheet Child K stated that he had asked and talked to himself about 'how are we going to work out how to move the robot and the space man'. This is significant as he used 'we' instead of 'I'. This can be seen as his acknowledgement of his on and off interaction with his partner in his thought process. He also added that he discussed with his friend how to make the robot say 'boo'. Other children in their problem solving sheets also reported this type of focused dialogue with their partner on a specific problem, question or task.

The data from the interviews also demonstrated that the conversations with both their 'self' and 'others' was taking place when the children were regulating their problem solving activities. One child expressed this as solving a problem in mind and said *"Yeah, I ask questions to my partner and I think, I talk to my brain. Can I do this, it is like my brain says yes and give me the answer, think like solving in my mind"*. Another child stated that he would use dialog with his 'self' to check and evaluate his design before sharing it with others. He articulated this as *"Before let people see, I would ask myself 'are you sure it is alright?' When I was making the robot fighting game. I wanted to see, I talked to myself how would make it more interesting and more detailed. To make it more like movement, maybe add voice. I just say in my mind 'What shall I do to fix this?' if something is wrong. This makes you think if you ask and repeat"*.

During the semi-structured interviews Child M, aged 9 remarked "First I write a piece of script like how the steps, then I think to myself, how can I make it better? I try to understand or sometimes decide by asking myself. I usually do this when I don't understand what I am doing, when I just check it or revise it I talk to myself. In other lessons I do it, but I don't do it a lot. Not as often as game design."

It is apparent from this comment that Child M used language in a form of a thought for asking himself for help when he doesn't understand something or is making decision. It is interesting that he is aware of a self-talk function and he states that he uses it in other lessons but not as often. By asking how he could make it better he is activating the thought process for evaluating and planning. Child H, aged 10 also stated similar comments but this time with a justification, suggesting that listening to teacher in other lessons limits the use of self-talk. This might pose a question as to whether or not too much 'teacher talk' would have an impact on both children's private and inner speech.

"I ask myself shall I do that, shall I do this, trying to make a decision. It kind of helps me to make sense of things. I do it in other lessons too but not that much. Because you have to listen what teacher is saying".

Child H's purpose of using self-talk is comparable to that of Child M's; both mentioned making decisions and making sense of things. They noted that they are both aware of self-talk as a function and pointed out that they use it more often during game making.

6.3 Learning behaviours

The findings of this study also found that during their game making project, whilst working on their design and programming scripts, pupils worked collaboratively, thought creatively and critically, debugged errors, tinkered with ideas and communicated these ideas using different modes of conversational exchanges.

Collaboration

The field notes presented that although most of the children usually worked directly with their partner, on many occasions they also walked around the classroom to look at others work, where they either made suggestions or got some ideas for their own games. There were some that asked for help from others. There was a constant discussion between the pairs and other game designers, which enabled them to evaluate and reflect on their own work and to re-organise their ideas. This collaborative approach to game making had motivational power by providing support for the children from their peers when they needed it.

Perseverance

One other interesting learning behaviour shown by many pupils was perseverance. When they identified their script error, they tried different solutions to debug it. Sometimes this was a simple action, but sometimes they had to spend a very long time trying different options until they found how to make it work. The records from the participant observations of the children working on their games showed that some children did give up when faced with a challenge and some persevered, so that they

did not stop until they had found a solution to their problem. The following record from participant observations shows the interaction, which took place between three children during one of the game making sessions demonstrate this:

Child T and Child A were making a game together using Alice application. They had a problem. They couldn't stop spacemen becoming smaller as they got closer to the spaceship. Child T looked for information on Google but she couldn't find anything. It was apparent from her facial expressions and body gestures that she was getting very annoyed. Child A suggested that they should re-start their work, however, this did not solve their problem. At this point Child T started to become disengaged. She did not answer Child A's questions, she just looked at the screen. Child A called Child C to help them. He wanted to quickly fix the problem but Child T wasn't happy with this. She asked Child C to show how to stop the spacemen becoming smaller. Rather than just doing it, she then took the mouse from Child C and completed the task.

Child T was disengaged with the activity when she couldn't solve a problem. Her partner suggested that they should re-write the script, but she didn't show any interest in this. It was only when another child offered to help them with their problem that she engaged with the game making activity again. This shows the importance of, allowing children to face challenges, facilitating discussions; group interactions and pairing when making games that will motivate children continue learning.

Communication

The field notes from the observation of the children have shown that they constantly communicated with their friends. They talked about their storylines, characters, backgrounds, code errors and rules. They discussed their solutions and actions to problems before they put them into practice. They gave feedback to each other and made suggestions for improving their work. This shows that communication was a core part of their game making. Many children mentioned asking for help during their interviews. For example Child K suggested that he first tried to solve a problem himself but if he couldn't, he asked his friend. He explained this as *"What I do is, if I have a problem, like the character is in the wrong place, I will try to move to different place by changing the code, but if it doesn't work I will ask my friend.* Likewise children's problem solving sheets also had records of children talking to their friends about problems they faced whilst making games.

Debugging

Identifying and debugging coding errors or solving problems related to the design of the games were also observed during game making. The children tested their codes frequently to check that it worked. When it did not they tried to identify the problem; sometimes alone, sometimes with others and then designed solutions. The written records of informal conversations with the children highlighted that the children evaluated their work and checked for errors during game making more than for any other lesson. One of the children reported the reason for this is as *"you can find your mistake very easy when making games because if the code is wrong, game doesn't work"*.

Creativity & Tinkering

I did not ask children about how creative their games were during this study; however, it is evident from their completed games and field notes that they were experimenting with ideas that involved decision-making, critical thinking, problem solving, designing solutions which can all be seen as part of creativity. The task of character and background designs also provided the children with an opportunity to develop creativity skills, as this would allow them to express their own ideas using technology. During the interviews children were asked about what they learned by making games. Child K replied as *"my imagination"*. When I asked how, he answered as *"Because like it expresses your imagination different points and it tells you, you can come up with good things to say"*. There were mention of pair work and how this impacts on creativity. Child M reported this *"we learned how to use our imagination and how to cooperate because we worked in a pair. If you work by yourself you may not do much, because two heads are better than one"*. This shows that some children may not be able to express their ideas alone and might need the input or support of a friend. The link between using imagination, creativity and brainpower was proclaimed by Child S who suggested that *"Having a wide imagination means, thinking a lot harder, harder you think, more intelligent and more creative you get"*.

Problem solving

During the interviews when asked what do they think they learned by making games, every single child mentioned 'problem solving' as one of the skills that they developed. Child S replied this as *"I think I learned imagination and a lots of skills. Designing, imagination, problem solving. I learned to do it by myself, not always many people around to help"*. Child K stated that during game making sessions when he had a problem, he would try new things to see if he could make it work or think about adding more things to improve it.

Both participant observations and children's problem solving sheets from this study showed that constant problem solving was at the core of the game making activities. The children's problem solving sheets where they recorded some of the challenges that they faced and how they solved them provided us with more detailed information about examples of problems that they had. When analyzing children's problem solving sheets this was also visible in the problems that the children recorded when making their games. In Scratch the children solved problems related to script such as creating a variable but they also had different problems; such as making sound work, locating sound files, duplicating a character, finding a costume. When

designing a game using Alice the children’s problems were mainly writing the code to make an object do something e.g. How to add a score, moving an object by itself, how to add a timer (variable), moving a left arm or right leg (robot).

6.4 Game mechanics

At the beginning of the study we had a class discussion about ‘what makes a game, a game’. The common answers that were given by students were: games is something you can play, it has rules, you get rewards, it has score, it needs timer, you get points if you win, you lose lives if you don’t play well, you have different levels, many games have stories, it has goals. My notes of informal conversations with children during the class discussions illustrates that they distinguish a game from animation mainly by its playability function. One student explained this as “*You play with games, but animation, you just watch them, don’t you?*”

I analysed 18 games that were created using the Scratch application and 15 games that were created using Alice I used Werner and colleagues’ study to examine game mechanics in children’s games. I added other visible actions and elements that represent game mechanics for each game and then looked for repeated patterns first in Scratch games, then Alice games. Finally, I compared the results of the two separate analysis to provide an insight into game mechanics that were used by the children whilst making their games and how this relates to their learning, especially the development of computational thinking skills. Table 6 shows the actions and elements that the children included in their games.

Analysis of the eighteen Scratch games that were created by the children showed that timed challenge and score/point were the most commonly used mechanics as these were included in eleven games. Levels and lives mechanics were used only in two games. Seven games contained the challenge of avoiding objects by controlling a sprite using either mouse or arrow keys on the keyboard. Three games included an action of clicking on the objects that were appearing and hiding on the screen in order to get points. Two games were basic racing games where two players were expected to each move a character to a finishing line, however this task did not have any visible forms of reward. One game included speed for objects (apples) dropping from top, which was a range of random numbers. When asked what the reason was for using speed, Child B replied as “*This made my game more challenging because players have to be ready for speed that changes all the time*”. There was one game contained which a mathematics quiz, for players to enter the correct answer into a box. This analysis illustrates that children used different mechanics in their game, which they thought had an impact on the level of players’ engagement with their games.

The analysis of the fifteen children’s games created using Alice showed different results from those of the Scratch games. The most commonly used game mechanic was moving objects where children created events to control the objects using arrow keys. Only one game included a timed challenge and score. Two games tried to create a boat racing game but had issues with creating the score and timer. One student created a simple race game by moving objects using arrow keys to a specified position. Most of the games, which the children created, using Alice, were in a format of animation rather than a game. When the children were asked why they did not include mechanics in their games, they mentioned how difficult it was for them to create a timer and score using Alice When I provided them with an instruction sheet for a game with a timer and score, they were then able to add these to their games. This showed that they needed more input and practice for creating games using Alice coding environment.

Mechanic	Description
Timer	Player is given a time limit to complete the task
Levels	Player is allowed to move to different stages when completing a challenge or reaching a target
Avoiding objects	Player avoids objects to complete a task. This is done sometimes by controlling the object using a mouse or arrow keys on the keyboard.
Clicking objects	Player is given points/ reward when clicking the objects
Moving objects	Player moves the objects by using mouse or keys on keyboard
Racing	Player moves objects to the finishing line. This sometimes involves time limit.
Guessing	Player completes a quiz by typing answers
Catching objects	Player controls an object to catch other falling objects. This is done using mouse or keys on the keyboard.
Points / Score	Player receives points or score for completing tasks.
Lives	Player is given number of lives for completing a task. In many games when the player runs out of lives, the game stops.
Speed	Player is given number of speed options for different level of challenge, engagement and interaction with the game.

Table 6: The actions and elements that children included in their games

7 Conclusions

This paper explored what Computational thinking constitutes and the ways to best evaluate it using both the support of literature and the data collected from this study. After a thorough literature review I proposed a definition of computational thinking which highlighted the interaction between computation and the elements of AI, computer, cognitive, learning and psychological sciences. This was also used for creating a framework for evaluating different aspects of the Computational Thinking Process, which can be listed as ‘computational concepts’, ‘metacognitive practices’, ‘learning behaviours’ and ‘Context’, in this study, computer game design. We can conclude that a multiple evaluation approach should be adopted to illustrate the full learning scope of the Computational Thinking Process.

Evaluation of children’s completed games exhibited that although a few students found using variables and abstraction challenging, children were able to use programming constructs including sequences, loops, parallelism, conditionals, operators and events. The gender based comparisons showed that there were differences between the girls’ and boys’ use of programming constructs both in Alice and Scratch. In the Scratch environment all except two girls created animations without using variables. There were no significant differences in the use of other programming constructs. In the Alice environment variables were found challenging by both girls and boys and only 35% of the girls’ games included abstractions in comparison to 50% of the boys’ games.

Data analysis of the children’s problem solving sheets, observation records, informal conversations and semi-structured interviews presented that planning, monitoring, and evaluation were the main metacognitive skills that the children applied and developed through metacognitive practices when making computer games. Monitoring through constant testing and evaluation was also evident in all of the children’s work, showing that metacognitive practices were used for controlling and regulating programming activities. Furthermore, the findings of this study showed that the children used different modes of conversation to make decisions, evaluate and regulate their activities.

The findings of the data also displayed that learning behaviours such as collaboration, communication, persevering, problem solving and creativity were visible whilst children were coding their games.

This was a small-scale study, limited to a class in a primary school in London, UK. Larger scale studies in various locations should be conducted to explore the challenges around how each aspects of Computational thinking process can be assessed consistently by teachers on a daily basis using simple evaluation tools. For example, the creation of tasks and ‘I can’ statements for each aspect of the Computational Thinking process that can be used by class teachers, alongside a guide will help them to recognise programming constructs, which would be very beneficial.

References

- [1] A. R. Basawapatna, K. H. Koh & A. Repenning, Using scalable game design to teach computer science from middle school to graduate school, In: Proceedings of the fifteenth annual conference on Innovation and technology in computer science education, ACM, New York, NY, USA, 2010, 224-228. <https://doi.org/10.1145/1822090.1822154>
- [2] J. Denner, S. Bean & J. Martinez, The girl game company: Engaging Latina girls in information technology. *Afterschool Matters*. 8, (2009), 26-35.
- [3] S. Papert, *Mindstorms: Children, Computers, and Powerful Ideas*, Basic Books, New York, 1980.
- [4] S. Papert, *Situating constructionism*. In I. Harel & S. Papert (Eds.). *Constructionism*. Ablex Publishing Corporation, Norwood, NJ, 1991.
- [5] J.M. Wing, Computational thinking, *Commun. ACM* 49 (2006) 33–35. <http://dx.doi.org/10.1145/1118178.1118215>.
- [6] J. Cuny, L. Snyder, and J. M. Wing, Demystifying computational thinking for non-computer scientists, 2010. <https://www.cs.cmu.edu/~CompThink/resources/TheLinkWing.pdf> (Accessed June 5, 2018).
- [7] C. Selby & J. Woollard. Refining an understanding of computational thinking, 2014. <https://eprints.soton.ac.uk/372410/> (Accessed June 5, 2018).
- [8] W. Sung, J. Ahn, S. M. Kai, A. Choi, J. B. Black. Incorporating Touch-Based Tablets into Classroom Activities: Fostering Children's Computational Thinking through iPad Integrated Instruction. In *Handbook of Research on Mobile Learning in Contemporary Classrooms*. IGI Global, (2016), pp.378–406. <https://doi.org/10.4018/978-1-5225-0251-7.ch019>
- [9] ISTE. *Computational Thinking in K–12 Education leadership toolkit*. 2011.
- [10] K. Brennan & M. Resnick. New frameworks for studying and assessing the development of computational thinking. In: *Proceedings of the annual meeting of the American Educational Research Association*. Vancouver, Canada, 2012, pp.1-25.
- [11] Y. B Kafai & Q. Burke, Constructionist gaming: Understanding the benefits of making games for learning. *Educational psychologist* 50, 4, (2015), 313-334.
- [12] A. V. Aho, Computation and computational thinking. *The Computer Journal* 55, 7, (2012), 832-835.
- [13] J.J. Lu, G.H.L. Fletcher, Thinking about computational thinking, in: *Proceedings of the 40th ACM Technical Symposium on Computer Science Education*, ACM, New York, NY, USA, (2009), 26–264.

- [14] G. Gadanidis, Artificial intelligence, computational thinking, and mathematics education. *The International Journal of Information and Learning Technology*, 34 (2), (2017), 133-139.
- [15] A. Yadav, C. Mayfield, N. Zhou, S. Hambrusch, and Korb. J.T., Computational thinking in elementary and secondary teacher education, *ACM Transactions on Computing Education*, 14 (1), (2014), 5
- [16] J. E. Davidson, R. Deuser & R. J. Sternberg, The role of metacognition in problem solving. In J. Metcalfe & A.P. Shimamura (Eds.), *Metacognition*. Cambridge, MA: MIT Press, 1994.
- [17] J.H. Flavell, Metacognition and cognitive monitoring: A new area of cognitive–developmental inquiry. *American psychologist*, 34(10), (1979), 906.
- [18] G. Claxton, *Wise up: The challenge of lifelong learning*. Bloomsbury, London, 1999.
- [19] M. Resnick, All I really need to know (about creative thinking) I learned (by studying how children learn) in kindergarten. In: *Proceedings of the 6th ACM SIGCHI conference on Creativity & cognition*, ACM, 2007, 1-6.
- [20] Barefoot, C. A. S. *Computational Thinking*, 2014. <https://barefootcas.org.uk/barefoot-primary-computing-resources/concepts/> (Accessed June 1, 2018).
- [21] Somerset e-Learning & information management team. *The Computational Thinker*, 2014. <https://slp.somerset.org.uk/sites/edtech/SitePages/Secondary%20Learning/KS3%20and%20the%20Computing%20Curriculum.aspx> (Accessed June 1, 2018).
- [22] M. Akcaoglu, Learning problem-solving through making games at the game design and learning summer program. *Educational Technology Research and Development*, 62, (2014) 583–600.
- [23] J. Denner & L. Werner, Computer programming in middle school: How pairs respond to challenges. *Journal of Educational Computing Research*, 37, 2, (2007), 131-150.
- [24] S. Bermingham, N. Charlier, FM. Dagnino, J. Duggan, J. Earp, K. Kiili, E. Luts, L. Stock, N. Whitton, Approaches to collaborative game making for fostering 21st century skills. In: *proceedings of the 7th European Conference on Games-Based Learning*. Academic Conferences Limited, 2013.
- [25] S. Ellis & J. Tod, *Behaviour for learning: proactive approaches to behaviour management*. Routledge, Oxon, UK, 2013.
- [26] S. Powell & J. Tod, A systematic review of how theories explain learning behaviour in school contexts. EPPI-Centre, Social Science Research Unit, Institute of Education, University of London, 2014.
- [27] L. Werner, J. Denner and S. Campe, Using computer game programming to teach computational thinking skills. In *Learning, education and games*, ETC Press, (2014), 37-53.
- [28] L. Werner, J. Denner, S. Campe & D. C. Kawamoto, The fairy performance assessment: measuring computational thinking in middle school. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, 2012, 215-220.
- [29] S.Grover. “Systems of Assessments” for Deeper Learning of Computational Thinking in K-12. In *Proceedings of the Annual Meeting of the American Educational Research Association*, Chicago, USA, 2015, 15-20.
- [30] D. T. Conley & L. Darling-Hammond, *Creating systems of assessment for deeper learning*. Stanford Center for Opportunity Policy in Education, Stanford, USA, 2013.
- [31] Department for Education and Skills (DfES), *14–19 education and skills HMSO*, Norwich, 2005e.
- [32] G. Claxton, *Building learning power: Helping young people become better learners*, Vol. 9. TLO, Bristol, 2002.
- [33] R. J. Sternberg, Metacognition, abilities, and developing expertise: What makes an expert student? *Instructional science*, 26(1-2), (1998) 127-140.
- [34] R. Fisher, *Teaching children to think*. Nelson Thornes, 2005.
- [35] G. Schraw, K. J. Crippen & K. Hartley, Promoting self-regulation in science education: Metacognition as part of a broader perspective on learning. *Research in Science Education*, 36, (2006) 111-139
- [36] D. Whitebread, P. Coltman, D. Pino Pasternak, C. Sangster, V. Grau, S. Bingham, Q. Almeqdad, D. Demetriou, The development of two observational tools for assessing metacognition and self-regulated learning in young children. *Metacognition and Learning*, 4(1), (2009) 63-85.
- [37] Lev S. Vygotsky. *Thought and language*. A. Kozulin (Ed.). MIT Press, Cambridge, MA, 1986.
- [38] M. M. Rohrkemper & B. L. Bershon, Elementary School Students’ Reports of the Causes and Effects of Problem Difficulty in Mathematics. *The Elementary School Journal* 85, 1, (1984) 127-147.
- [39] S. Lundgren & S. Bjork, Game mechanics: Describing computer-augmented games in terms of interaction. In *Proceedings of TIDSE*, 3, 2003.
- [40] R. Hunicke, M. LeBlanc & R. Zubek, MDA: A formal approach to game design and game research. In *Proceedings of the AAAI Workshop on Challenges in Game AI*. 4 (1), (2004), 1-5.
- [41] Matthew Weise, *The Future Is Now - Emergent Narrative Without Ridiculous Tech*. Paper presented at the Game Developers Conference Online, Austin, 2011.
- [42] N.K. Denzin and Y.S. Lincoln, *The SAGE Handbook of Qualitative Research* (4). Sage Publications, London, 2012.
- [43] A. B. Marvasti, *Qualitative research in sociology. An introduction*. Sage Publications, London, 2004.
- [44] BERA Ethical Guidelines for Educational Research, 2011. <http://www.bera.ac.uk/wp-content/uploads/2014/02/BERA-Ethical-Guidelines-2011.pdf>. (Accessed June 1, 2018).

- [45] R. Flewitt, Conducting research with young children: Some ethical considerations. *Early child development and care*, 175(6), (2005), 553-565.
- [46] M. B. Miles & M. A. Huberman, *Qualitative data analysis: A sourcebook of new methods* (2.). Sage Publications, Newbury Park, CA, 1994.
- [47] A. Strauss & J. Corbin, *Basics of qualitative research: Grounded theory procedures and techniques* (3). Sage Publications, Newbury Park, CA, 2008.
- [48] Hsieh, H.F. and Shannon, S.E., Three approaches to qualitative content analysis. *Qualitative health research*, 15(9), (2005), 1277-1288.
- [49] Berg, B.L. *Qualitative Research Methods for the Social Sciences*. Allyn and Bacon, Boston, 2001.
- [50] J. Moreno-León & G. Robles, Dr. Scratch: A web tool to automatically evaluate Scratch projects. In *Proceedings of the workshop in primary and secondary computing education*. ACM Press, 2015, 132-133
- [51] Y.B. Kafai, *Minds in Play: Computer Game Design as a Context for Children's Learning*. Lawrence Erlbaum Associates Hillsdale, NJ, 1995.
- [52] Y.B. Kafai, Electronic play worlds: Gender differences in children's construction of video games. In: Y. B. Kafai & M. Resnick (Eds.), *Constructionism in practice: Designing, thinking, and learning in a digital world*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1996, pp. 97-123

Appendices

Appendix A – Data analysis of interview script

①

YA [00:00:02] This is the interview with [REDACTED]

YA [00:00:07] Can you tell me about your game making experiences when using Scratch and Alice.

YA [00:00:12] Which one do you prefer and why?

[00:00:14] Am I like them both the same. Is like. to me game making is doesn't matter what you program you use, it just matter that what kind of game you make, what kind of content you have for your game. so but The thing with Scratch that is kind of a little bit different than Alice but.

YA [00:00:37] How different do you think?. What is the difference?

I think with Scratch is s more like for beginners because it's 2D and is what with it is not a lot options, more easier than Alice.
COMPARING ALICE & SCRATCH

[00:00:49] You see with Alice, once you've done yeah it you can put extra stuff like duration and its expressions and.
↓ ALICE

YA [00:00:55] So Alice is 3D?. Do you think that is important.

[00:01:00] If you want to make a game like a game you want people to see I think it is a bit important.

YA [00:01:06] Let's talk about your thinking process When you make games. how do think. What are the steps that you've followed.

[00:01:14] Firstly like. Where. When i need do like run ball sometimes I do an If you told me to make game like just on the spot Firstly I'll try to think if i get something like real kind of thing and change a bit like change it into something different. So I'll do that. Or maybe I can just think of it in my head. If you want one and then then I will go on to try and find out how I want them movement to happen.
↓ THINKING IN HEAD

YA [00:01:43] So you will hear that idea in your mind and then go and try..

[00:01:50] Like explore how you can different ways you can make them move and then you find the best way do it then you can put that in your script.

YA [00:01:56] so you explore it and then you kind of get ideas after that?

[00:02:01] You explore it and once found the best way to do it. → EXPLORING

[00:02:04] Like the way looks best then you put it in your actual script. Do
↓ MAKING DECISIONS

YA [00:02:08] you ever change that or do you keep.

[00:02:10] Yep I change a lot. Oh yeah yeah because I always notice that is will be better like this or I've done this wrong. I need to go back and do it.
CHANGING THE PLAN

YA [00:02:21] Do you think learning game design has changed the way you think?

[00:02:29] I think it changed a lot in maths. → IMPACT ON MATHS

2

YA [00:02:31] How?

[00:02:32] Because the way in Maths is English you there.

[00:02:36] There is no specific answer like you just write or read. There isn't a specific answer but in maths there's always a specific answer for it. For instance if we do two plus two it won't be nothing else but 4. Like when you make a game. If the script has to be exactly perfect or it won't make that game.

↓ PROBLEM SOLVING

YA [00:02:55] So it's kind of help you to think when you solve problems.

[00:03:00] yeah problems every kind of things .

[00:03:05] What is the most important thing that you think you learned by making games?

[00:03:12] I think My imagination.

YA [00:03:13] How?

IMAGINATION CREATIVITY ?

[00:03:14] Because like it expresses your imagination different points and it tells you you can come up with good things to say.

[00:03:21] Games do. And also I think it boosts your confidence as well because if you like to give it to other people, like show other people and they say if you did they give you negative or good feedback then you do it. Okay.

↓ FEEDBACK & CONFIDENCE

YA [00:03:49] Would you be able to use what you learn in game design in other lessons?

[00:03:59] like skills that you learned and then games. Yes. Problem solving, you said. What else do you think?

YA

[00:04:05] ermm maybe it could help you, I don't really know.

[00:04:12] It is maths..

[00:04:15] It is like. I was thinking about scripting and storyboard and things so we create. would you be able to use them?

[00:04:23] you would find it easier to make story.

Impact on LITERACY

[00:04:25] Yeah my help you with your story.

YA [00:04:29] When you make game you mentioned thinking in your head at the beginning, so do you talk or ask questions to yourself.

[00:04:34] Yeah I do when. When. I use one before. I like make people, make people see. I say to myself Are you sure it is alright. I go and check that everything seems alright. now even though one little thing I try and change it change.

→ TALKING TO SELF

YA [00:05:01] Can you give me an example, like, just give me an example that you did communicate to

3

yourself.

[00:05:08] ermm when I was making the rebel fighting game. I wanted to see I thought to myself How do you make your life more interesting more detailed and make it more like a movement, because you told me to make moving, so I thought maybe let me just think of a way to do it like maybe add boys or make them to do extra stuff like, detail basically.

→ - MAKING DECISIONS
- THINKING
- TALKIN TO SELF

YA [00:05:35] How do you ask yourself. Give me an example. How do you.

[00:05:38] I just say in my mind. I just say I did ask myself. Once I ask myself I try to think.

↓ TALKIN TO SELF

YA [00:05:43] Give me an example.

[00:05:45] I say. ermm What should I do. What should I do fix this life.

[00:05:50] There's something wrong I'd say that in my head and.

YA [00:05:52] What does that do when you ask that to yourself. What does.

[00:05:55] It kind of makes you think better because it making me, it makes when you don't ask yourself you just try to figure out. when you ask yourself sounds a bit weird. But you know like definitely you need, like do it.

↓ TALKING TO SELF

YA [00:06:08] I see.

[00:06:08] If.... repeating is better.

YA [00:06:10] ok. Can you list some new vocabulary that you learn when making games.

[00:06:27] Expressions.

[00:06:28] Yeah.

[00:06:36] Opacity. it makes like invisible Yeah.

[00:06:42] Variables. algorithm. Yeah. Yeah.

COMPUTATIONAL CONCEPTS

[00:06:56] Debugging is if there is a problem you go back to check the coding and correct it. Break into pieces and make it work, thats called debugging.

YA [00:07:12] And what did you do when you come across a problem. How did you go about solving it. Can you talk about the steps.

[00:07:20] Firstly I'll try to figure out it myself.

YA [00:07:22] How do you try to do that?

[00:07:24] Like I said I've asked myself, you just think.

→ - TALKIN TO SELF
- PROBLEM SOLVING

[00:07:26] You go search through your game, search and look at everything that script was right. And you need to also ask yourself Like.

Programming Constructs in Scratch and Alice

Sequences can be described as the series of steps for completing a task that can be executed by the computer. In Scratch an example of this could be a script to program a sprite to move across the screen. In Scratch any object that can be programmed is called sprite. In Alice consecutive sequences for one object are created using a ‘do in order’ block. For more than one object a ‘For all in order’ block is used. This command will execute an operation on each object in a list one at a time, beginning with the first object in the list and completing the list in order. Figure B.1 shows what a sequence looks like in Scratch and Alice environments.

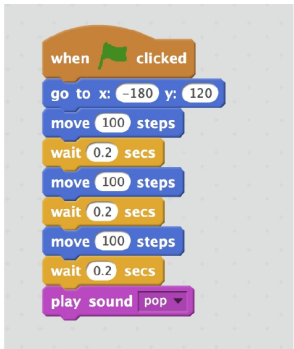

Scratch	Alice 2.4
<p>This instruction sequence programs the ball sprite to move across the screen and then play the sound ‘pop’.</p> 	<p>Using ‘Do in order’ statement in Alice, it is possible to run the codes in a consecutive order for an object.</p> 

Figure B.1: Sequence in Scratch and Alice

Loops involve the repeated execution of a sequence of statements. They make code writing more efficient by using the repeat function instead of creating a long script that would describe the same actions. For example in Figure B.2 we created a script that makes the ball sprite move across the screen. A more efficient way of writing this would be repeating the move 100 steps and wait 0.2 secs code blocks 3 times rather than using six coding blocks. An instruction in Scratch can be repeated for a specific number of times or infinitely which is the forever block. In Alice a ‘Loop’ block is used for repeated actions. Figure B.2 displays the script using the repeat function in Scratch and Alice.

Scratch	Alice 2.4
	

Figure B.2: Loop in Scratch and Alice

Events refer to one action causing another action to happen. In Scratch there are different ways an event can produce an action. For example; when the green flag is clicked or when a key is pressed. In Alice creating a new event to produce an action is possible by clicking on the ‘create a new event’ tab which has options such as when the world starts, when a key is typed, when a variable changes or when the mouse is clicked on something. Figure B.3 shows some of the ways that events can produce actions in Scratch and Alice.


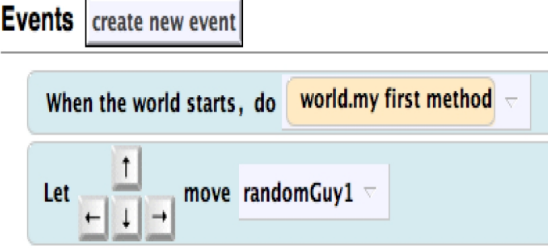
Scratch	Alice 2.4
	

Figure B.3: Events in Scratch and Alice

Parallelism is making events take place at the same time for different characters or for the same character. In Scratch this is done through using the same event block for different actions. For example you can make a character think, change costume and play sound all at the same time ‘when the green flag’ is clicked. In Alice parallelism is supported using ‘Do it together’ block. ‘For all together’ block is used for performing an operation on all the objects in a list at the same time. Figure B.4 displays how parallelism is supported in Scratch and Alice.



Scratch	Alice 2.4
	

Figure B.4: Parallelism in Scratch and Alice

A **‘conditional’** is an instruction in a program that is only executed when a specific condition is met. In the Scratch application a conditional is defined using the ‘if block’. For example in our example in Figure B.5 the condition for the ball sprite to play the ‘pop’ sound is to touch the apple sprite. In Alice conditionals are set using ‘if / else’ block and it allows actions to be performed when a Boolean expression is true. For example ‘if the car is within 1 metre of the tree, the car will play the doorbell sound’. There is also the while statement, which repeats the instructions inside a loop as long as the Boolean condition, is true.

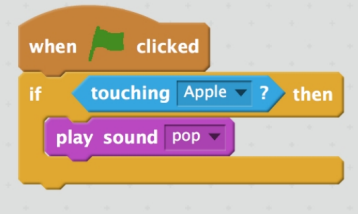
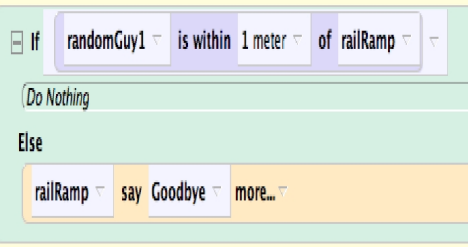
Scratch	Alice 2.4
	

Figure B.5: Conditionals in Scratch and Alice

Operators are functions for both mathematical and logical expressions and it enables the use of both numeric and string operations. In Scratch this is done using the codes inside the ‘operators palette’. In Alice logical operators such as ‘not a’, ‘both a and b’, ‘either a or b, or both’ are used for connecting comparisons to form more complex Boolean expressions. Relational operators such as equal to, greater than, less than are also used for forming comparisons. Mathematical expressions such as addition, subtraction, division and multiplication are also used in the Alice 2.4 programming environment.

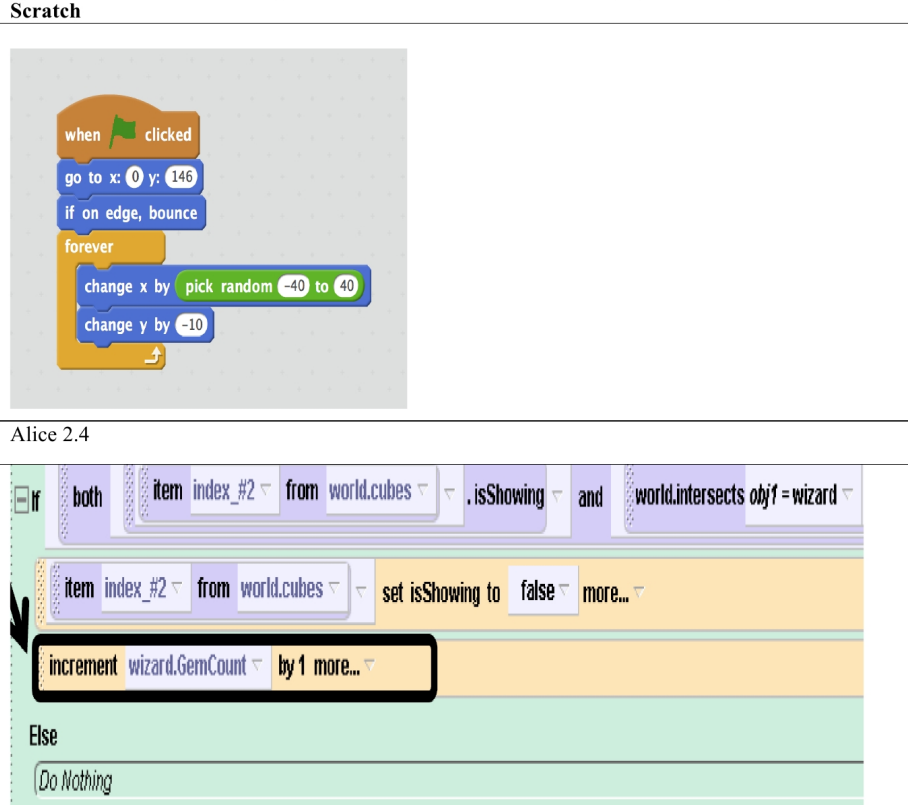


Figure B.6: Operators in Scratch and Alice

Variables are the placeholders for information to be used by programs, which can change depending on conditions. The common use of variables in Scratch and Alice is creating score or a timer for games.. In Scratch variables are created using a ‘Data’ block. . In Alice, variables are created using the ‘Create a new variable’ tab and it includes data as numbers, objects, boolean (true and false) or string. Variables in Alice 2.4 can be created for a method (local variable), to hold an argument (Parameter variables), for a specific object (Class-level) or for all objects in a world (World –level variables). Figure B.7 shows scripts for timer variables in both Scratch and Alice.

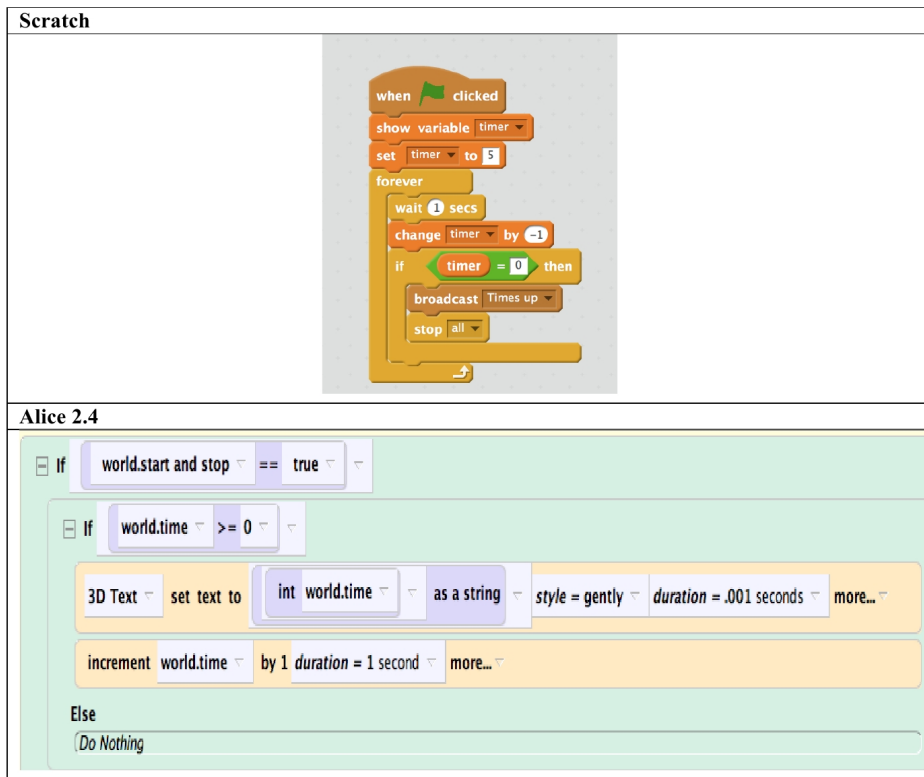


Figure B.7: Timer variable in Scratch and Alice

Abstraction is the process of removing details about an object to reduce complexity and increase efficiency. In Scratch this can be seen as organising instructions into code stacks based on their functions using user defined blocks. The example for Scratch in figure B.8 defines the script for drawing a rectangle.

In Alice abstractions are performed through methods as they include actions that can be executed by objects in the world when they have been requested. In Alice methods can be defined at either character level (applying to one object only) or at world level (applying to many objects).

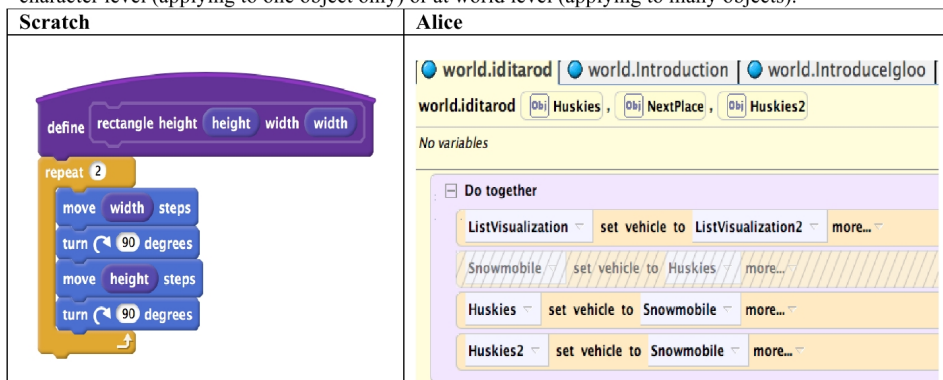


Figure B.8: Abstraction in Scratch and Alice

Individual Case study: Scratch Game

The individual case study of one game created by two boys below will illustrate the programming constructions that some children used whilst creating their games and animations. The reason for choosing this game was that it included more sophisticated coding structures than the other games, therefore it makes it possible to illustrate the range of programming concepts that the students used whilst making their game.

The game selected for this individual case study was called ‘Kick about’ and it was created by two boys (B and K). The aim of ‘kick about’ game is to stop the ball sprite (character or object) that is coming from the top of the screen by moving the kicker sprite vertically using the left and right arrow keys. The kicker sprite stands on a red line and if the ball touches this line, then the kicker loses a life. Once all of the five lives have been used, ‘game over’ text appears on the screen and the game stops. If the kicker sprite can stop the ball before it touches to the red line, it gets one point as a score. Figure C.1 displays the game interface.



Figure C.1: Kick about game interface

Children B and K created the game collaboratively. They used programming constructions e.g. sequences, loops, parallelism, conditionals, operators, variables and events for creating their game. The students did not use any custom built blocks to define a new instruction or any other method of abstraction. One reason for this might be that they did not need any action that would require the creation of a new function; therefore they were able to complete their task with pre-built code blocks. Another reason is that they may not have had sufficient knowledge for creating new functions as I only modelled this once in the classroom, however, they could have used the Internet to develop their understanding in this area as they did use online videos often when they did not know how to do something.

‘Kick about’ game programming constructs	
Sequences	For the ball sprite students first set the score to zero, then defined the position of the ball using series of codes.
Loops	Forever block was used for repeating the movement of the ball sprite for 8 steps and making it bounce when it reaches to the edge of the screen.
Parallelism	Three sets of script were




	written for the ball sprite and they were all executed at the same time when the green flag clicked.	
Conditionals	If statement was used for defining the behaviour of the ball sprite when it touches to the 'kicker' sprite.	
Operators	The condition for the ball sprite had a statement that used minus (-) operator. If the ball touches the kicker ball should point direction less 160 (direction - 160). This script sets the direction for the ball sprite when it meets the condition, which is touching the kicker sprite.	
Variables	Data Score for the ball and lives variables for the kicker sprite were used for this game.	
Events	When the ball sprite broadcasts game over, then the hidden game over text (sprite) appears on the screen and the whole game stops.	
		Scripts for ball sprite
Abstraction	The student did not use abstraction for this game.	

Table C.1: Scratch Case Study

Individual case study: Alice

The game that was selected for this case study was called Badguyrobot. It was created by two boys aged ten. It is an animation with two characters: badguyrobot and goodguyrobot. This game was selected because it included many of the programming constructs therefore it was appropriate for illustrating these using example scripts from the game. The scene background is space. Figure D.1 shows the selected animation screen.

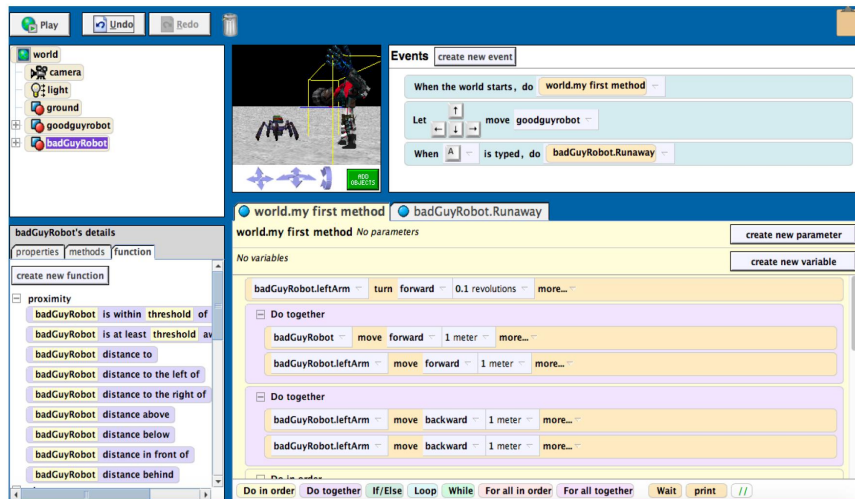


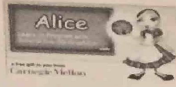
Figure D.1: badguyrobot and goodguyrobot animation screen

The students used both built-in methods and created their own ones. They started by programming the left arm of the badguyrobot character to point forwards by specifying the turn revolutions. They used a few 'Do together' constructions to make both characters move at the same time. They used 'Do in order' statements to program the goodguyrobot character to complete 4 actions simultaneously. A 'Loop' was used for making the goodguyrobot say goodbye if the position of 'badguyrobot' is within a metre of goodguyrobot is untrue. In the events section the students had their first method called when the world starts. They then had another event to control the goodguyrobot with arrow keys. They created a new method called runaway and an event handler to run it when 'A' letter key is clicked on the keyboard.

The students did not create any variables or parameters when creating the 'Goodguyrobot and Badguyrobot' animation. They seemed to use built-in 'say' construct for creating a dialog between the two characters. They used constructs to manipulate subparts of the objects e.g. left arm. They recorded their own voices and used these for each character. They were able to create and use new methods, which illustrates that they were able to apply abstraction to complete their task. They used conditionals and relational operators (Numerical and logical expressions) to define the behaviour of the characters. Their programming constructs were tested and they worked as expected.

	Goodguy robot and Badguy robot animation programming construct
Sequences	<p>'Do in order' statement</p> <p>Do in order</p> <p>goodguyrobot roll right 0.25 revolutions duration = 2 seconds more...</p> <p>goodguyrobot say I will not be defeated by the dark side more...</p> <p>badGuyRobot say oh yes you will duration = 3 seconds more...</p> <p>goodguyrobot roll left 0.25 revolutions more...</p>
Loops	<p>Loop statement</p> <p>Loop 2 times times show complicated version</p> <p>badGuyRobot say We will meet soon again. You can't run away! more...</p>
Parallelism	<p>'Do together' statement</p> <p>Do together</p> <p>badGuyRobot.leftArm move forward 1 meter more...</p> <p>badGuyRobot move forward 1 meter more...</p>
Conditionals	<p>If/Else statement</p> <p>If badGuyRobot is within 1 meter of goodguyrobot</p> <p>Do Nothing</p> <p>Else</p> <p>goodguyrobot say Goodbye more...</p>
Operators	<p>Relational operators</p> <p>If badGuyRobot is within 5 meters of goodguyrobot</p> <p>badGuyRobot say Ha ha ha more...</p>
Events	<p>Something that occurs while an Alice program is running</p> <p>Events create new event</p> <p>When the world starts, do world.my first method</p> <p>Let move goodguyrobot</p> <p>When A is typed, do badGuyRobot.Runaway</p>
Variables	<p>Variables</p> <p>None</p>
Abstraction	<p>Methods</p> <p>world.my first method badGuyRobot.Runaway</p> <p>world.my first method No parameters</p> <p>No variables</p>

Table D.1: Alice Case study



MY GAME DESIGN LEARNING JOURNAL

My game was about...

a Alien Scaring a space man in space

I have learnt to...

use the web gallery for more objects to put in the world

I asked/talked/thought to myself...

I asked my self that how ^{is} it doing to the work and how is it going to ~~exist~~

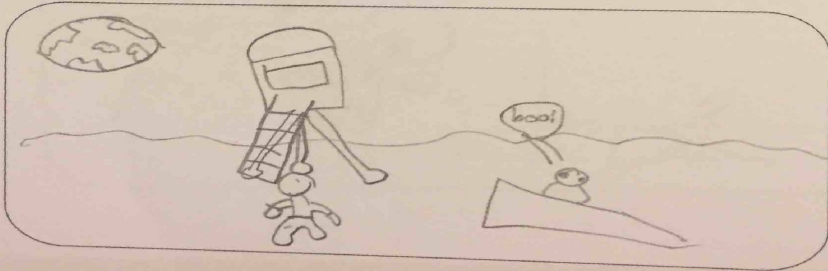
I discussed with my friend about...

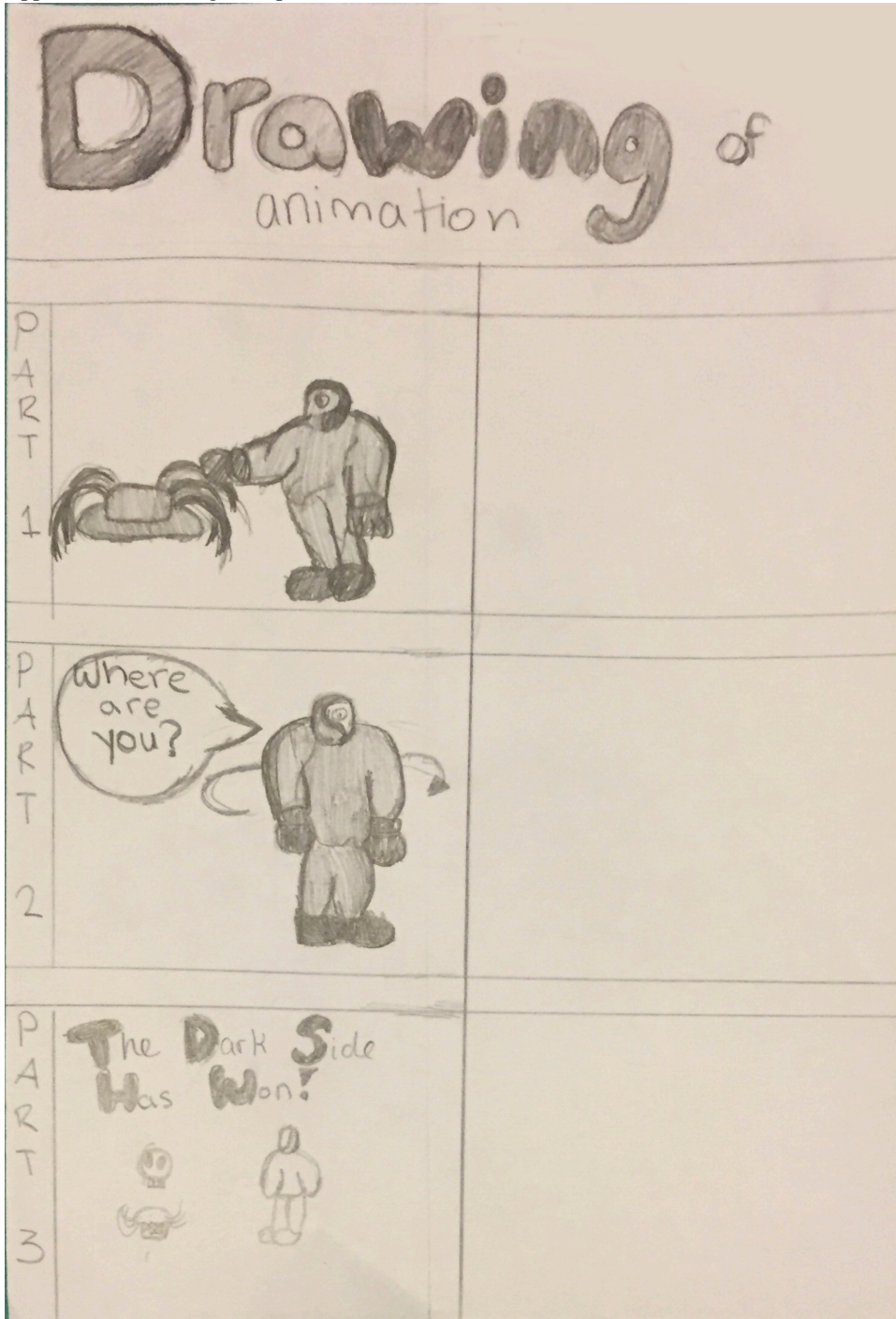
How to open the web gallery

My problem and how I solved it:

We didn't have any problems but our friends did, they didn't know how to open the web gallery. Some of them did but they did not have some of the characters.

My game looked like this:

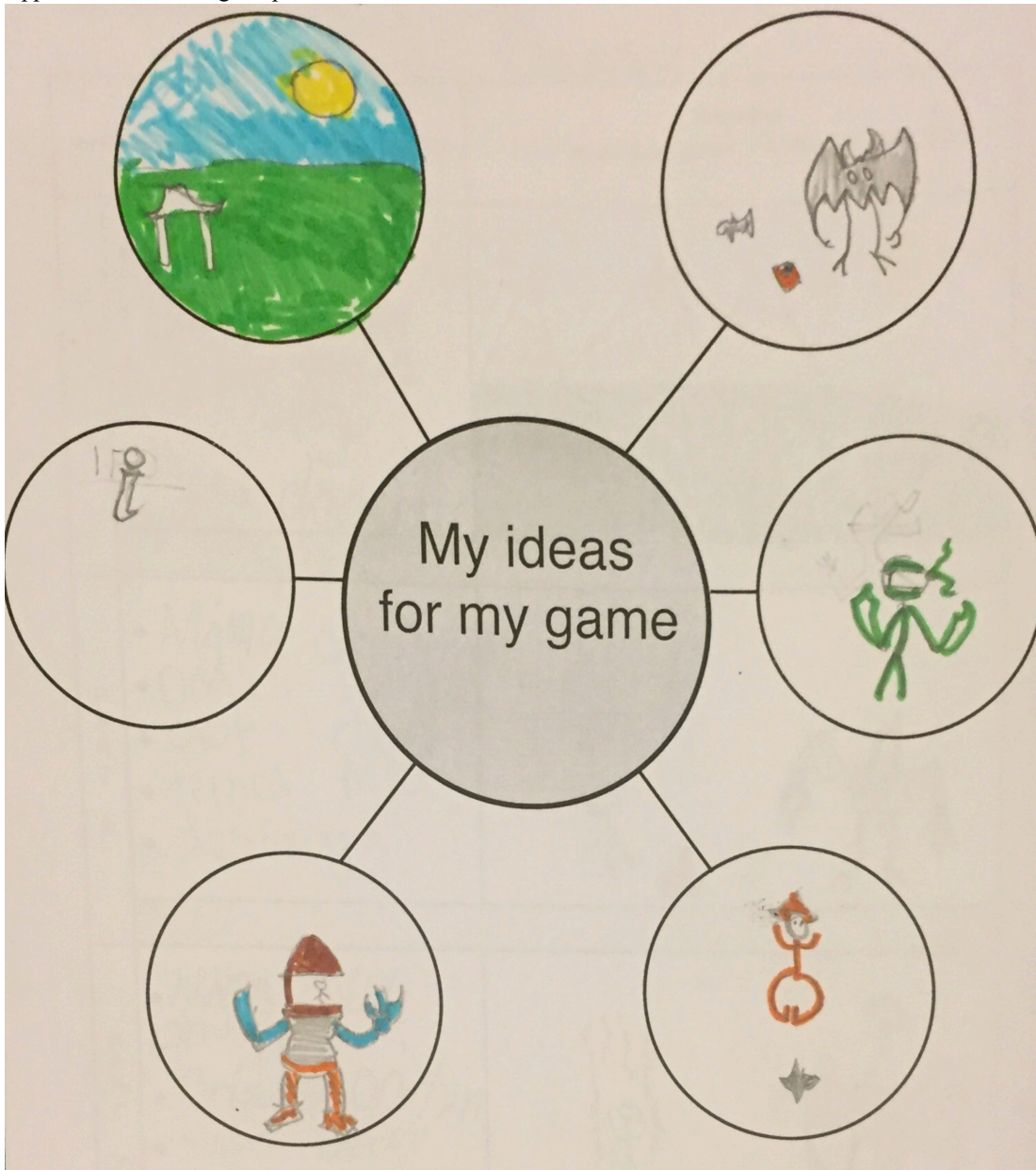




My Title: City races
 Background: City
 Characters: Josh

My

	Drawings	Events
P A R T + 1		<ul style="list-style-type: none"> • Josh comes home • Cat lays on door step • Police chases car • Josh "hi Normal" • Normal "Purs"
P A R T + 2		<ul style="list-style-type: none"> • Car drives up to house • Josh says "Look that guy got pulled over" • Police move to car • Police step out of the vehicle • Guy gets out of car
P A R T + 3		<ul style="list-style-type: none"> • Police gets out of car • Josh "shut the fridge don't arrest him hes my dad townhouse" • "Not that fridge" • townhouse "Sorry"
P A R T + 4 		<ul style="list-style-type: none"> • "Really dad really" Josh • Dad - Hey Josh I got the eggs! • Dad Haha Haha • But its not funny • Normal ^{thinks} says not again • Normal moves away from house



Scripting		Drawing	
Write down the main events as a list to help you with your planning		How would your game / animation look like?	
P A R T 1	<ul style="list-style-type: none"> ◆ Ninja walks towards Bat ◆ Bat mutates ◆ Poison drops from sky ◆ Ninja doesn't see 		
P A R T 2	<ul style="list-style-type: none"> ◆ Ninja confused ◆ Drink's bat blood ◆ got cube ◆ turned mech ◆ destroys 		
P A R T 3	<ul style="list-style-type: none"> ◆ Ninja sets on fire ◆ Drinks poison ◆ turns better ◆ gold power 		