ONDOKUZ MAYIS UNIVERSITY

GRADUATE SCHOOL OF EDUCATIONAL SCIENCES

COMPUTER EDUCATION AND INSTRUCTIONAL TECHNOLOGY DEPARTMENT

USING STARLOGO IN EDUCATION: DEVELOPING EDUCATIONAL GAME WITH PROGRAMMING

NOMIN BOLDBAATAR

ADVISOR:

Asst. Prof. Dr. EMİNE ŞENDURUR

THESIS OF MASTER DEGREE

JULY, 2017

ONDOKUZ MAYIS UNIVERSITY

GRADUATE SCHOOL OF EDUCATIONAL SCIENCES

COMPUTER EDUCATION AND INSTRUCTIONAL TECHNOLOGY DEPARTMENT

USING STARLOGO IN EDUCATION: DEVELOPING EDUCATIONAL GAME WITH PROGRAMMING

NOMIN BOLDBAATAR

ADVISOR:

Asst. Prof. Dr. EMİNE ŞENDURUR

THESIS OF MASTER DEGREE

JULY, 2017

TELİF HAKKI

YAZARIN

Adı :

Soyadı :

Bölümü :

İmza :

Teslim Tarihi :

TEZİN

Türkçe Adı :

İngilizce Adı :

ETİK İLKELERE UYGUNLUK BEYANI

Tez yazma sürecinde bilimsel ve etik ilkelere uyduğumu, yararlandığım tüm kaynakları kaynak gösterme ilkelerine uygun olarak kaynakçada belirttiğimi ve bu bölümler dışındaki tüm ifadelerin şahsıma ait olduğunu beyan ederim.

Yazar Adı Soyadı: ……..

İmza: ……..

KABUL VE ONAY

……. tarafından hazırlanan "**……**" adlı tez çalışması aşağıdaki jüri tarafından oy birliği/oy çokluğu ile Ondokuz Mayıs Üniversitesi Bir öğe seçin. Anabilim Dalı'nda Yüksek Lisans / Doktora tezi olarak kabul edilmiştir.

Danışman: (Unvanı Adı Soyadı)

(Anabilim Dalı, Üniversite Adı) **...**……

Başkan: (Unvanı Adı Soyadı)

(Anabilim Dalı, Üniversite Adı) ……...

Üye: (Unvanı Adı Soyadı)

(Anabilim Dalı, Üniversite Adı) ……...

Üye: (Unvanı Adı Soyadı)

(Anabilim Dalı, Üniversite Adı) ……...

Üye: (Unvanı Adı Soyadı)

(Anabilim Dalı, Üniversite Adı) ……...

Bu tezin Bir öğe seçin. Anabilim Dalı'nda Yüksek Lisans/ Doktora tezi olması için şartları yerine getirdiğini onaylıyorum.

Tarihi: __/__/____

Eğitim Bilimleri Enstitüsü Müdürü

(İmza ve Mühür)

STARLOGO TNG YAZILIMINI KULLANARAK GÖRSEL PROGRAMLAMA YARDIMIYLA EĞİTSEL OYUN TASARLANMASI

Yüksek Lisans Tezi

Nomin Boldbaatar

ONDOKUZ MAYIS ÜNİVERSİTESİ

EĞİTİM BİLİMLERİ ENSTİTÜSÜ

Temmuz, 2017

ÖZ

Programlama eğitimi, son zamanlarda bilişim teknolojileri alanında temel konulardan biri haline gelmiştir. Fakat öğrenciler genelde metin tabanlı programlama dillerinin karmaşık yapısından dolayı bazı zorluklarla karşılaşmaktadır. Bu nedenle programlama eğitimi daha çok programlamanın soyut yapısını somutlaştırmaya odaklanmaktadır. Eğitim ortamında eğitsel oyunlar çok yaygın kullanılmaktadır. Çalışmanın amacı öğrencilerinin daha önceki programlama deneyimi ve becerilerini yeni ortama nasıl aktardıklarını gözlemlemektedir. Bilgi ve becerilerinin aktarılması için geriye sönümlü metot kullanılmıştır. Bu çalışmada, üniversite son sınıf öğrencilerinin (N=21) "E-Oyun Tabanlı Öğrenme" dersi kapsamında görsel programlama kullanarak oyun tasarlamaları süreci incelenmiştir. Bütün katılımcılar metin tabanlı programlama hakkında temel bilgilere sahiptirler (Python, Java vs.). Fakat bu öğrencilerin bazıları (N=11) daha önceden blok temelli programlama deneyimine sahip iken diğerleri (N=10) sadece metin tabanlı programlama deneyimlerine sahiptirler. Araştırma boyunca çalışan örnek, tamamlamalı örnek ve tam uygulamalı örnekler sunulmuştur. Bu çalışma bilgisayar oyun tasarlama sürecindeki geriye sönümlü örneklerin değerini vurgulamayı hedeflemektedir. Farklı programlama deneyimine sahip olan öğrencilerin çalışma boyunca yaptıkları oyunlar karşılaştırılmıştır. Ayrıca yapılan bütün örnek formatların, algılanan zorlukların ve bilişsel yüklerin etkilerini karşılaştırmıştır. Başlıca dört bulgu dikkat çekmektedir. İlk olarak metin temelli programlama deneyimi olan öğrenciler diğer öğrencilere göre daha yüksek final puanları alması uzak transfer olduğunu göstermiştir. İkinci olarak tamamlamalı örneğin, deneyimli öğrenciler için daha etkili bir yaklaşım olduğu görülmüştür. Üçüncü olarak öğrencilerin algıladığı zorluk derecelerine dayanarak, tam uygulamalı örneğin diğer çözümleme ve tamamlamalı örneklerden daha çok zorlandıkları görülmüştür. Son olarak örnek formatlarının etkililiğine baktığımızda tamamlamalı örnekler bütün öğrencilerde yüksek etkililik göstermiştir. Fakat bütün örneklerin ortalama etkililik dereceleri,* daha önce hiç blok temelli programlama deneyimine sahip olmayan öğrenciler için yüksek etkililik göstermektedir.

Anahtar kelimler **:** oyun tasarımı, çalışan örnek, geriye sönümlü örnek, blok temelli programlama

Sayfa sayısı **:** 75

Danışman : Yrd.Doç.Dr Emine Şendurur

USING STARLOGO IN EDUCATION: DEVELOPING EDUCATIONAL GAME WITH PROGRAMMING

MS Thesis

Nomin Boldbaatar

ONDOKUZ MAYIS UNIVERSITY

GRADUATE SCHOOL OF EDUCATIONAL SCIENCES

July/2017

ABSTRACT

Programming education has certainly become one of the main fields in computer education for the last few decades. However, students generally have difficulties with sophisticated nature of text-based programming languages. Therefore, programming education has focused more and more on making abstractness of programming tangible. The implement of educational game is widespread in educational settings. The purpose of this study is to observe how students can transfer their previous programming knowledge and skills into new situation. In order to observe the transition of their knowledge, backwards-fading method used during the study. This study examined senior university students' (N=21) game making process using block-based programming within "E-Game Based Learning". All participants have basic knowledge of text-based programming language such as Python, Java, etc. However, some of students (N=11) had prior block-based programming experience where as other students (N=10) had no block-based programming experience. During this study, different example formats were presented: *worked-example*, *completion-example*, and *full practice*. This study addresses the added value of faded worked examples in a computer game-design process. Games of students with different programming experience have compared throughout the study. Moreover, the comparison of effect of three examples, their perceived difficulty and cognitive load are also observed. Four findings are on interest. First, students having text-based programming experience had higher scores than final scores of others which produced far transfer. Second, completion-example format was more efficient than others for non-novice learners. Third, full-practice format was more difficult than either worked-example or completion-example based on student perceived difficulties. Fourth, based on the efficiency of example formats, completion example represented high efficiency for all students. However, average efficiency of all example formats represented high efficiency for students who had no prior block-based programming experience. The theoretical and practical implications are also discussed.

Key words : game design, worked-example, backwards fading, block-based programming
Number of Pages : 75
Advisor : Asst. Prof. Dr. EMİNE ŞENDURUR

CONTENTS

TABLES

FIGURES

CHAPTER I

I. INTRODUCTION

For the young generation, playing computer games have become a significant part of their lives and culture (Wilson, Connolly, Hainey & Moffat, 2011). Besides the growing popularity of playing computer games, it's believed to result in diversified amount of benefits, like increasing interest and motivation on students as well as, for instructors like reduction of teaching time and instructor load (Petri & Gresse von Wangenheim, 2017). For the last few decades, educational computer games have become a part of education. Educational games certainly motivate and engage learners with the subject in an enjoyable way. Computer games are used in many subjects such as math (e.g. Astra Eagle), physics (e.g. Supercharged), biology (e.g. The Minecraft Cell), chemistry (e.g. EMD PTE), history (e.g. World History Games), and language (e.g. Duolingo) education. Many years have passed since the first computer programing education was introduced to schools in the early 1980s. According to Papert (as cited by Begel, & Klopfer, 2004), programming in school curricula encountered some serious problems and as known from the history of instructional media, computers alone were not the panacea for all problems. However, with the increased popularity of the Internet, learning started to take various forms. Apparently, computer science has become one of the fast-growing and developing industries and technology has begun to be used in various ways in education. Thus, way of learning has changed over time. For example, they know how to access any information on the web whenever needed, which can lead to a tendency towards searching and filtering skills rather than rote memorization. Besides learning, the Internet, apparently, has started to take a big part in everyday lives of people. According to the Turkish Statistical Institution (2016), many people spend significant amount of their time with the computers or mobile devices for various activities ranging from social media (82,4 %), online news (69,5 %), and health (65,9%) to entertainment such as videos (74,5 %) and web radio (63,7 %).

Many students have grown up by playing games including non-digital ones such as board games and hide-and-seek. However, digital games are one of the most frequently preferred activities especially among young users (Fromme, 2003). The literature shows that one of the major environments which aim to help learners to learn computer programming and to adopt logical thinking skills in more interesting way is educational games (Backlund & Hendrix, 2013; Battistella & Wangenheim, 2016; Malliarakis, Satratzemi, & Xinogalos, 2013; Maloney, Peppler, Kafai, Resnick, & Rusk, 2008; Petri & Gresse von Wangenheim, 2017). The educational games can serve as appropriate tools for both formal and informal learning situations (Kirriemuir & Mcfarlane, 2004) of the new generation learners since they are more motivated to learn from educational games which, provide attractive graphics and scenarios (Malliarakis, Satratzemi, & Xinogalos, 2013; Malliarakis, Satratzemi, & Xinogalos, 2014). In many years, number of studies have investigated about using computer games in educational settings. However, there are not enough studies about how students create and design their own (educational) games and its impact on students learning. Recently, according to the some educators it's assumed that making games for learning instead of playing game for learning is an efficient instructional strategy, there are examples in education enabling students design

and develop their own games (Hayes & Games, 2008). In this way, they could learn either to code or to design as they produce games. Moreover, designing computer games makes students active learners and creators instead of being consumers of computer games.

There are considerable amount of research investigating both educational games and programming languages (Leutenegger & Edgington, 2007; Malliarakis et al., 2013; Vasilateanu, Wyrazic, & Pavaloiu, 2016). Classical programming languages require great effort due to being abstract, and therefore keeping students involved, motivated, and interested in the topic throughout the course can become a challenge. There are various methods, strategies, and techniques to make students interested and motivated. Using games as a method or integrating games as tools for design and development can leverage the attention span. Unlike abstract programming environment; visual programming environments such as Scratch provide students with more concrete results. In other words, instead of writing codes, they are allowed to drag-drop the code blocks. When they run the code blocks, the output is more tangible. Although such environments are very useful for beginners, they are hardly suitable for applications requiring complex programming (Barista , Connolly, & Angotti, 2016).

Lately, there has been an increasing interest in the combination of programming language and game design. Especially there are many studies mostly utilizing Scratch (Malan & Leitner, 2007; Maloney, Peppler, Kafai, Resnick, & Rusk, 2008; Wilson, Connolly, Hainey, & Moffat, 2011; Wilson, Hainey , & Connolly, 2013) and other tools: Alice 2.0 (Kelleher & Pausch, 2006); EToys (Lee, 2011); ToonTalk (Kahn, 1996); Starlogo TNG (Begel & Klopfer, 2004; Klopfer, Scheintaub, Huang, Wendel, & Roque, 2009). Using game has offered many opportunities to promote learning in an effective way (Garris, Ahlers, & Driskell, 2002). Students are encouraged to develop their own games with the integration of their knowledge and skills. Working on the visual design part of the games may trigger the intention to give movements, to set rules, and so on, which can be possible through the utilization of visual blocks of codes. Many studies are based on whether learners' learning performances are improved by teaching programming language using visual programming software. Participants of those studies are mostly young and adult learners who have (no) previous programming knowledge. However, there aren't enough studies focusing on students' ability to transfer their previous abstract programming knowledge into a three-dimensional visual-based programming environment. This study presents students ability to transfer their abstract programming language knowledge and skills into game-design with visual programming tools.

1.1. Purpose of the Study

One of the difficulties in education, especially in higher education, is transferring complex topics successfully into real life. In these situations, using educational game is considered to be the one the mostly used and beneficial methods (Mann, Eidelson, Fukuchi, Nissman, Robertson, & Jardines, 2002). As we know using game as a tool in learning has a plenty of

advantages. Educational games are expected to provide such an effective environment, which contains experiences similar to the real-life experiences. Moreover, it is proved that educational game can make up a strong educational environment that activates learner's previous knowledge and makes it easy to use (Papastergiou, 2009). Therefore for students, it becomes easy to learn these kinds of experiences by doing on their own and learning from their mistakes (Petri & Gresse von Wangenheim, 2017). According to the Ebner & Holzinger (2007), games are beneficial and useful for educational fields such as civil engineering and medicine (which can be interpreted that learning is more efficient when complex topics are presented to the students in the form of games. One of the most four common purpose of the game, which is created for educational purpose, is to help students to understand programming language (Hayes & Games, 2008). A considerable amount of studies have investigated the impacts of using educational games in various fields. However, there has been little attention to the learning experience of students as game designers. For the last few decades, designing computer games has been used in educational setting. The Logo Programming language developed by Seymour Papert, Wallace Feurzeig, and Daniel Bobrow in 1968 was the major game design environment where learners can make simple conversations with the computer through "turtles" (Papert, 1980). Similar to this, many tools in order to create 2D or 3D games are developed to help students to learn (simple) programming concepts and method. Most common software are Scratch, LogoBlocks, Unity 3D, Alice, Moose Croosing, ToonTalk, StarLogo TNG, and many others. Numerous studies that are related to Scratch have been done recently. Scratch is designed on the basis of visual programming. Using Scratch application, users can create various projects including games, animated stories, greeting cards, simulations, and many others. Users are able to program objects with drag-and-drop code blocks, rather than inputting a programming syntax. On the other hand, there is also software which allows students to design, create, and use their own fully functional mobile apps for Android. It is called AppInventor and it's also a drag-and-drop visual programming tool.

The related literature indicates that game design in education have a focus on teaching programming languages and other academic domains (math, history, science, etc…) as well as understanding game design concept. Many studies have investigated learners' ability to transfer their (previous) text-based programming language knowledge into block-based programming environment or vice versa. For example, Tabet, Gedawy, Alshikhabobakr, and Razak (2016) examined transfer from programming with Alice, which has the drag-and-drop interface, to programming with Python, which is a text-based environment. In the first year of their study, they prepared Alice-based curriculum for teaching computing for grade seven students. In the following year, they implemented Python-based curriculum for grade eight students. They found that starting programming education with Alice helped the knowledge transfer, and thus it facilitated the adaptation to Python. Another example, Armoni, Meerbaum-Salant, and Ben-Ari (2015) investigated the use of Scratch environment for teaching a professional text-based programming language (Java or C#) at secondary school. They observed that such an environment contributed students' motivation in addition to learning how to write codes. In a recent study, Weintrop and Holbert (2017) investigated how learners use both block-based and text-based environment. They used Pencil Code

programming environment. In their study, there were two groups of participants: high school condition and graduate condition. Both groups started using block-based environment of Pencil Code. During assignments, while learners were shifting from text-to-blocks, they preferred to use drag-and-drop over typing; therefore, they can prevent themselves from syntax errors. Findings also showed that learners generally shift from text to block-based modality to add new command or edit existing commands. It's shown that novice learners tend to work on block-based environment, while experienced learner are eager to work on text-based environment. They emphasised that using dual-modality provides learners to be able to switch between them as they wanted; therefore, it is encouraging modality for novice learners; at the same time; it keeps experienced learners engaged.

In this study, it's aimed to observe how students create educational game both using block-based programming language and designing their game in 3D medium. Also, it's aimed to observe how students can transfer their previous programming knowledge and skills into new situation. Moreover, in order to observe the transition of their knowledge, backwards-fading method used during the study. Backwards-fading has been integrated into many subject areas, but the examples are very rare in programming education, and therefore, this study has importance to shed light for further similar studies.

1.2. Research Questions:

This study aimed to investigate the following research questions:

- Does the use of backwards fading technique facilitate the transfer of block-based programming experience into 3D-game making in comparison to that of text-based programming experience?
    - Does the format of practice examples affect the 3D-game making performance of participants?
        - Does the format of practice examples affect the perceived difficulty?
        - Does the format of practice examples affect cognitive load?
- What are the perceived challenges and facilitators of working with 3-D game making environment?

1.3. Keywords

Text-based Programming: It is any programming language requiring the typing the codes in line with certain syntax.

Backward Fading: Although this concept is used in many other fields including maths and science, in this study we mean by backwards fading that small accomplishments of step by step solutions of code blocks. It starts with worked examples, continues with completion examples, and ends with full practice.

Block-based Programming: It is any programming language enabling the user to drag-drop coding blocks to create functional programs.

Cognitive Load Theory (CLT): It is "a set of learning principles that are proven to results in efficient instructional environments as a consequence of leveraging human cognitive learning process" (Clark, Nguyen, & Sweller, 2011, p. 7).

Completion Example: It exemplifies the solution to a certain extent and generally leaves a few steps to be solved by the learner.

Educational (computer) Game: It is any computer game designed according to certain instructional goals.

Far Transfer: Far transfer is about tasks, which require being adapted one's skill to a new situation each time that is more like transfer between contexts.

Full Practice: It is used as a term to define the tasks requiring the solution of a whole problem by the learner without any guidance.

Game Design: In this study, we assumed it as a process consisting of scenario creation, rule definition, character creation, graphical environment creation, and building functional code blocks.

Information Processing Model: It is a computer-like model explaining how people process the incoming information through sensory, working, and long-term memory structures.

Near Transfer: Near transfer is about procedures of task which are done more or less the same way each time they are performed.

Students as game designers: Students who lead, control, and develop their own games through programming and designing graphical environment.

Transfer of Learning: It occurs when students apply their knowledge and skills anytime and anywhere they need.

Worked Example: It exemplifies the solution of a problem with the inclusion of all steps fully.

CHAPTER II

II. LITERATURE REVIEW

This study was primarily built upon the principles of cognitive load theory. The focus is on backwards fading approach within a game-programming environment. Since the intervention group included two types of participants having different programming backgrounds, there was also a focus on the effects of near and far transfer issues. Programming education within various contexts constitutes the secondary part of the literature chapter. Therefore, this chapter consists of two main parts: *Cognitive Load Theory* and *Programming in Instructional Setting*.

2.1. Cognitive Load Theory

Cognitive Load Theory (CLT) was first introduced during 1980s and then developed and expanded by 1990s. Before meaningful learning starts, learners are drowned under a number of information and their interaction that needed to be processed simultaneously; at this point CLT is involved. The theory suggests that learning happens best under conditions that are aligned with human cognitive architecture, which consists of *sensory memory*, *working memory*, *visual* and *auditory channels*, and *long-term memory*. This model was first adopted by cognitive psychologists in the early 1950s and constituted the basics of CLT. This model explains the information processing in a computer-like fashion, which led some criticism about the nature of cognition. The model assumes that the information travel through sensory memory, working memory, and long term memory. Sensory memory is the initial stage of stimuli perception and it is associated with the senses. It retains stimulus which is important or interesting and move them into the working memory. This takes only a few seconds. It is well-known that working memory has extremely limited capacity that can hold about 7±2 meaningful items at a time (Miller, 1956). Working memory deals with the stimuli coming through our visual and auditory channels and the way we process the information by thinking, remembering and comparing. We process the stimuli by thinking, remembering and comparing in working memory. However, long-term memory is the fundemental knowledge storage and has vast capacity, which stores unlimited amounts of information from small facts to large complex procedures. These stored information are categorized and organised according to their manner and Piaget described this structure of memory as a *schema*. According to Piaget (as cited in Ultanir, 2012) development of learning process forged through *adaptation* which is the process of *assimilation* and *accommodation*. Assimilation is a process of dealing new object or situation using existing schema. Accommodation is a process of modifying existing schema while dealing with new object or situation in order to accommodate the new information. Some fundamental functions of *schema* are to provide organization and storage for information and to reduce working memory load (Sweller, van Merrienboer, & Paas, 1998).

Automation of schema is an important process, which occurs after sufficient practice, and therefore, with automation, working memory load can be reduced for other activities.

Moreover, it can become less difficult to perform similar tasks requiring automated schema. However, without automation, task can be completed eventually but the performance can be slow (Sweller, van Merrienboer, & Paas, 1998). For instance, each time we drive to work is an automated task and we can focus on another thing such as listening to the radio or talking to the person sitting in your car while driving. However, if you are driving to a new place, you are hundred percent focused on the map or GPS and you won't even listen to the radio not to distract your attention. Thus, for you it is a new task, rather than an automated one, so once you focused on the driving, there is almost no capacity for another task or activity. But it is possible to turn this new task into automated schema after sufficient practice. Therefore, learning takes place in working memory while new (automated) schemas and skills are stored in long-term memory.

According to the information processing model, learning occurs when all these processes are achieved: *attention*, *activation of prior knowledge*, *elaboration- rehearsal*, *encoding and retrieval*. When we receive new information, we pay our *attention* to sort out the information that is important. Once relevant information enters the working memory, it is integrated with pre-existing schemas in long-term memory. This transfer process of schemas from long-term memory into working memory is called *activation of prior knowledge.* The process of integrating new information and activated schemas is called *elaboration- rehearsal. Encoding* is to store new knowledge in long-term memory. Since thinking and learning processes take place in working memory, all these new and pre-exiting schemas must be brought back into the working memory when we needed. The process is called *retrieval.* As a result of all these processes, learning occurs. It is important to have achievement of successful retrieval of new knowledge and skill which is the basis for transfer of learning. The transfer of learning is directly connected to the schemas. For instance, when you learn a new computer application, your new schema will incorporate with the pre-existing schema and new procedure will be retrieved as long as the application has similar medium that you experienced before. Nevertheless, this process does not always end with successful learning, which may mean designing efficient instructional environments, and CLT can be one of the ways to manage efficient message designs.

"Cognitive load Theory (CLT) is a universal set of learning principles that are proven to result in efficient instructional environments as a consequence of leveraging human cognitive learning processes" (Clark, Nguyen, & Sweller, 2006, p. 7). According to the literature review, sources of cognitive load could be both human cognitive architecture and the environmental context (Choi, Van Merrienboer, & Paas, 2014; Chu, 2013). CLT is concerned with the usage of people's limited cognitive processing capacity to foster their ability to implement acquired knowledge and skill to the new situation (Lui, Lin, Tsai, & Paas, 2011). When it comes to the difficult and complex subject, CLT is highly relevant. According to Clark et al. (2006), there are three types of load: *intrinsic*, *extraneous* and *germane* (pp. 9-12).

Intrinsic load relates to the inherent structure and complexity of the instructional materials. Moreover, intrinsic load is relevant to how much different type of information the working

memory needs to deal with simultaneously. Extraneous load (ineffective load) relates directly to the weak design of instructional materials or messages. In extraneous processing learner engages learning material that does not support learning. Germane load (effective load) is relevant to the learning procedure that engages learner into deep cognitive process such as relating new subject to prior knowledge which depends on learner's motivation, prompts, and support during the lesson. The main goal of CLT is to make it convenient to learn complex tasks by efficiently using relationship between working memory and long-term memory (Paas & Ayres, 2014). Therefore, learning materials are designed to provide efficient learning performances or motivate learners by reducing intrinsic load which is to divide complex task into smaller parts and by reducing extraneous load which is to extract components that is unnecessary for learning. On the other hand, increasing germane load makes learning materials more effective with the inclusion of numerous relevant examples. These three cognitive loads are directly connected to the working memory.

2.1.1 Near & Far Transfer of Learning

Every day in our lives we do some action in the idea of transferring our knowledge without even realizing it. For instance, in our daily life when we buy a new mobile phone, we search a certain file based on the steps that we know from former mobile phone without any conscious thought. Another example is that after learning how to solve a math problem, solving a physics problem may become less challenging. Therefore, it is one of most important goals to be achieved by students in any educational case. If one can display his/her knowledge and skills later in a new environment, it is obvious that learning has occurred successfully. The transfer of learning is not a new matter in education. Its history is based upon studies of Thorndike and Woodworth on educational psychology perspective in the early 1900s (Thorndike & Woodworth, 1901). Several studies based on the transfer effect were concluded negatively in a preponderance way, but still there are studies that are concluded positively (Atkinson, Renkl, & Merrill, 2003; Price & Driscoll, 1997; Renkl, Atkinson, & Maier, 2000).

Educational transfer of learning can enable students to apply their knowledge and skills anytime and anywhere they need. It is also an important factor for motivation of students when they succeed the task in the way of transferring knowledge without realizing it, especially within the real life context. The whole process of transferring knowledge is to apply one's prior knowledge or experience to similar or different situations. There are two types of transfer of learning: *near* and *far* transfer. Near transfer is about procedures of task that are done more or less the same way each time they are performed. For instance, near transfer occurs when students solve problems in math exam that is similar to the problems they have solved earlier with their homework. Far transfer is about tasks, which require being adapted one's skill to a new situation each time that is more like transfer between contexts. Learning a new language or musical instrument can be shown as examples.

It is important to understand the circumstances facilitating the occurrence of transfer. There are some conditions for transfer of learning. Firstly, providing learners with boundless

practice of the performance may cause better transfer (Butler, 2010). Secondly, while learners are applying the strategy, they should also observe their own thinking process so that they can recognise later when and how to apply the strategy they have learnt (Perkins & Salomon, 1992). Thirdly, learners should study the new subject by thinking the metaphor or analogy (Misko, 1999). For instance, learner can understand the idea of a cell better by thinking of it as the tiniest structural and functional unit of an organism (Perkins & Salomon, 1992). Transfer may not occur as often as we hoped to be. However, if the instructor provides the convenient environment for transfer of learning, there is much better prospect for it to occur successfully. This may lead to meaningful learning in the end.

2.1.2 Worked-Example with Backwards Fading Effect

Designing instructional materials based on worked-example trends is quiet effective and widely used in especially in mathematics, physics, and computer science subject areas. Worked-example instruction provides learners with developing either their skills on complex problem solving or ability to transfer their learned skills. Moreover, significant amount of research has found that with the help of worked example instructions, learners are able to gain generalizable problem solving skills which can be retrieved to solve similar problems in the future in a way such as transfer of near and far problem solving (Atkinson, & Grose, 2004; Moreno, Reisslein, & Delgoda, Renkl, 2006). There are different ways that problem solving method can be integrated into the overall instructional design such as: *worked examples*, *completion examples*, *full-practice examples*, and *faded-worked examples*. Worked examples contain problem, full guidance for learner to solve the problem that presented as step by step instruction, and final answer (Atkinson, Renkl , & Merrill, 2003; Chen, Kalyuga, & Sweller, 2015). Therefore, learners are expected to study the worked example in order to be able to apply it to similar problems. According to the cognitive load theory (Clark et al., 2006), all elements are held in working memory during problem solving process. Solving problems without any guide can affect learning performance in an obstructive way, because of overloading the working memory. Therefore, literature shows us that worked-examples shows better learning solution than conventional problem solving with no guidance. Completion example is shown in a practice problem as some of the steps are represented as in worked example and other steps are to be completed by the learner. Full-practice example is a problem of which steps are fully solved by learners alone. Faded worked example is presented as fully worked-out example firstly and followed by another worked example with one step missing.

Transfer effect in worked-example promotes learning when both worked–out problem and full problem. Thus, while learners are moving from worked-out problem to full problem, extraneous cognitive load reduces, especially when the cases are integrated in an adaptive way (Najar, Mitrovic, & McLaren, 2016). Fading methods can facilitate the smooth transition between worked-out example and full problem. The role of fading in worked-example is to help learners to remember what they have learned in previous stage and apply it to their current task. Fading is to asking learners to gradually solve problems with increasing number

of steps after they get the full worked-out problem example. One of the advantages of fading is the expectation of fewer errors. There are two kind of fading steps according to their order: *backward fading* and *forward fading*. In backward fading, learners accomplish the last one step of the first problem, the last two steps of the second problem, and so on until they succeed all the steps. In forward fading, learners accomplish the problem solving as the first step rather than the last step, which is opposite to the backward-fading process.

For novice learners studying with worked-examples help them to experience or form relevant problem solving schemas, which provide them to gain expertise as a result. However, for learners who already have some schema in their long-term memory, working with worked examples will be gaining no benefit from it (Kalyuga, Ayres, Chandler, & Sweller, 2003). Based on the literature, in computer-based instruction near transfer is promoted example-problem practice with fading effect (Moreno, Reisslein, & Ozogul, 2009), but still the level of effect may depend on the prior knowledge of learners (Reisslein, Atkinson, Seeling, & Reisslein, 2006).

2.2. Programming in Instructional Settings

Unlike its predecessors in the history, today' programming languages appeal to a wide range of people. Thanks to block-based programming environments, children have opportunities to get to know coding concepts in their early lives. There are various programming environments designed for programming education. Although learning how to code is a popular issue, developing a complete program is still a challenge, especially for novices. With the increased usage of computers at universities, elementary schools and even preschools, recently people have changed their approach towards the integration of computers into the current settings. Computers are now considered as daily tools like telephone or washing machine. The perception of using computers in everyday situations including education has been changing. Modern educators have started to consider computers as creative technologies that can be used for any subject area. Because of being a fundamental part of computer technologies, programming languages are considered as a subject area to be explored and taught in a creative way with the inclusion of computers. Its challenging nature, programming courses have been increasing rapidly. Since then, value of text-based programming language has been increasing in education.

Text-based programming education can be explored in two categories: software engineering perpective and educational perspective (Robins, Rountree, & Rountree, 2003). Software engineering studies focus on programmers and practical implications such as how to develop software. On the other hand, programming education is now one of the main contents for modern curriculum as well as still being part of any engineering education. In order to have generations with 21st century skills, such an educational perspective may facilitate further skill improvements. Both perspectives can be integrated into curriculums and STEM approach can be an example for that case. With the increasing interest on programming

languages, it's started to take an important part in pedagogy from preschool to undergraduate students.

For teaching such complex subjects as programming, educators need to motivate their students in order to increase their interest. There are various techniques and methods that can be used in programming education. Some classical methods such as demonstration or drill-practice etc. can be used in programming education as well as more creative ones such as game-based methods. Teaching by demonstration is providing learners with examples and practice. Main principle of demonstration method is learning by doing. The teacher explains the subject in a very comprehensible way, while students are expected to repeat the steps the teacher has done. Demonstration method was found as an effective method in some fields such as chemistry (Omwirhiren & Ibrahim, 2016), agriculture science (Ekeyi, 2013), and physics (Miller, Lasry, Chu, & Mazur, 2013). While implementing demonstration method in IT field, instructors explain programming structures that will be needed in following tasks. Then, students are expected to complete the given tasks. In other words, through programming by demonstration, learners become automated on common tasks even without being trained (Guibert, Guittet, & Girard, 2005). Drill-practice method is also an effective method to obtain meaningful learning. It involves repetition of a specific task in order to help learners acquire the skill at their own pace. With drill and practice, learner practice the new material repeatedly until the reaction is automatic (Cooke, Guzaukas, Pressley, & Kerr, 1993). Nevertheless, using creative methods such as game programming/designing is also effective in programming education. By programming/making game, instructors inform learners on how to program a game using specific software (e.g. Scratch, Alice, StarLogo, etc.). It's important to give detailed information such as implementing sound or graphics or to show an example game programming code. In this way, student can build and design complete game with specific software by considering purpose, scenario, characters, and complexity. There are many studies showing the effectiveness of game designing as a method for novice learners on programming education (Akcaoglu & Kale, 2016; An, 2016; An & Cao, 2017; Garvey, 2015).

In addition to conventional programming environment, many other techniques and methods have been used in programming education. Teaching and learning programming through e-learning environment has promoted development of interactive educational software and role of technology in education (Maasoumi, Khazaei, & Shafiei, 2014). Using new devices such as mobile robots (e.g. LEGO NTX robot) in programming education increases attitude towards programming and also provides learner with more practical programming knowledge (Pasztor, Lakatos Török, & Lakatos Török, 2010). Lately, block-based programming education has become popular in early (elementary) education or introductory programming stages in university education. Beside the increasing popularity of block-based programming, new methods have visited the programming education. Because of the difficulty of teaching or learning programming language, some educators have started to teach block-based programming language in their early stage of CS education to make it more simple and tangible. When students get basic skills, educators teach text-based programming language in the next level or vice versa. With block-based programming language, students learn to

program effectively and efficiently without consuming their time. By presenting text-based programming graphically, students' learning attitudes and motivation can be improved. With these methods, students become more active in learning. Teachers guide their students by asking them to create meaningful and enjoyable projects that lead them to create games for learning programming. Consequently, by implicating block-based programming environment into programming education, "learning by teaching" and "learning by design" have become the main strategies for teaching computer programming (Lin & Yang, 2009). Therefore, many block-based programming tools are being used lately in education.

2.2.1 Block-based Programming

Block-based programming environments aim to make the coding process more concrete. It is known that learners have encountered many difficulties of text-based language due to its abstract nature. Unlike text-based ones, block-based programming provides the learners with well-designed graphical interfaces allowing simple dragging-dropping actions. Today, block-based language is aiming to ease those difficulties. The biggest advantage in block-based programming is to represent text-based programming language better with well-designed graphic user interface. Young learners have various opportunities to engage in programming through block-based environments. Generally, these environments are software that are used to design games. In a simple block-based programming environment, students combine code blocks in a meaningful way using a mouse and receive visual and sometimes audio feedback. Code blocks having similar shapes and colours can be put together in order to run in a functional way. Once a meaningful set of blocks was arranged, the learner can easily run and see the output instantly. Moreover, as the program starts running, one can follow the code sequence thanks to the highlighted blocks. In this way, debugging may become less challenging especially for the beginners.

Any instructional method can be integrated into programming education. Many studies presenting the variety of the methods are available in the literature. During early stages of learning programming, some concepts are difficult to learn. Especially for younger learners, doing robotics allows younger students to begin their programming education with more basic concepts. Using robotic is a successful way to teach introductory programming for students as well as for teachers (Koppensteiner, Vittori, Miller, & Goodgame, 2015). It is found that using robotic also draw attention of large number of female students. Moreover, Lye and Koh (2014) reviewed 27 empirical studies on programming in K-12 and higher education. K-12 students were using easy to use visual programming language to create digital stories and games. They found that, most of the studies reported positive outcomes. According to Kalelioglu (2015), teaching programming skills to K-12 students using Code.org is a new approach. This study attempts to investigate the effect of teaching code.org site on reflective thinking skills towards problem solving and also whether there is a gender difference in terms of students' reflective thinking skills towards problem solving. Overall

results showed that students developed a positive attitude towards programming, and female students showed that they were as successful as their male counterparts, and that programming could be part of their future plans.

In another study, Chapman and Jeannie (2015) used Python as the first programming language for students who have little or no prior programming experience. Participants were students from biomedical informatics graduate students, human genetics, molecular biology, statistics, and similar fields. During one semester, the researchers provided the students with sufficient programming experience that they could use programming in their graduate studies. Overall, they have reached the conclusion that Python was a good choice for participants who found it challenging yet useful and helpful. Ali and Smith (2014) did a similar study but with a different method. In their study, teaching an introductory programming language in a general education course for both Computer Science students and those from other major. During their study, they found out that dividing introductory programming course into two courses is the effective way for all students. First part is introduction to programming with Alice and it is effective for students from other majors. Second part is introduction to programming with Java programming and this is for CS student. Therefore, all students would be able to take more advanced, Object-Oriented Programming.

Lately, one of the popular approaches to teach programming is game-based instruction. Like well-designed games, well-designed instructions can easily motivate learners. From that perspective, the concept of students as designers emerged. In this way, learners design, then develop, and then play what they created. Educators of various subject areas apply game programming based approaches in their courses. Rich content and friendly interfaces of block-based programming software enable teachers as well as learners to practice and integrate coding into a range of teaching/learning experiences. Popularity of these software are increasing rapidly. Games are used in several different purposes in game programming based approach (Fernandez Leiva & Civila Salas, 2010). One of the purpose using game as a teaching environment is to motivate students. Other one is to use game as a tool for game design. Students complete a project to demonstrate knowledge and skill acquired during the course.

There are various ways to develop games depending on the complexity of the interaction, visual elements, scenario, and so on. For beginners, block-based programming environments can serve as a tool for developing games demanding neither sophisticated programming nor graphical design. This generation of block-based environments offers ready-to-use visual materials, but it is also possible to design inside or outside of the environment, which enhances the flexibility of the environments. Lego, Pet Park Blocks, Alice, Tinkertoy, Scratch, Starlogo series, and many more are well-known coding environments. For novice programmers, Scratch is preferred as a common block-based environment.

Instead of writing codes to tell a computer what to do, block-based programming uses drag-and-drop colored code blocks and creates flow chart. Through developing games in block-based environment, students have become the designer of their own games. Majority of youth

have already become consumers of video games, but with the approach of making games for learning, game players have started to program their own games and learn to work on related software and interface design (Hayes & Games, 2008; Peppler & Kafai, 2007). Thus, students become 'designers' or 'makers' of their own products. *Maker Movement* trend also supports this idea in educational setting as well as outs-of-school. Maker culture represents learning-by-doing with open-source software in social environment as well as technology-based environment.

2.2.2 Starlogo: The Next Generation (TNG)

Over the past years, game has a great influence on young programmers and beginners. Many simple video games were built by researchers with Starlogo since 1990s. This programmable modelling environment of Starlogo supports idea of learning by building in a way constructionist paradigm does.

StarLogo TNG which, is the next version of StarLogo, provides graphical programming language/blocks and three-dimensional world (Klopfer, Scheintaub, Huang, Wendel , & Roque, 2009). StarLogo TNG is client-based modelling and simulation software. It builds on the tradition of Logo-based languages developed by MIT Scheller Teacher Education Program and is free for all the operating systems in order to be used in instructional goal. Designing and developing StarLogo TNG project was started in 2002 and was released in July 2008. Its main aim was to reduce the barriers by making programming easy for novice learners, persuading younger learners into programming by making games, and providing with effective 3D games and simulations creation environment. At first, StarLogo TNG is designed for secondary students mostly who are new and willing to learn programming. StarLogo TNG is useful for not only programming but also science subject area. Therefore, StarLogo TNG with its easy programming environment is also appropriate for teachers. Both teachers and students are able to understand and develop their own 3D world (Begel & Klopfer, 2004). Starlogo TNG is the next and developed version of Starlogo. The major developments of Starlogo are StarLogoBlocks which represents code in the shape of puzzle pieces and Spaceland which provides rich 3D interface for building games (Begel & Klopfer, 2004, Klopfer et al., 2009).

2.2.2.1. StarLogoBlocks

StarLogoBlocks is a visual programming language interface where codes are presented as in the shape of puzzles. Moreover, "StarLogoBlocks is considered as an instruction-flow language, where each step in the control flow of the program is represented by a block." (Begel & Klopfer, 2004). Blocks are placed as a categorized palette of blocks on the left side of the interface and can be replaced by dragging onto workspace (see Figure 1). Code blocks are puzzle-piece shaped and coloured differently according to the programming function. One of the conveniences provided whilst using puzzle-piece blocks is software itself making sound of "click" when syntactically appropriate commands are placed together. Such a feedback can help the beginners to overcome hesitations, and shape for further development.

One of the important code blocks is *procedure* block. Procedure is useful to use a set of instructions or commands multiple times StarLogoBlocks also contain math operations such as brackets [], parentheses (), and others. Procedure, if-else, and math operations such as parentheses blocks are used in order to make Mario (main character) agent move (see Figure. 2). In workspace there is a canvas for each breed to drag and drop blocks to build programs.

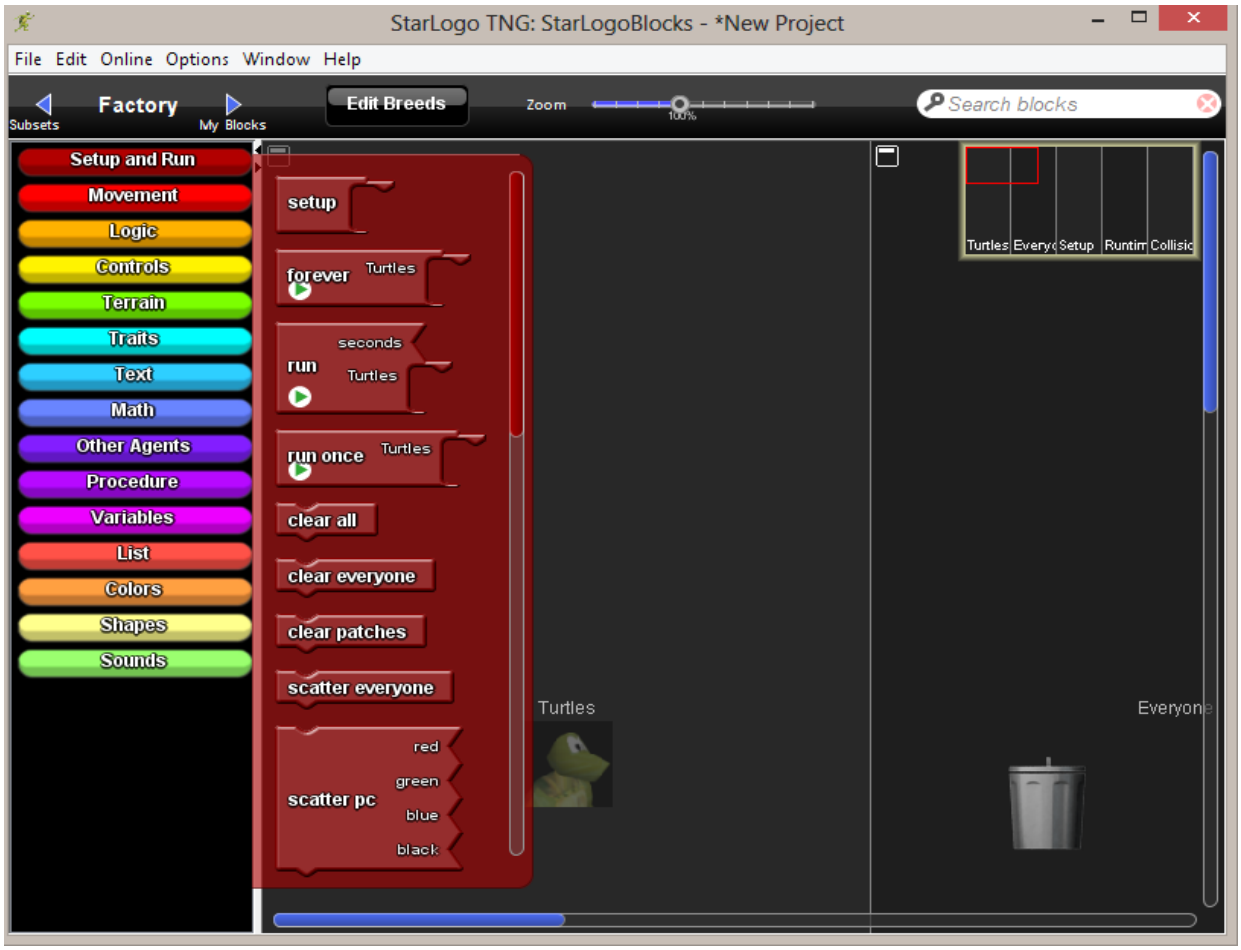


Figure. 1. StarLogoBlocks which contains all commands available to the programmer in an categorized palette.

Figure 2. Procedure of movement which contains If-Else and math blocks.

In the first canvas (see Fig. 3 from the left), the available space is used to declare variables and procedure for the main agent. In *everyone* canvas, the space is used to declare variables and procedure for other agent. The *setup* canvas is used for initiating conditions, i.e. with *setup* block one can create and place all the agents and set the initial values of variables. The *runtime* canvas is used to run all the code command in all canvas with the help of *run* and *forever* code blocks. When *forever* block is placed in *runtime* canvas, agents execute commands continuously. However, when *run* block is placed in *runtime* canvas, agents execute commands in certain time period that set by the programmer. In *collision* canvas, the programmer arranges instructions for agent. This is where the decisions including if agent collides or touches different breeds or each other in Spaceland by using *collision* block are set.

Figure 3. The view of canvas from the Mini map on the right top corner of the screen

Breed contains the entire agent and its default agent is named Turtle. While programming, the programmer can give command to all breeds. There are many types of agents in breed (see

Figure 4) and some of them are animals, basic-shapes, buildings, landscapes, letters, numbers, objects, people, and vehicles. One can choose the character according to his/her scenario. Programmer can also import 3D models as an agent into his project in the form of SketchUp (.kmz) and Collada (.zip) files.

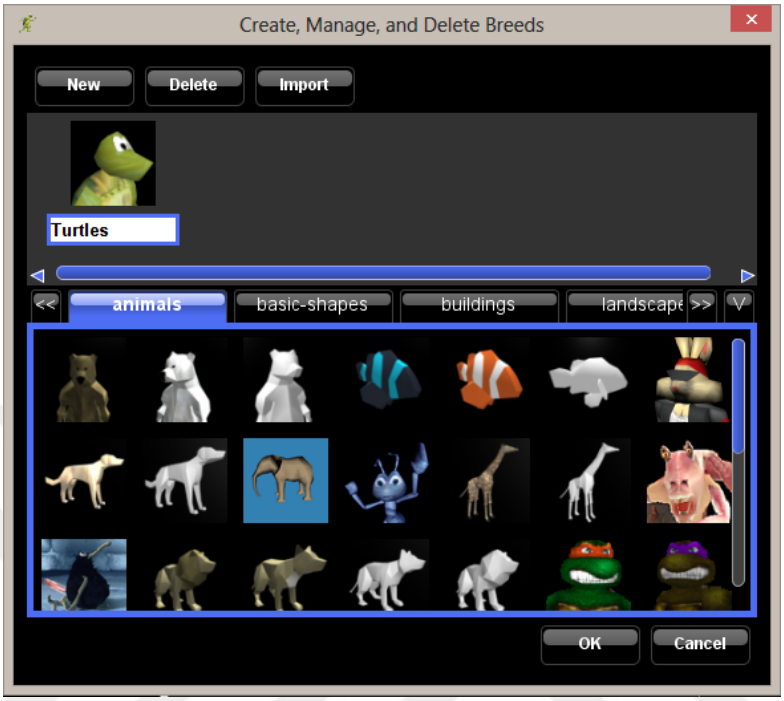


Figure 4. Edit Breeds

2.2.2.2. Spaceland

Another convenience of StarLogo TNG is the output visualised within a 3D environment (Klopfer et al., 2009). The tab named *Edit Terrain* (see Figure 5) allows users to choose certain region and create a hill or crater in the shape of round or square. Once student clicks the edit terrain, the terrain will show a grid to assist in selecting a region for editing. When students choose the region to edit, it will be highlighted in purple. Code blocks that need to run the game locate in tab labelled *runtime*. The *level* tab is about stages of game. Other tab labelled *drawing* consists of five drawing tools, namely: rectangle, circle, polygon, pencil, and image. When student click rectangle tool, it draws a rectangle and fills it in with the RGB colors. Circle tool draws a circle or oval and fills it in with the RGB color. Polygon tool draws a series of connected lines. The pencil tool draws free forms. With image tool, student can put 2D image (.jpg or .png) on the terrain.

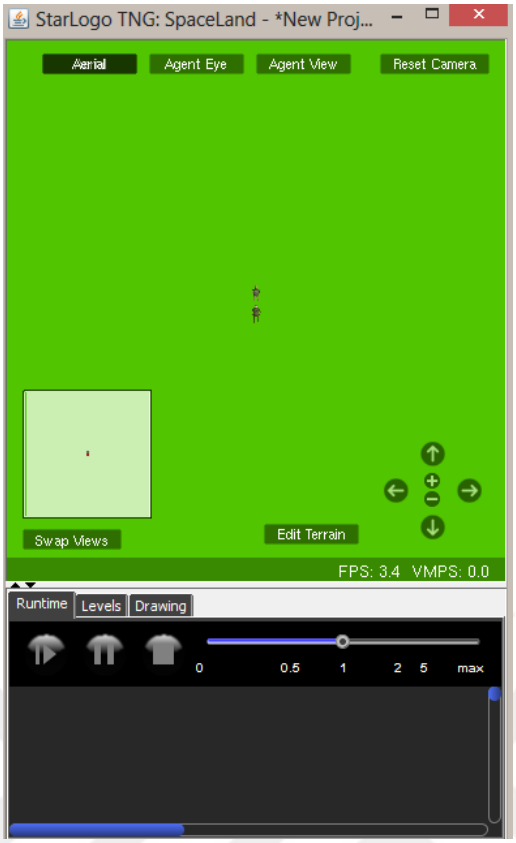


Figure. 5. Game output in a 3D Environment.

2.3. Backwards fading and Block-based Programming

Many studies have been done in text-based programming education with worked examples and fading effect. Gray, Clair, James, and Mead (2007) found that using faded worked example in programming education is effective. They indicated that fading worked example is an effective strategy for lowering cognitive load by providing gradual exposure to programming concept. Similarly, much of the worked examples with computer programming literature are available, but, nonetheless, fading examples have not been well explored or used in block-based programing education.

2.4. Summary

In order to find the answer of the main research question of this study, we need to dig related literature. The main research question is to find out whether the usage of backwards fading approach facilitates the transfer of block-based programming experience into 3D-game making in comparison to that of text-based programming experience. Moreover, other questions are about whether the formats of practice examples affect the 3D-game making performance of participants and also cognitive load. According to these questions, our literature consists of two main parts: *Cognitive Load Theory* and *Programming in Instructional Setting*.

The total amount of mental effort being used in the working memory is cognitive load. Cognitive load could arise from both human cognitive architecture and the environmental context (Choi, Van Merrienboer, & Paas, 2014; Chu, 2013). Human cognitive architechture is concerned with how people process the incoming information through sensory, working, and long-term memory structures. It's well know that short term memory holds only a small amount of information (7±2 meaningful item) at a time. Therefore, CLT is concerned with the usage of people's limited cognitive processing capacity to foster their ability to implement acquired knowledge and skill to the new situation (Lui, Lin, Tsai, & Paas, 2011). Students' ability to apply their knowledge and skills to the new situation is called transfer learning. There are two types of transfer of learning: *near* and *far* transfer. Near transfer refers to transfer of one's prior skill and knowledge between very similar contexts, whereas, far transfer refers to transfer between contexts. In educational setting, students obtain this transfer skill by practicing (examples) in similar or dissimilar contexts. It's found that using various example formats in education such as *worked-example*, *completion-example*, and *full-practice* is effective for learners to adopt new skill and knowledge. Transition from worked-example to full-practice, is called fading effect. The role of fading in example formats is to help learners to remember their prior skill and knowledge and apply it to their following stage. There are two kind of fading steps according to their order: *backward fading* and *forward fading*. In backward fading, learners accomplish the last one step of the first problem, the last two steps of the second problem, and so on until they succeed all the steps, whereas forward fading is vice versa.

Beyond doubt, programming course has become one of the main courses of this century. Value of text-based programming language has been increasing in education. However, for teaching complex subjects such as programming, there have always been many difficulties. Therefore, educators need to motivate their students in order to increase their interest. There are many methods that are used to teach programming languages. But last few years, in programming language education, using game has become popular. Nevertheless, block-based programming and relevant software have become more familiar in classrooms. With block based programming, instead of writing codes to tell a computer what to do, block-based programming uses drag-and-drop colored code blocks and creates flow chart. There are many block- based programming software that is used for designing games. Many studies have shown that using game making in introductory programming education is efficient. Through developing games in block-based environment, students have become the designer of their own games which makes students more active learner.

In this study, we used game making approach in a different way. Game making approach is not only used to give introductory programming concepts, but also used to explore how students transfer their prior text-based programming knowledge into their game making process. During the process of game making, worked-examples, completion- examples, and full-practice examples are used with fading effect. Students practiced how to use code blocks and design a game. At last, students were free to decide which kind of games they were going to design. Students' final projects (games) were developed by StarLogo TNG software. More importantly, in this study, the purpose of using games is not to teach programming, but to see

students' ability to transfer their previous programming knowledge into a new 3-D block-based programming software through practicing in three different formats (worked; completion, and full practice examples).

CHAPTER III

III. METHOD

In this chapter, we described our research methodology in details. Starting from our specific research design, we introduced participants, and then gave details of procedures with weekly activities. Finally, the instruments and data analysis methods were described.

3.1. Research Design

The current study includes a case of Computer Education and Instructional Technologies (CEIT) students' 3-D game making process. The case study method often involves simply observing what happens to a single participant or group of individuals such as a school class or a specific social group in its context. One of the advantages of this approach is the close collaboration between the researcher and the participant, while enabling participants to tell their stories (Miller, & Crabtree, 1999). Important key characteristics of case study are (a) it's flexible to cope with complex or dynamic phenomena of the real world and (b) its conclusions are based on multiple sources of evidence. Different kinds of data such as evidence, statements, products, and documents are linked together to support the conclusion. It's assumed that results driven from a case study are more realistic. On the other hand, data of case study is hard to interpret. It's believed that results are difficult to generalize or replicate the findings. This study includes one particular group of students who are enrolled to one course and one particular game making tool is used.

Block-based programming has been used widely for younger or novice learners recently. It's found that for novice learners starting with block-based programming environment is easier and more interesting whereas text-based environment is more sophisticated and challenging. Instructors have studied the learners' ability to transfer their prior text-based programming skill to block-based programming environment or vice versa (Armoni, Meerbaum-Salant, & Ben-Ari, 2015; Tabet, Gedawy, Alshikhabobakr, & Razak, 2016 Weintrop, & Holbert, 2017). In these studies, block-based programming tools are used as introductory programming environment. Learners are taught block-based programming such as Alice or Scratch initially in order to be able to work with text-based programming environment. Unlike these, learners are not novice in programming in this case. This study emphasizes on students who already have prior text-based/block-based programming skill and their process of transferring it into new block-based environment while designing 3D game.

The aim of this case study is to explore the outcomes of backwards fading. Moreover, students in this case have different programming backgrounds. Some of them (N=11) have previous experiences with block-based programming environments, whereas others are used to write codes in text-based environments (N=10). As a result, backwards fading method is expected to create different effect on learning outcome because of students' prior knowledge. In theory, backwards fading can support the expertise gaining (Gray, St Clair, James, & Mead, 2007). In this case, students both have programming background; thus, this case study does not aim to teach programming skills, but to observe how those different programming

skills are transferred into either similar or different programming environment. This study emphasis on backwards fading effect on students' game making process. In order to observe the students' game making process and find out the outcomes, this case study includes a series of embedded units of analysis (Yin, 2009). Although the main case is the students making 3-D games, their different backgrounds and the details in the backwards fading approach constitute the smaller units. The embedded design type of the current study was shown in Figure 6.

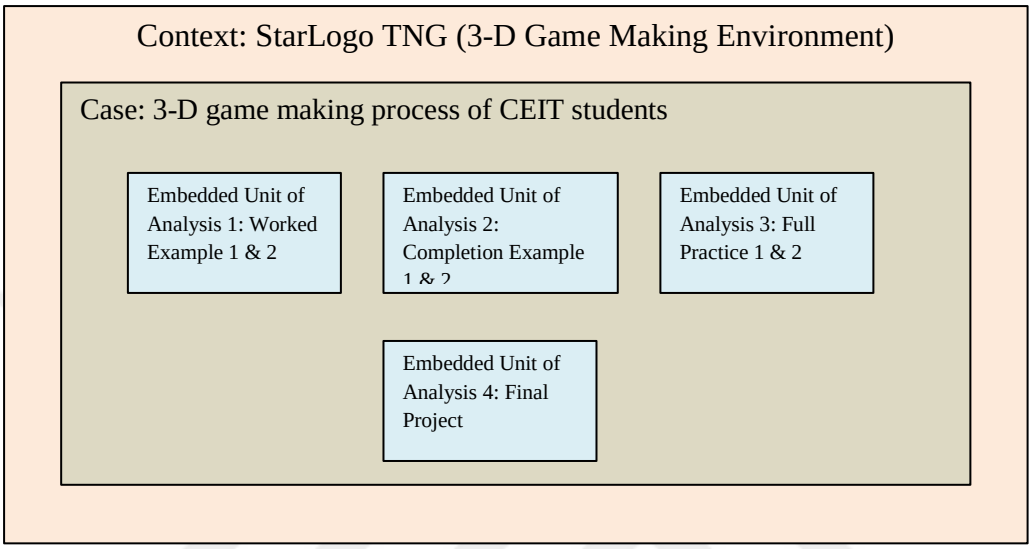


Figure 6. The Embedded Design of the Study (adapted from Yin (2009))

3.2. Participants

The participants of the study were CEIT students (*N*=21) enrolled to an elective course about games (see Table 1). The majority of them were males (*N*=19), which is very typical for that department. All participants have basic knowledge of abstract programming language such as Python, Java, etc. The participants had all completed a programming language course for two semesters. In the beginning of the semester, information about students' previous programming experiences was gathered, and it was found that almost half of them already experienced Scratch, which is a block-based programming environment. They know how to design 2-D scenes and define codes for simple games with Scratch. On the other hand, the rest of them had never used Scratch or any similar block-based environment, but wrote codes for simple games. None of the participants ever heard about StarLogo TNG, which is a block-based environment specific to 3-D game making. This case study explored how difficult for students to transfer their previous programming knowledge into block-based programming while designing 3D game using Starlogo TNG program. It's aimed to observe how different the game making process of students with block-based programming experience from that of students with text-based programming experience.

Table 1

Participants' Demographics

	N (21)
Gender	
Male	19
Female	2
Previous Experience with Block-Based Environment	
Yes	11
Scratch	11
Alice	10
No	10
Previous Experience with Text-Based Environment	
Yes	21
Python	21
PHP	1
JavaScript	3
No	-

3.3. Procedures

This study was conducted in Ondokuz Mayis University in Samsun, Turkey. The course title was "E-Game Based Learning". The class met every week for 180 min for one semester. The students in the class created computer games using StarLogo TNG. StarLogo TNG is client-based modelling and simulation software that teaches students and teachers how to design and create their own 3D games and simulations for understanding complex systems. It's assumed that game design is one of beneficial method in teaching programming language. Designing games in 3D environment is more appealing and realistic which is familiar with everyday life experiences (Begel, & Klopfer, 2004). *Starlogo TNG* is not the only 3D game making tool. *Alice* is also 3D modelling and programming environment. However, its level of sophistication is higher which makes it more appropriate for advanced learners (Begel & Klopfer, 2004). *StarLogo TNG* is a game making tool to motivate novice learners and allows them to begin programming immediately without requiring any level of text-based programming knowledge (Smith & Dunchan , 2011). In this study, StarLogo TNG was appropriate for all participants in terms of their prior programming knowledge.

The tasks were all about creating educational 3-D games after taking few weeks of classes based on the basic knowledge about Starlogo TNG program. Tasks were completed at a computer laboratory, where each student had the opportunity to work individually. All tasks

were the same for all participants and provided within the same sequence by the same instructor. Since the participants possess basic programming skills, the instructor did not go into details of concepts and commands. During the first 6 weeks, students were provided with theoretical information about electronic games, and then at the seventh week, they were given basic information about Starlogo TNG program via PowerPoint slides including tips about how to work with the interface. The next two weeks, students were shown how to use some basic code blocks including *procedure block* and *if-else blocks* in the beginning of the course. "Worked-example" method was applied during those two weeks. The researcher showed every single step of two simple three-dimensional games created with Starlogo TNG and students were asked to make the same games. The following two weeks, the researcher showed students some other code blocks including *collision block,* etc. In this week, "completion-example" method was used during the course. Like the previous two weeks, the researcher showed the game but incomplete forms of two games were given to the students. They were asked to do the uncompleted parts of the game by themselves. During the last two weeks, the researcher gave two game scenarios to the students and asked them to create games according to the scenario. "Full-Practice example" method was used during the two classes. One of the scenarios was easy and other one was more difficult. All the games were collected during these six weeks to evaluate them according to the rubrics created specific to each week. At the end of the semester, participants were asked to create their own games in two weeks and they were collected as a final project. The procedures were summarized on Table 2.

Table 2

Weekly plan of the study

	Content	Method	Game
Week-1-6	Educational Game	Presentation	---
Week-7	Introduction to Starlogo TNG	Presentation + Starlogo TNG program interface	Movement of Main Character
Week-8	Logic Blocks : If-Else	Worked Example	Mario
Week-9	Logic Blocks : If-Else	Worked Example	Treasure
Week-10	Collision Block	Completion Example	Running away From the Bears
Week-11	Collision Block	Completion Example	Catching crashed cars
Week-12	Summary	Full-Practice	Red Riding Hood: Collecting Carrots
Week-13	Summary	Full Practice	Crossing Bridge (Collecting coins)
Week-14-16		Students' Final Project	

3.4. Weekly Examples

3.4.1. Introduction to Starlogo TNG software

This week researcher prepared presentation to give students main knowledge about StarLogo software and what kind of games user can create. Six example games created with StarLogo TNG also presented to the students. Therefore, students had the main idea about code blocks and what they are going to deal with following six plus three weeks. After the presentation, researcher showed main content of a game, which is "Movement of Main Character". In this game, it is only about how keyboard control is provided by using code blocks. When player press the keyboard-up button, main character goes forward. For keyboard-right and keyboard-left buttons, main character turns right and left. For keyboard-back button, the main character goes back. Whilst researcher was showing the game step by step, students were simulating.

3.4.2. Worked Examples: "Mario" and "Treasure" Games

At the beginning of each session, participants were given additional information about some obligatory code blocks, which were going to be used in the following two games such as *Setup* code block to create main role of the game, *Scatter PC* and *Run* code blocks. Afterwards participants are shown every step of game. Mostly "If-Else" code block was the main task of these two sessions. *If-Else* code block is used to give movement to the main role with the control of keyboard.

Game-1 Mario: Mario is the main character of the game. Game ground is divided in equal squares. Colours of the squares are red, blue, and black and they were all placed randomly on the ground. When Mario is created, size and position of the Mario are set. Size of main character was set bigger than default size of it. Total duration of the game is 200 seconds. During the game, player has no control over main character. Once game starts, Mario starts to move forward by itself. When Mario walks on the "red" square, Mario turns right and size of it increases by one point. When Mario walks on the "blue" square, Mario turns left and size of it decreases by one point. When Mario walks on the "black" square, Mario turns 180 degrees and Mario continues to move forward. This loop continues in two hundred seconds. Code blocks and 3-D world are shown below (see Figure 7-8).

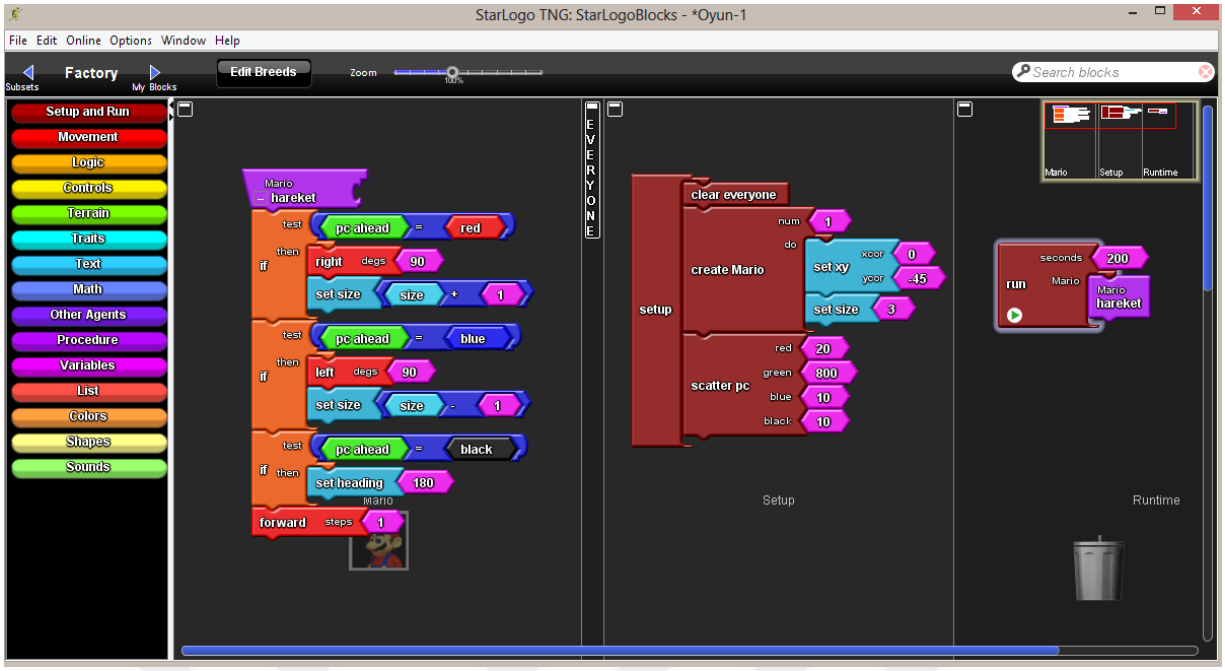


Figure 7. Code block of "Mario"

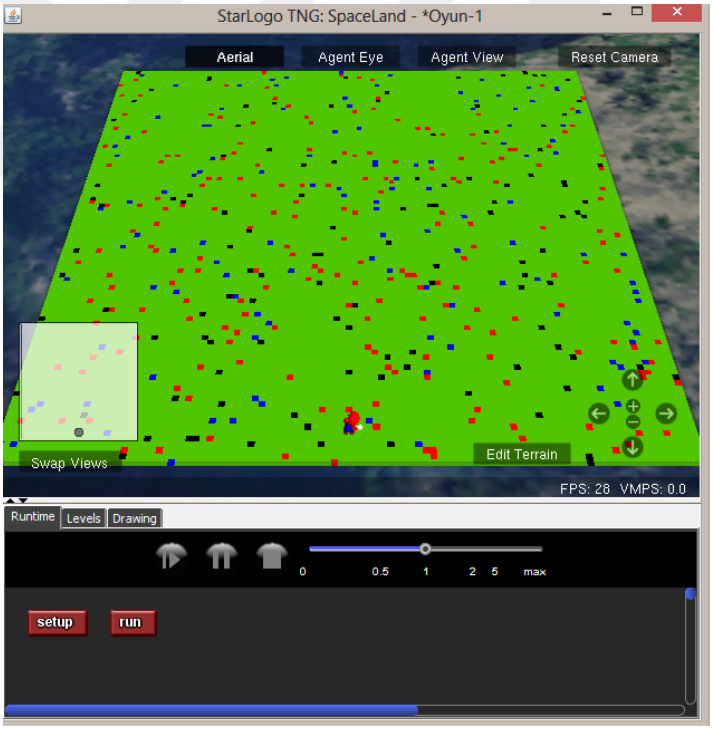


Figure8. 3-D environment of the game "Mario"

Game-2 Treasure: "Treasure" game has an additional code block, which is *Collision*. This block is used to show the result of an action when main character hits one object. Scenario of this game consists of a man, a house, trees, and treasures. The man is the main character of the game. The house and trees are located randomly in 3D game environment. In addition, tree is one of the agents that exist in Breed section of StarLogo. The house is designed by participants with SketchUp program and uploaded into game. SketchUp is a 3D modelling

program, which is similar to 3Ds Max program, but easier. The participants had experience to work with 3Ds Max. Models designed by 3Ds Max program can't be imported into Starlogo TNG program, so they designed house using SketchUp (see Figure 11). There are three different coloured treasures: red, yellow, and black. Movement of the main character is controlled with keyboard buttons (up, down, right, and left buttons) by player. The duration of the game is also 200 seconds. In this period of time, the main character has to collect the treasures as many as possible. When main character collects yellow treasure, he gets hundred points. When main character collects red treasure, he gets fifty points. When main character collects black treasure, he loses fifty points. Code blocks and three-dimensional world are shown below (see Figures 9-10).

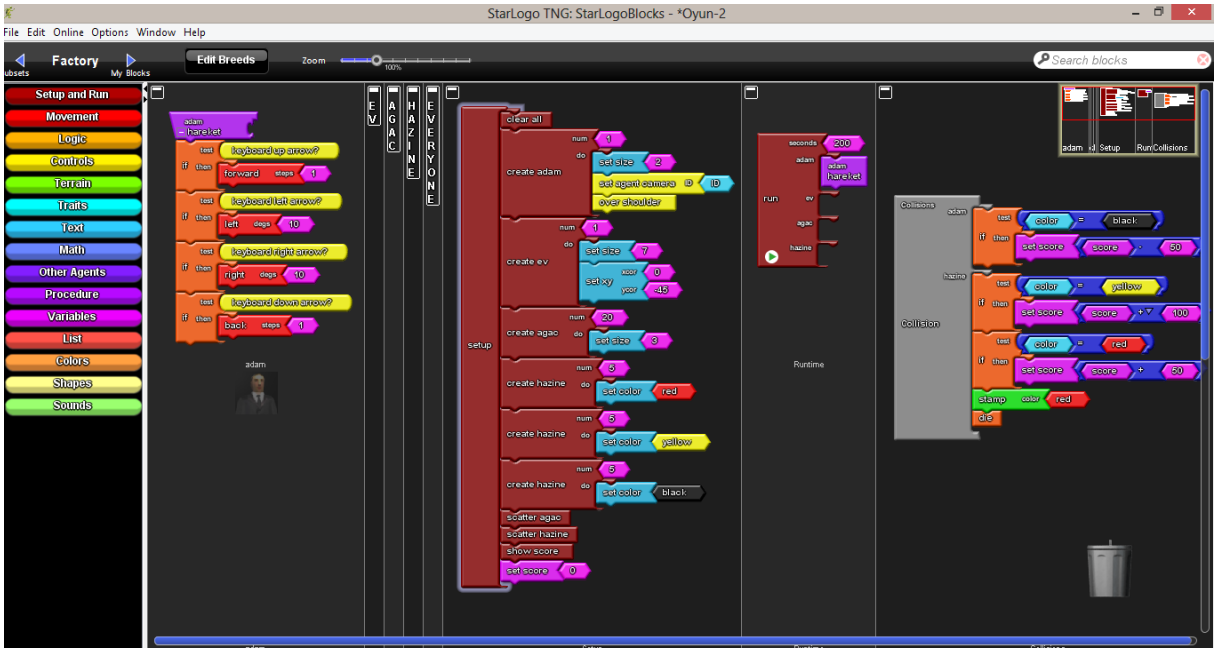


Figure 9. Code Blocks of "Treasure"

Figure 10. 3-D environment of "Treasure"

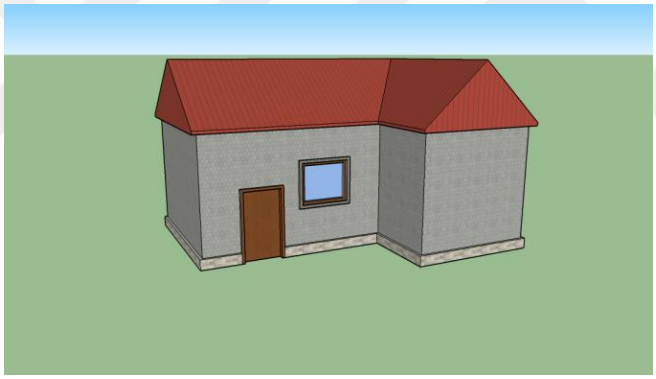


Figure 11. 3-D Model designed by SketchUp

3.4.3. Completion Examples: "Running away from the Bears" and "Catching crashed cars" Games

During these two sessions, participants were also given additional information about some obligatory code blocks which will be used in following two games. Firstly, participants are shown every steps of the game with code blocks. Since the participants already had knowledge about *if-Else* block, *Collision* block was the main task of the 10^{th} and 11^{th} weeks. Therefore all the participants get half-prepared game to complete the missing code blocks.

Game-1 Running away from the Bears: Like previous games, this game also has *If-Else* and *Collision* blocks and the additional task is to give control on the other role in the game with the code blocks. The scenario of this game consists of Simpson (main character), bears, trees and a house. The house, trees, and bears are located randomly in 3D game environment. This game has no time limit. When game starts, Simpson has to go to the house without getting caught by bears. If a bear catches Simpson, then the player loses. If Simpson gets to home

without getting caught, the player wins and it says "Tebrikler! Ayiya yakalanmadan eve geldiniz!" ("Congratulations, you made it without getting caught by bears") which pops up above the house. Full version and a handout version of code blocks are shown below (see Figures 12, 14).

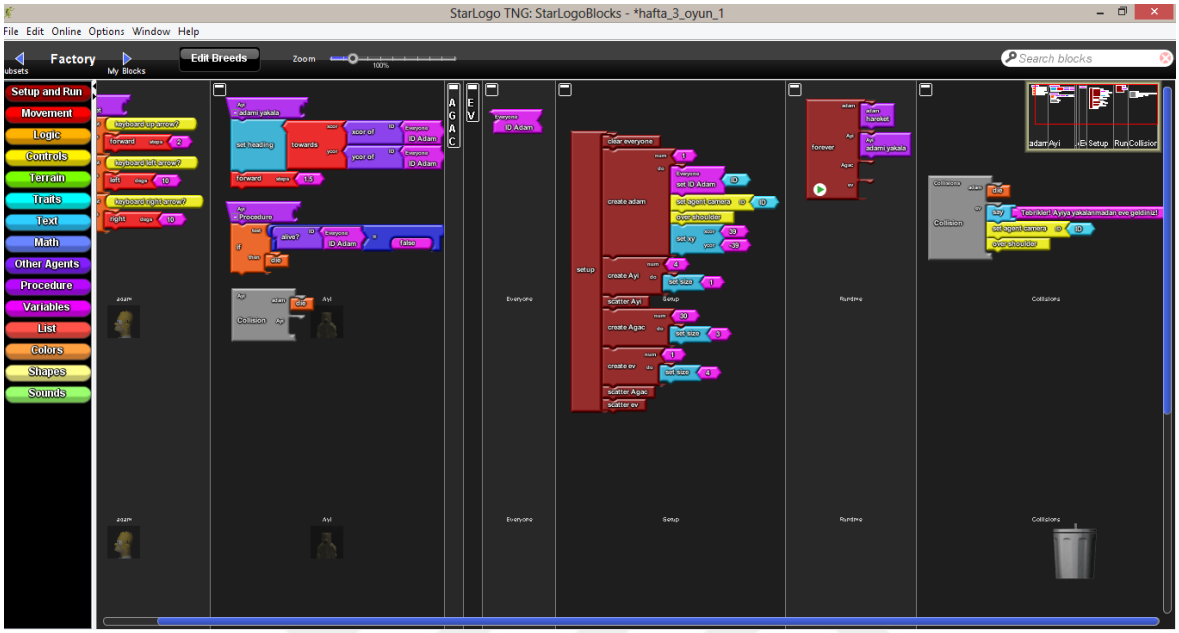


Figure 12. Full Code block of "Running away from the Bears"

Figure 13. 3D environment of "Running away from the Bears"

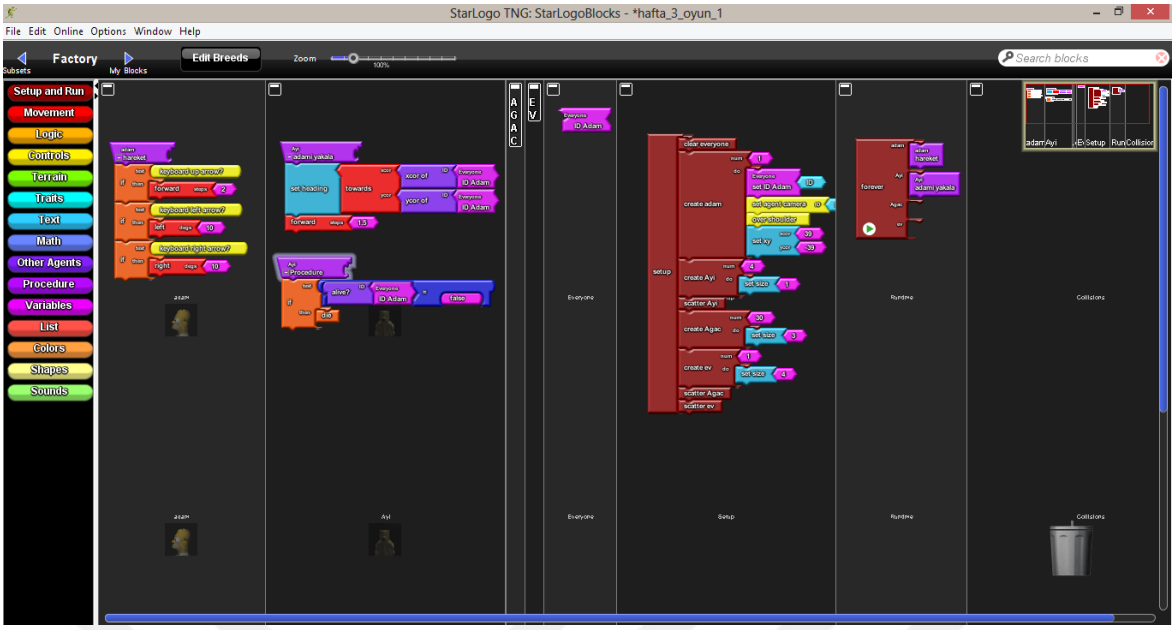


Figure 14. Handout of "Running away from the Bears"

Game-2 Catching crashed cars: The scenario of this game consists of police car (main character), two different damaged cars, one undamaged car, people, and trees. Main task of the game is to catch the crashed cars in thirty seconds. This information is shown on the bottom left corner of the screen. Both cars and people are moving continuously. Therefore challenge is to catch the crashed car without hitting person, undamaged car or trees. One of the two damaged cars is more damaged and the other is less damaged. When the police car catches the more damaged car and less damaged car, the damaged car disappears and the player gets ten points for more damaged car and five points for less damaged car. Every time when police car hits the person, undamaged car and trees, player gets minus points: minus five point for undamaged car, minus three point for trees, and minus ten point for person also the person dies after the crash. As a result, player is required to catch as much as possible in thirty seconds. Full version and a handout version of code blocks are shown below (see Figures 15, 17).

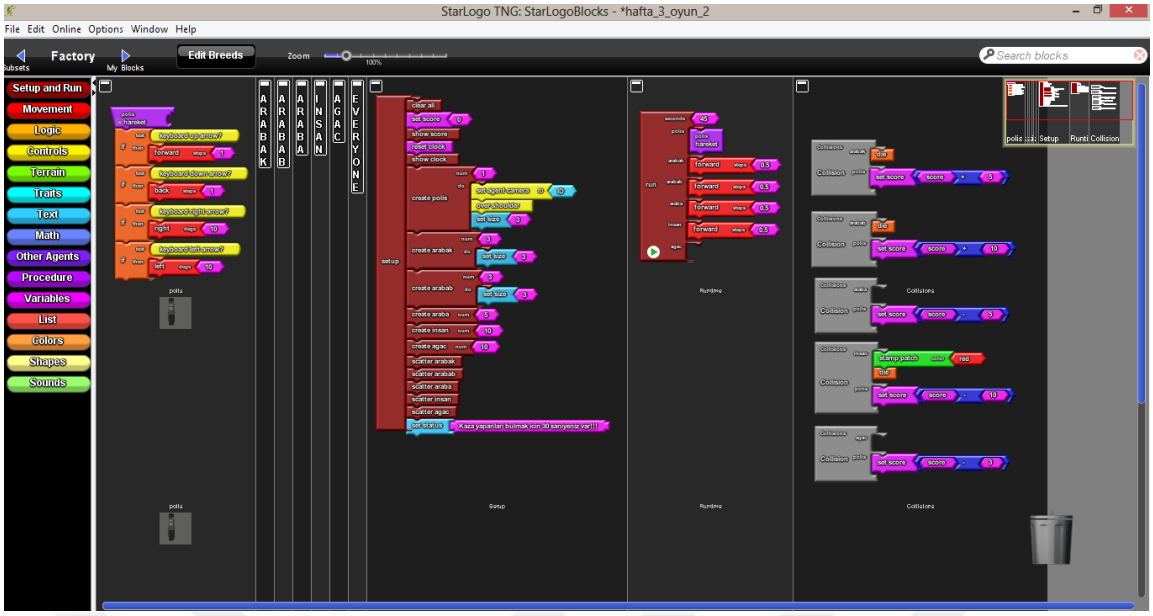


Figure 15. Full code block of "Catching crashed car"

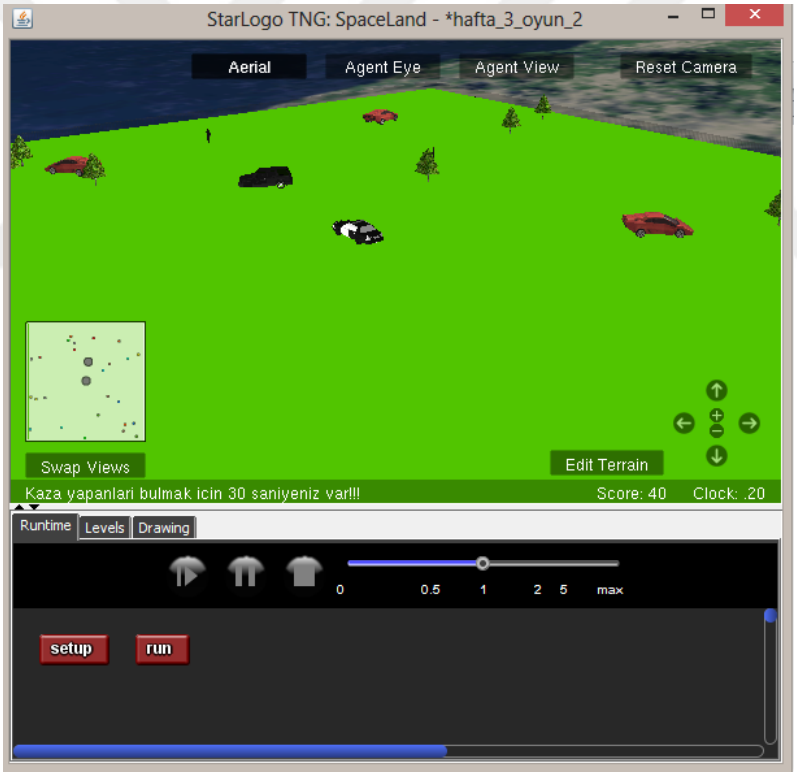


Figure 16. 3-D environment of "Catching crashed car"

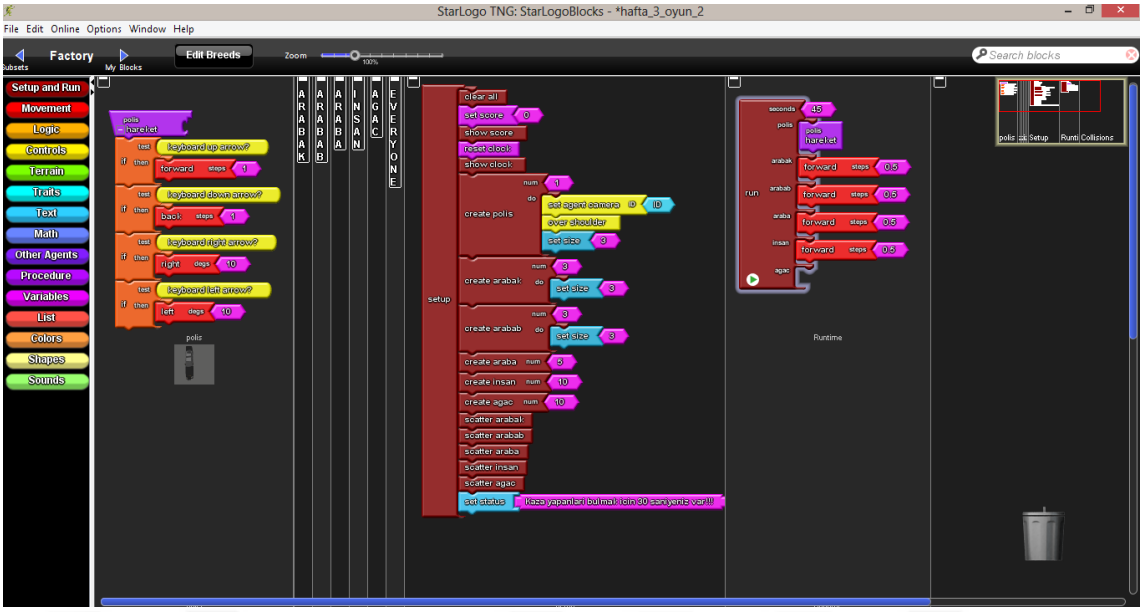


Figure 17. Handout of "Catching crashed car"

3.4.4. Full Practice Examples: "Red Riding Hood: Collecting Carrots" and "Crossing Bridge (Collecting coins)" Games

During these sessions, students get handouts. Each handout contains scenario of the game and guidelines. In order to apply full-practice method, students didn't get any help before they started to work on the handouts. Only task is to design a game, which is described on the handout.

Game-1 Red Riding Hood-Collecting Carrots: Scenario of the game is about Red Riding Hood, who is collecting carrots for her grandmother's dinner. It consists of Red Riding Hood as a main character, her grandmother's house, carrots, and stinging nettles. When she hits the carrot, she becomes bigger and carrot disappears. When she hits the stinging nettle, she becomes smaller. As soon as all the carrots are collected, she goes home and game ends. *Collision*, *Set Size*, and *Die* blocks are the most required code blocks of this game. (see Appendix A)

Game-2 Crossing Bridge (Collecting Coins): Scenario of this game is about a car, which collects coins while going across the bridge. It consists of a car as a main character, a bridge, cars in motion, and turning coins. The bridge is placed between two mountains and there is a river under it. The bridge has two-row car road. The participants are asked to design the bridge and coin by using SketchUp. Other cars are continuously moving and each turning coins are placed differently on the bridge. While the car (main-role) is going across the bridge, it gets point for hitting the turning coin or it loses point for hitting the other car. When the car gets to another side of the bridge, the game ends. *Collision*, *Set Size*, *Die*, and *Run* blocks are the most required code blocks of this game. (see Appendix B)

3.4.5. Final Projects

After all these sessions, participants are asked to design their own games using code block they have learned so far. It was free to choose the desired scenario.

3.5. Instruments

The data collection instruments of this study included students' weekly activity difficulty ratings, rubric for each game design (see Appendix C2-C8) and an online form for participants' views (see Appendix D). In order to calculate the efficiency metric, students' self-ratings for the difficulty of the activity were gathered. At the end of each lesson, students were asked to rate the difficulty on the scale ranging from 1 (very low) to 9 (very high) as suggested by Paas (1992). To calculate efficiency of each example, equation shown below was used.

$$\frac{\text{Average Performance in Z Score} - \text{Average Difficulty Rating in Z Score}}{\sqrt{2}}$$

Efficiency metric is calculated by subtracting mental load (ML) from performance (P) outcomes.

$$E = P - ML$$

Since students performed a different activity per week, the assessment of the outputs was guided by rubrics prepared by the researchers. All criteria were in line with the instruction and the examples. Rubrics also helped researchers to reach some information on each task such as completion status, difficulties, transfer of knowledge, and examining if these methods are appropriate for programming education. Sample of our rubric consists of six fundamental statements. Each rubric has some supplementary statements depending on the session. Each statement has three grading: not tried, incomplete, complete. They were all graded and converted to a total point per week. An example rubric is summarized on Table 3.

Table 3

Rubric Example

	Code Blocks	**Not tried**	**Incomplete**	**Completed**
1.	Edit Breeds			
2.	Creating *Procedure* Block			
3.	Using *Logic-If* Block (movement)			
4.	Math Block			
5.	Creating *Setup* Block (Creating characters and other objects)			
5.1.	Set xy & size			
6.	Creating *Run* Block			
7.	Terrain Screen			

Lastly, the online participant view form helped us to have insights of participant's feelings and opinions on Starlogo TNG program. Moreover, it helped to understand their experiences during three-dimensional game design with code blocks. This form had seven main issues. Followings are a few sample questions of online form:

- In your opinion, what are the top 3 difficult issues with regards to programming languages?
- What kind of difficulties did you experience in creating codes that you already know in another platform while coding with Starlogo?

3.6. Data Analysis

The present study includes both qualitative and quantitative data, and thus the data analysis were completed through the use of different methods. For the first research question, we used the scores of final project. The comparison of mean scores was done by running an independent t-test. In order to see the effects of examples' formats, we ran Friedman's ANOVA due to violated assumptions of parametric tests. For that analysis the scores of weekly activities were used: 1^{st}-2^{nd} worked-examples; 1^{st}-2^{nd} completion examples; 1^{st}-2^{nd} full practice examples; and 1 final full practice. The same analyses were done for perceived difficulty ratings across different formats. Students' difficulty ratings were transformed into Z-scores, and then efficiency metrics were calculated for each activity. In this way, the efficiency metrics were calculated to see how efficient the inclusions of examples were. Content analyses were conducted for students' online forms. The revealed themes were used to answer the last research question. Table 4 shows a sample coding scheme of these data.

Table 4

Sample Coding Scheme for Coding with Starlogo TNG

Theme		*f*
Coding with blocks:		
	easy	4
	Funny	1
	Sometimes complex	1
Stacking is challenging		1
Stacking requires complex thinking		1
Finding command blocks		2
Similar with Scratch		1
Time consuming		1

CHAPTER IV

IV. FINDINGS

In this chapter, the results of the data analysis are presented. The data were collected and analyzed in response to the problems stated in chapter 1. The first three sections present the result derived from analysis using ANOVA and correlation analysis. Fourth section presents efficiency metrics in terms of cognitive load. The final section presents the content analysis of qualitative data.

4.1. The Role of Backwards Fading in the Transfer of Block-Based Programming Experience into 3D-Game Making

The following hypotheses were tested to explore the first research question:

- H_0: There are no significant differences between the final scores of students having text-based programming experience and those of block-based programming experience.
- H_a: The final scores of students having block-based programming experience are higher than the final scores of others.

For the analyses, students' final projects were evaluated according to a rubric (see Appendix C-8). Those scores are assumed as dependent variables and the grouping variable was the condition of having previous experience on block-based programming. First, to test the null hypothesis, we ran independent t-test. Before the main analysis, we tested the assumption of normality and found that the distributions tend to be normal (see Table 5), which allowed us to continue with one of the parametric tests.

Table 5
Tests of Normality for Groups

Tests of Normality						
	Kolmogorov-Smirnov[a]			Shapiro-Wilk		
	Statistic	df	Sig.	Statistic	df	Sig.
Group 1	.270	10	.047	.841	10	.056
Group 2	,244	10	.093	.888	10	.160

Although the descriptives showed that students with a previous block-based programming experience (M=83.5, SE=3.3) had lower scores than the ones with text-based programming experience (M=85.5, SE=2.2), such a difference was not found significant with respect to t-test results $t(18)$=-.503, p>.05 with a small-sized effect r=.12. As a result, we fail to reject the null hypothesis.

Table 6
Independent t-test Results

		Levene's Test for Equality of Variances		t-test for Equality of Means				
		F	Sig.	t	df	Sig. (2-tailed)	Mean Difference	Std. Err. Diff.
Final	Equal variances assumed	.643	.433	-.503	18	.621	-2.00	3.979
	Equal variances not assumed			-.503	15.4	.622	-2,00	3.979

4.2. The Effects of Practice Examples' Formats on 3D-Game Making Performance

The following hypotheses were tested in order to understand the effects of example formats:

- H_0: There are no significant differences among the Worked Example, Completion Example, and Full Practice scores of participants.
- H_a: There are significant differences among the Worked Example, Completion Example, and Full Practice scores of participants.

This section presents the quantitative analyses of data set, which includes the scores gathered for each different example type. While grouping variable for this analysis was the conditions of worked example, completion example, and full practice, the scores of students' weekly games were the dependent variables. In order to see the significant differences among data sets, Analysis of Variance (ANOVA) was an appropriate test, but before conducting the test, a group of assumptions needs to be tested. First, we checked the normality assumption.

In order to see whether the distributions are normal or not, the histograms and Skewness-Kurtosis values were examined. Some of the distributions, such as worked example, were skewed to right as observed on both histograms (see Figure 18) and Skewness values. Moreover, either Kolmogorov-Smirnov or Shapiro-Wilk tests reported that all of the scores were significantly non-normal ($D_{WE}(21)=0.28$, $D_{CE}(21)=0.34$, $p<.001$) (see Table 7). Despite the full practice scores' tendency to be normal ($D_{FP}(21)=0.17$, $p>.05$), the majority of the distributions are not normal.

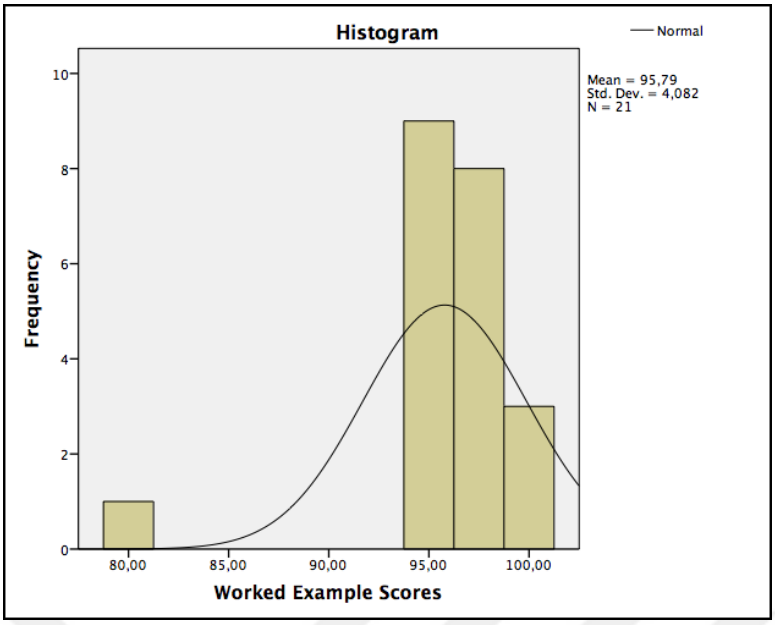


Figure 18. Histogram of Worked Example Scores

Table 7

Normality Tests for Example Formats

Tests of Normality

	Kolmogorov-Smirnov[a]			Shapiro-Wilk		
	Statistic	df	Sig.	Statistic	df	Sig.
WORKED	.283	21	.000	.660	21	.000
COMPLETION	.335	21	.000	.761	21	.000
FULL	.174	21	.095	.935	21	.171

Since the normality is one of the assumptions of ANOVA, violating it may lead incorrect results. That is the reason why we ran Friedman's ANOVA, a non-parametric alternative based on ranks (see Table 8). The results of the analysis indicated that the 3D-Game making scores of students significantly changed over different example types ($X^2(2)=35.11$, $p<.001$), and therefore we reject the null hypothesis. In order to understand the details of differences, we conducted post-hoc analysis.

Table 8
Distribution Ranks

	Mean Rank
WORKED EXAMPLE	2.31
COMPLETION EXAMPLE	2.69
FULL PRACTICE	1.00

Wilcoxon signed-rank tests were utilized as post-hoc with Bonferroni corrections (α/3=.0167 level of significance). Follow-up results were all resulted in significant values. Based on negative ranks the scores changed from Worked-Example to Completion-Example conditions. The scores of game design were significantly higher in Completion-Example (*Mdn*=100) than those of Worked-Example (*Mdn*=96.5), *z*=-2.80, *p*<.017, *r*=-.35. Based on positive ranks, the scores changed through Full-Practice (*Mdn*=82.5). In other words, the scores of game design were significantly lower in Full-Practice than either those of Worked-Example, *z*=-4.02, *p*<.001, *r*=-.51 or Completion-Example, *z*=-4.02, *p*<.001, *r*=-.51. The overall comparisons and the results of Wilcoxon signed-rank test were summarized on Tables 10.

Table 9

Follow-up Ranks

		N	Mean Rank	Sum of Ranks
CE – WE	Negative Ranks	4[a]	3.50	14
	Positive Ranks	12[b]	10.17	122
	Ties	5[c]		
	Total	21		
FP – WE	Negative Ranks	21[d]	11	231
	Positive Ranks	0[e]	.00	.00
	Ties	0[f]		
	Total	21		
FP – CE	Negative Ranks	21[g]	11	231
	Positive Ranks	0[h]	.00	.00
	Ties	0[i]		
	Total	21		

a. CE < WE, b. CE > WE, c. CE = WE, d. FP < WE, e. FP > WE, f. FP = WE, g. FP< CE, h. FP> CE, i. FP = CE

Table 10
Wilcoxon Signed Ranks Results

	CE – WE	FP - WE	FP - CE
Z	-2.798[b]	-4.018[c]	-4.016[c]
Asymp. Sig. (2-tailed)	.005	.000	.000

b. Based on negative ranks.

c. Based on positive ranks.

4.3. The Effects of Practice Examples' Formats on Perceived Difficulty While Making 3D-Games

In this section, the following hypotheses were examined:

- H_0: There are no significant differences among the Worked Example, Completion Example, and Full Practice scores of participants' perceived difficulty.
- H_a: There are significant differences among the Worked Example, Completion Example, and Full Practice scores of participants' perceived difficulty.

Each week, as soon as students complete and submit their works, they were asked to rate the difficulty of weekly activities. The value ranged from 1 to 9. We calculate the average of activities for similar activities. In other words, we had 2 WE activities and 2 ratings for each, and then calculated the average to have one value. Before going further for the main analysis, we checked for normality because violating that assumption may cause serious errors.

In order to decide whether the distributions are normal or not, the histograms and Skewness-Kurtosis values were examined. Some of the distributions, such as completion example, were skewed to left as observed on both histograms and Skewness values. Moreover, either Kolmogorov-Smirnov or Shapiro-Wilk tests reported that the scores were significantly non-normal ($D_{CE}(21)= 0.20$, $D_{FP}(21)= 0.23$, $p<.05$), except for Worked Example ratings that are tending to be normal ($D_{WE}(21)= 0.16$) (see Table 11). However, we did not risk the overall results and continued with a non-parametric test named Friedman's ANOVA.

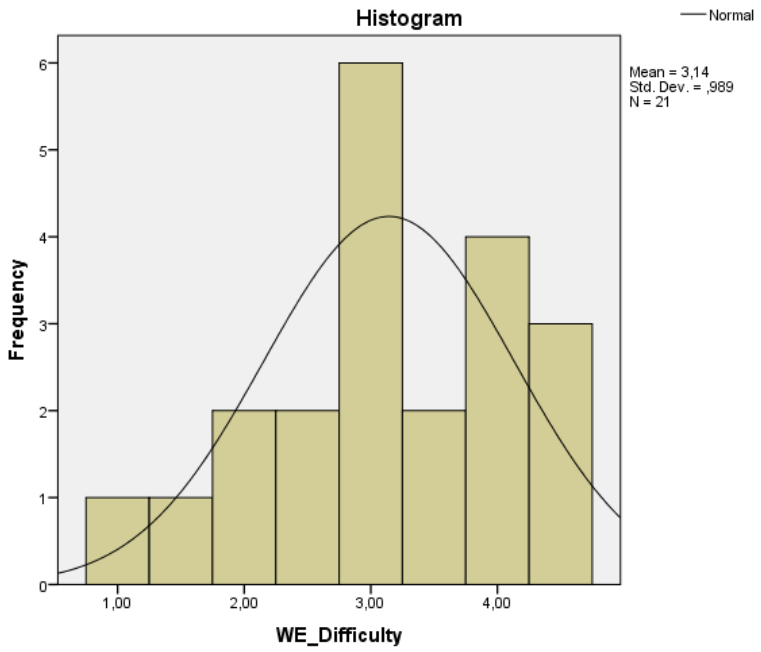


Figure 19. Histogram of Difficulty Ratings in Worked Examples

Table 11
Normality Tests for Difficulty Ratings

Tests of Normality						
	Kolmogorov-Smirnov[a]			Shapiro-Wilk		
	Statistic	df	Sig.	Statistic	df	Sig.
WE_Difficulty	.157	21	.192	.942	21	.236
CE_Difficulty	.199	21	.029	.891	21	.023
FP_Difficulty	.230	21	.005	.912	21	.059

The results of the Friedman'ANOVA generated meaningful differences across formats. Examining the mean ranks already pointed differences on mean ranks (see Table 12), but to decide the significance, just observing different rankings is not enough. The test statistics for this case showed that the perceived difficulty scores of students significantly changed over different example types ($X^2(2)=31.63$, $p<.001$), and therefore we reject the null hypothesis. In order to understand the details of differences, we conducted post-hoc analysis.

Table 12
Distribution Ranks

	Mean Rank
WE_Difficulty	1.74
CE_Difficulty	1.31
FP_Difficulty	2.95

Wilcoxon signed-rank tests were utilized as post-hoc with Bonferroni corrections (α/3=.0167 level of significance). Follow-up results were resulted in different significant values. It was found that students perceived the Full-Practice (*Mdn*=5.50) as significantly more difficult than either Worked-Example (*Mdn*=3.00, *z*=-3.00, *p*<.0167, *r*=-.88 or Completion-Example (*Mdn*=2.*50*, *z*=-2.50, *p*<.0167, *r*=-.87) based on negative ranks. On the other hand, based on positive ranks, it can be inferred that students did not find neither Worked-Examples nor Completion Examples more difficult in comparison to each other, *z*=-2.25, *p*>.0167, *r*=-.45. Tables 14 Summarized the results of post-hoc results.

Table 13
Follow-up Ranks

		N	Mean Rank	Sum of Ranks
CE - WE	Negative Ranks	14[a]	9.79	137
	Positive Ranks	4[b]	8.50	34
	Ties	3[c]		
	Total	21		
FP - WE	Negative Ranks	0[d]	.00	.00
	Positive Ranks	21[e]	11	231
	Ties	0[f]		
	Total	21		
FP - CE	Negative Ranks	1[g]	1	1
	Positive Ranks	20[h]	11.50	230
	Ties	0[i]		
	Total	21		

a. CE_Difficulty < WE_Difficulty, b. CE_Difficulty > WE_Difficulty, c. CE_Difficulty = WE_Difficulty, d. FP_Difficulty < WE_Difficulty, e. FP_Difficulty > WE_Difficulty, f. FP_Difficulty = WE_Difficulty, g. FP_Difficulty < CE_Difficulty, h. FP_Difficulty > CE_Difficulty, i. FP_Difficulty = CE_Difficulty

Table 14
Wilcoxon Signed Ranks Results

	CE - WE	FP - WE	FP - CE
Z	-2.254[b]	-4.025[c]	-3.991[c]
Asymp. Sig. (2-tailed)	.024	.000	.000

b. Based on positive ranks.

c. Based on negative ranks.

4.4. The Effects of Practice Examples' Formats on Cognitive Load

In this research question, we tried to understand the cognitive load students experienced during different formats of examples of game making. For this purposes, we used both performance scores (the average scores for each example type) and difficulty ratings (the

average ratings for each example type). In order to calculate cognitive load, efficiency metric was used. We transformed all values into z-scores, and then calculated efficiency values. Efficiency metric for worked example represents low efficiency with average performance value of 0.0000005 and an average difficulty value 0.0000010. In contrast, metric for completion example represents high efficiency with average performance value of 0.0000033 and an average difficulty value 0.0000019. However, the metric for full practice represents efficiency on line *E*=0 with equal average performance value and an average difficulty value of 0.0000005. Considering the places of efficiency values within the efficiency graph, WE condition can be referred as low; CE as high, and FP as neutral in terms of efficiency. The summary of the efficiency scores can be found on Table 15 and also in Figures 20, 21, 22. Moreover, the averages of all practice problems of two groups were taken and are compared. Considering the cognitive loads of all practice problems depending on two different groups of students: (group-1) students who had experience with block-based programming language (Scratch), (group-2) students who only had experience with text-based programming language (Python), efficiency value of all practice problems of Group-2 was higher that the efficiency value of other group (see Figures 23, 24).

Table 15

Efficiency Metric

Example Format	Performance	Mental Effort	Efficiency
WE	0.0000005	0.0000010	-0.00143
CE	0.0000033	0.0000019	-0.00095
FP	0.0000005	0.0000005	0

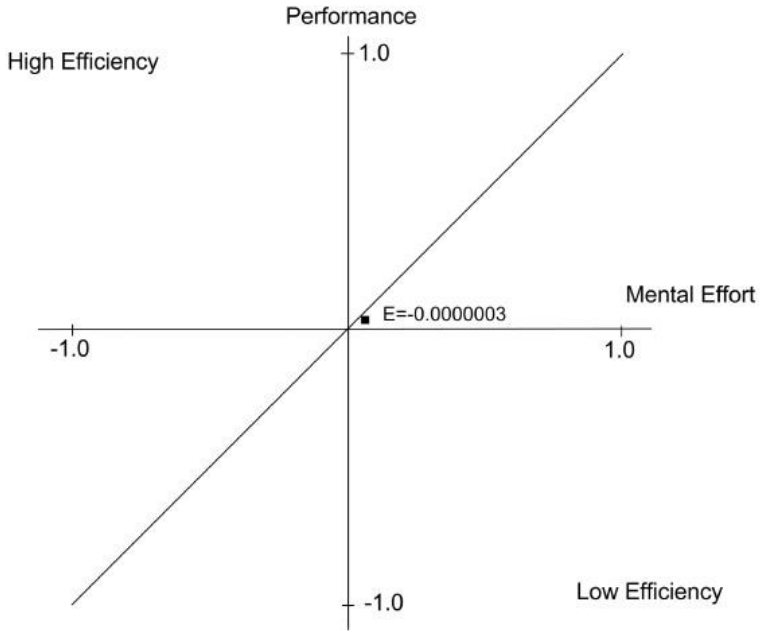


Figure 20.Efficiency metric for Worked Example

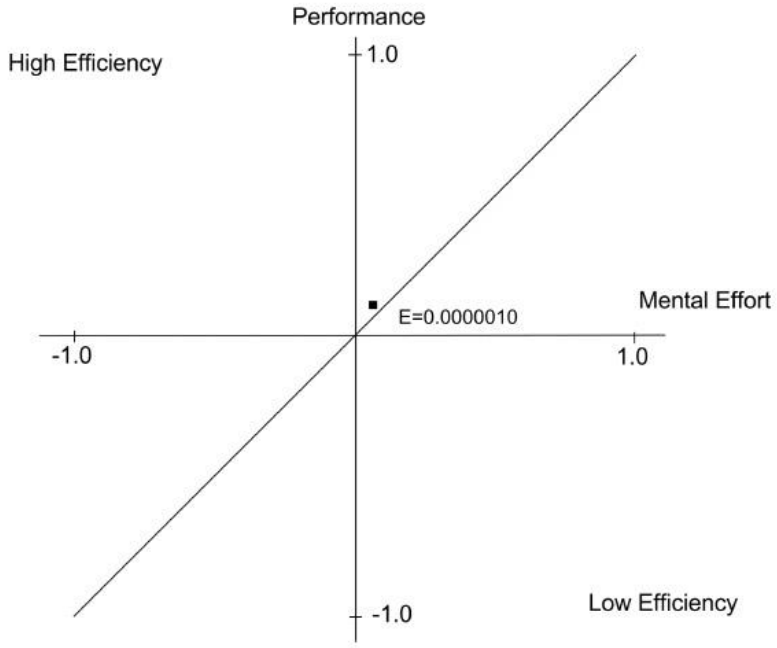


Figure 21. Efficiency metric for Completion Example

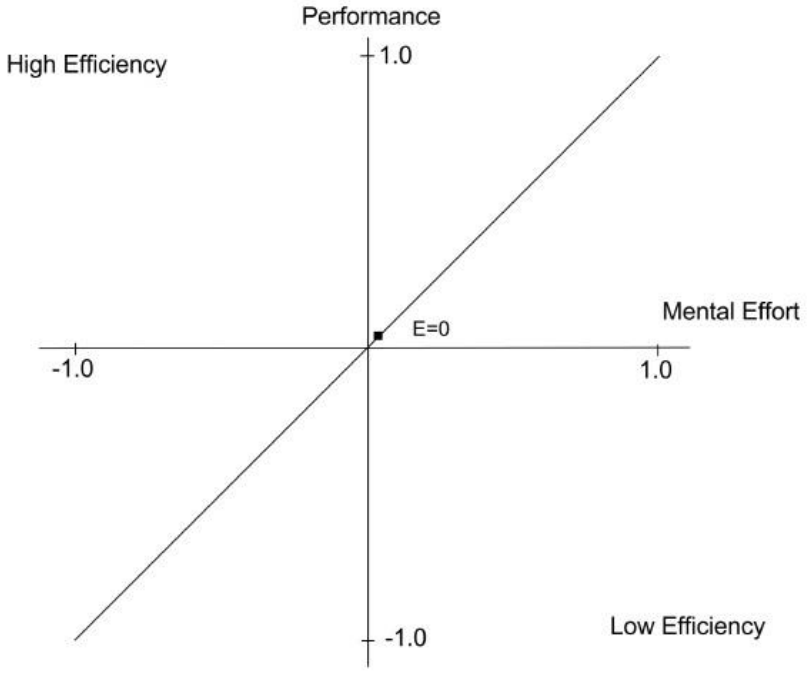


Figure 22. Efficiency metric for Full Practice

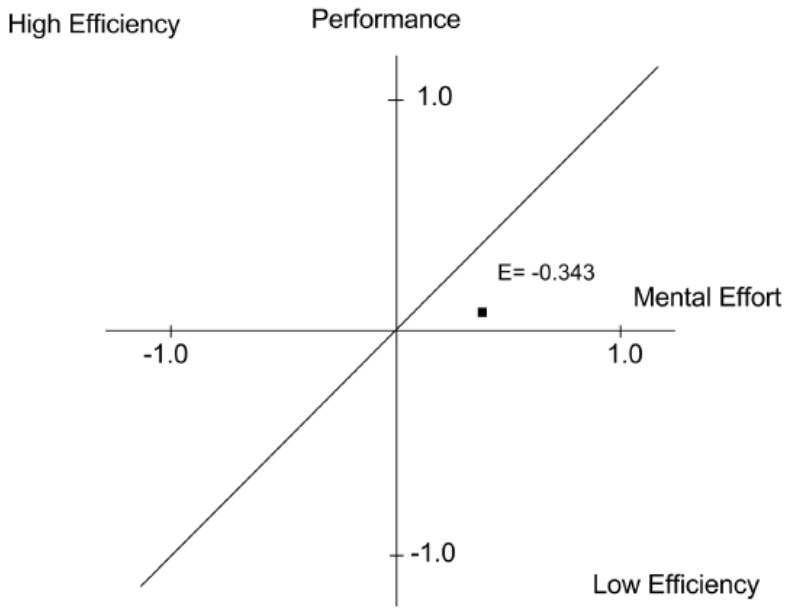


Figure 23. *Efficiency metric for Group-1*

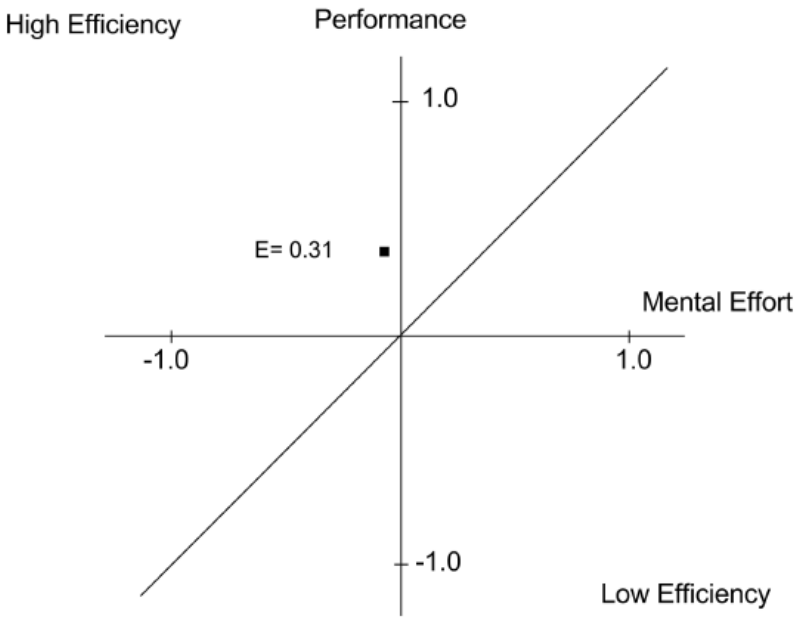


Figure 24. *Efficiency metric for Group-2*

4.5. The Challenges and Facilitators of Working with 3D Game Making Environment

As the majority of our data consists of quantitative parts, we needed to include a qualitative part. We asked participants to join our online questionnaire to find out their opinions about

Starlogo TNG program. It consist of seven questions and totally seven participants voluntarily have joined. Themes and categories are formed in order to present challenges and facilitators of both text-based and block-based programming environment. Overall findings are represented below (Table 16).

Table 16

Themes and Categories

Theme Title		N (7)
1. Previous Programming Experience		
Text-Based	Python	6
	PHP	2
	Visual Basic	3
	HTML	1
	JavaScript	1
	AS3	1
	KIVY	1
Block-Based	Scratch	4
	None	3
2. Programming Language Difficulties/ Challenges		
Text-Based	Low level of English competency	1
	Difficult Structures (syntax, functions, and logic…)	2
	Code Editing	1
	Debugging	3
	Need for extra focus for perfectly running program	2
	Need for high competency for doing what you want	2
	Thinking multiway/dimensional (higher order thinking)	2
	Need for high math skills	1
	Like puzzle, sophisticated	2
Block-Based (Starlogo)	Coding with block is sometimes complex for complex games	1
	Stacking is challenging	1
	Stacking requires complex thinking	1
3. Thoughts about Starlogo TNG		
Visual	Intelligible	5
	Menu design is good	1
	Ease of use	2
	3D environment is not developed well	1
Coding with blocks	Easy	4
	Enjoyable	1
	Need for complex thinking	3
	Due to its resemblance with Scratch, it was easy	1
	Finding commands were hard	5
	Designing complex 3D games were challenging	1

	Flexible	1
	Drag-and-drop is easy	1
	Time consuming	3
4. 3D Designing + Coding		
Pros	Deciding & designing elements	3
	Deciding on game genre	3
	Easy, simple, and understandable	1
	Design, play and edit	2
	Designing more realistic games	1
Cons	Not enough 3D elements	1
	Time consuming	2
	Restricted for educational games	1
	Not flexible for visual changes	1

4.5.1. Previous Programming Experience

Participants were students who had prior knowledge of one or more than one programming languages. Mostly, participants had knowledge about several text-based programming languages. At the same time, 3 participants (out of seven) had experience with Scratch (block-based programming language).

4.5.2. Participants' perceived difficulties of both text-based and block-based programming languages.

Because of its complicated nature, programming language has always been difficult for students to learn and as well as for instructors to teach. Participants' answers of second question reveal some challenges they had during their prior text-based programming courses. Students generally indicated that they have such difficulties whilst creating procedure, debugging (N=3), and code editing (N=1). Moreover, students indicated that one of the challenges of text-based programming is it requires ability to think in multiway (N=2). One student pointed out that "*programming requires (at least) low level of English knowledge*" (N=1) in order to understand the function of each code and how to write them correctly. Moreover, the other challenge in programming is it requires high math skills (N=1). It shows the relation between programming and mathematics. On the other hand, programming has as similar logic as puzzle has (N=2). If you group logical puzzle pieces together, you will have right picture. Similarly, if student has an adequate programming knowledge, it is simple as playing puzzle. However, students generally found easy and funny to program in block-based programming environment (N=4). It demonstrates that presenting text-based code tags in the form of code blocks facilitates the abstract nature of text-based programming language. Nevertheless, students encountered some difficulties with block-based programming environment. Even though, its' simplistic nature, one student indicated that "*stacking the code blocks were challenging*" and also, "*stacking the code blocks requires complex thinking*" (N=3). Therefore, it's difficult for them to create complex code design (N=1). Based on their answer, coding using code blocks is easy when it compared to writing codes.

4.5.3. Students' perception about Starlogo TNG

The Starlogo TNG is software for making 3D games by programming with code blocks. Depending on the participants' answer, it considered into two parts: visual and coding with blocks. Participants revealed their impression whilst working with Starlogo TNG.

When considering the visual part, most students indicated that it is easy (N=4) and enjoyable (N=1) to work with. Starlogo TNG is intelligible and easy to use. Moreover, it has good menu design (N=1). All these statements show that participants' overall perception about visual part of Starlogo TNG was positive. However, the visual of 3D environment was not comprehensible enough (N=1). One participants stated that "*Its' appearance is comprehensible; however, visuals in 3D environment is not developed enough*". The characters or objects are shown with bad quality.

As programming with code blocks using Starlogo TNG, participants revealed their opinion about both positive and negative sides. For most participants (N=5) coding with blocks was easy and enjoyable. One participant stated that "*ease of coding with block is result from its drag-and-drop feature*" (N=1). Other participant stated that "*ease of coding with blocks was because of its similarity with Scratch*" (N=1). On the other hand, it's shown that participants have had some difficulties with coding with blocks. For most participants it was difficult to find the needed code blocks (N=5). It was also difficult for them to design complex 3D games (N=1). One participant pointed out that "*designing a complex game using code blocks is challenging because it requires complex thinking*" (N=1). Moreover, for some other participants, it is easy to code with blocks yet it is time consuming (N=3). One participant pointed out that "*if he had enough experience, he would complete the tasks in less time*".

4.5.4. Pros and cons of designing 3D games using code blocks

It's shown that there are many pros of designing games using Starlogo TNG more than its cons. One of pros that participants pointed out was designing the characters or objects as they wish for their games (N=3). And also, with Starlogo TNG, students can design any kind of games that they want (N=3). Another advantage is that it's easy to fix any errors while designing the games or afterwards (N=2). One participant also came up with the idea that more realistic games can be designed using Starlogo TNG (N=1). However, there are certain cons of designing using Starlogo TNG. Firstly; there aren't enough agents or objects within Breed Editor (N=1). Therefore, game designer need to import the 3D shapes if is not found in Breed Editor. Also it's difficult to make visual changes in 3D environment (N=1). One other con can be boundary of Starlogo TNG program. One participant pointed out that "it is restricted for educational games" (N=1). Moreover, it is considered that working with this program is time consuming with both simple and complex games (N=2).

CHAPTER V

V. DISCUSSION

There are various methods that are used in programming education. For the last few years, using game making method is the one of the most popular method used in programming education. The goal of the present study was to examine whether backwards fading technique facilitates transfer of block-based programming experience into 3D game making in comparison to text-based programming experience and cognitive load perceptions. This study is compared following example formats: worked-example, completion-example, and full-practice. The following sections summarize the hypotheses raised in the introduction, and discuss the theoretical and practical implication of findings of this study.

5.1. The Role of Backwards Fading in the Transfer of Block-Based Programming Experience into 3D-Game Making.

In order to achieve acquisition of a new skill, it is beneficial to use both example study and problem solving with fading procedure. According to Renkl, Atkinson, Maier, & Staley (2002), fading from worked-example to problem solving produced reliable effects on near transfer but not on far transfer items within computer based environment, and it was more beneficial to fade out worked-out solution steps using a backward fading approach (Atkinson, Renkl, & Merrill, 2003). The same result was also found in physic education, which has found that backward-fading procedure fosters near transfer (Renkl, Atkinson, & Maier, 2000). We tested hypothesis of backwards fading in the transfer of block-based programming experience into 3D-game making. We observed the transferring process of programming experience in two different groups of students with different prior knowledge of programming: students who had only text-based programming experience and students who had also block-based programming language experience. Therefore, we expected scores of students having block-based programming experience would be higher than students having text-based programing experience. The result, however, didn't support this hypothesis. Instead, we found that the students having text-based programming experience had higher scores than final scores of others. In addition to the backward fading method effect, Renkl et al. (2002) study showed that backwards fading condition produced more accurate solutions on far-transfer problems. The fact that backward fading methods promote far transfer is consistent with the prior research.

5.2. The Effects of Practice Examples' Formats on 3D-Game Making Performance

It's shown that example-based learning is more effective for novices when compared to problem solving. During this study, students accomplished their activities in three different formats of practice examples: worked-example, completion-example, and full-practice.

Using worked-example in education provides many advantages. Worked-example contains problem itself, solution steps, and the final answer. This procedure demonstrates whole image problem solving to the learners. Therefore, worked example requires low mental effort. According to Sweller & Cooper (1985), using worked-example promote better learning of solution procedure when it compared to the traditional problem solving procedure with no guidance. Prior studies have shown that, worked example is effective when the learner has less prior knowledge (Van Gog, Kester, & Paas, 2011). On the other hand, Nievelstein, van Gog, van Dijck, and Boshuizen (2013) found that worked example were effective for learning than problem solving, both for novice and advanced students on less structured tasks. And they didn't found any evidence for an expertise reversal effect. In other study, Carroll (1994) found worked example in mathematics was helpful for students who had lower math skills. During this study, students who were given worked-example required less acquisition time, needed less direct instruction, and made fewer errors. Alhassan (2017) found positive effect for the use of self-explanation strategy supported with worked-example when learning topic related to the programming. Researcher indicated that this positive effect was obvious through students' ability to write correct programming code and statement, and ability to find errors.

Completion-examples support acquisition of knowledge effectively and more time-efficient (Renkl and Atkinson 2003). Completion-example is the combination of worked-example in instructional setting and demonstrates better learning outcome. Renkl et al. (2002) suggested that worked-examples with backwards fading effect would produce better results than forward fading effect on near transfer items. Gray, St Clair, James, & Mead (2007) suggested that worked-examples with backwards fading effect can be employed in computer science.

Full-practice is the traditional problem solving procedure. Instruction which consists of solving traditional problems forces learners who have weak problem solving strategies. Even though, with such weak strategies, learners may succeed in solving the problem eventually, their learning outcome is little (Van Gog, Kester, & Paas, 2011).

When it comes to acquisition of a skill, it's important to include convenient example-format. It's shown that full-practice requires more effort when compared to the other formats. There are some studies which compares these example formats. Previous studies have shown that worked-example causes higher learning outcomes than full-practice that has no guidance. Worked-example is effective for learner who has less prior knowledge, where completion-example is effective for more advanced students.

Depending on different example formats, the hypothesis was to show the difference between students' final scores of each example format. However, there were two groups of students: (1) who were novice in block-based programming language, (2) who had little knowledge about block-based programming language. We compared students' final scores of each example format; even though it was mixed group. The result supported the hypothesis and it showed that students' final scores of 3D gaming making significantly changed over different example types. The scores of game design in completion-example were significantly higher than those of worked-example.

Effectiveness of example format in educational setting depends upon the learners' level of prior knowledge (Reisslein, Atkinson, Seeling, & Reisslein, 2006). Fact that final scores in completion-example are higher than scores in worked example is showing that there was reversal of the worked example effect. For novice learners, studying with worked-examples helps them to experience or form relevant problem solving schemas, which provide them to gain expertise as a result. However, for learners who already have some schema in their long-term memory, studying with worked examples will be gaining no benefit from it (Kalyuga, Ayres, Chandler, & Sweller, 2003). Moreover, according to the Renkl and Atkinson (2003), low prior knowledge learners benefit more from studying worked-examples. However, as knowledge increases, completion-example solving becomes the more effective learning activity. For more knowledgeable learners, worked example becomes redundant, and also becomes the source of extraneous cognitive load, but not germane cognitive load. This phenomenon is called *expertise reversal effect*. However full-practice scores of game design were significantly lower than either worked-example or completion-example. It's clear that full-practice format requires more effort to make.

5.3. The Effects of Practice Examples' Formats on 3D-Game Perceived Difficulty.

Mostly mental load is measured by learner ratings of lesson difficulty on a 1 to 7 or 1 to 9 scale. In our study, each week students rated difficulty from 1 to 9 scale during all formats of examples. The rating scale technique can be regarded as a valuable research tool for estimating cognitive burden in instructional research (Paas, 1992). The hypothesis was to investigate whether students' perceived difficulties affected by example format. Depending on the difficulty of example formats, we expected full-practice format would be the most difficult. The results regarding the measures of cognitive load that are based on mental effort revealed that significantly different value was found on students' perceived difficulty of full-practice. It was found that full-practice was more difficult than either worked-example or completion-example. Students reported that very high levels of cognitive load required during full-practice and got low scores on their full-practice activity. Fact that full-practice assumed as the most difficult format in this study is that, it requires different levels of mental activity such as computational thinking, problem solving, and design thinking simultaneously. However, there was no significant difference on students' perceived difficulties between worked-example and completion-example. Worked-example provides learners with full guidance that are needed for problem solving. Completion-example also provides learner with solution steps yet one single solution step is omitted. Therefore, it requires no/very little mental effort (Skudder & Luxton-Reilly, 2014). As students gained the acquisition of a new skill with the help of worked-example, solving problem with completion-example format would be easier than solving the whole problem with no guidance. Full-practice requires learners to complete full solution on their own. One of the reason that students' perceived difficulties between worked-example and completion-example may result from their prior knowledge about block or text-based programming language. Moreover, whether students are novice or not, providing them with full-practice format diminished their motivation.

5.4. The Effects of Practice Examples' Formats on Cognitive Load

In order to understand how students' cognitive loads affected by example formats, we measured the efficiency of each example formats by performance scores (the average scores for each example type) and difficulty ratings (the average ratings for each example type). Efficiency metric was used in order to calculate cognitive loads. In cognitive load theory, efficiency is defined by learner performance and learner mental effort. According to the Clark et al. (2006, p. 19) "Instructional environments that result in higher learning outcomes with less mental effort are more efficient than environment that lead to lower outcomes with greater mental effort". Efficiency metric is used to quantify the efficiency of instructional product. According to Paas and Van Merrienboer (1993), in order to calculate efficiency value, all values are transformed into z-scores. Performance z scores and difficulty rating z scores are represented in a cross of axes. In figure 23 or 24, the line labelled "E=0" indicates that efficiency is zero. All points on this line are parallel to E = 0 represents the same mental efficiency. Upper-left part of the coordinate system shows an increase in efficiency which means higher performance in relation to less mental effort. Lower-right part of the coordinate system shows a decrease in efficiency, which means lower performance in relation to more mental effort.

First, cognitive loads of practice problems were measured in order to see impact of examples' format. The results regarding the measures of cognitive loads that are based on efficiency metric revealed that efficiency value of completion-example was the highest among all practice example formats. However, efficiency value of worked-example was resulted as low and efficiency value of full-practice was resulted as neutral. Cognitive loads of all practice problems were measured depending on two different groups of students: (group-1) students who had experience with block-based programming language (Scratch), (group-2) students who only had experience with text-based programming language (Python). The averages of all practice problems of two groups were taken and are compared. The results regarding the measures of cognitive loads that are based on efficiency metric revealed that efficiency value of all practice problems of Group-2 was higher that the efficiency value of other group (see Figures 23, 24).

5.5. The Challenges and Facilitators of Working with 3D Game Making Environment

With the qualitative data, a group of important results were achieved. Overall examination of qualitative data showed that participants' perception about 3D game making environment can be gathered under four sections including students previous programming experience, challenges/difficulties of programming languages (text-based and block-based), perception on StarLogo TNG, students perception on designing 3D games using code blocks with StarLogo TNG.

Students' overall perception showed that text-based programming is more challenging than block-based programming language. It is well drawn that text-based programming language skill has always been difficult to gain. However, block-based programing environment was easy for students because of its tangible nature. Mostly, block-based programming

environment is effective for novice in programming language (Denner, Werner, & Ortiz 2012).

For most students, Starlogo TNG is an easy and enjoyable to work with. Considering the interface, StarLogo TNG was easy to deal with. However, programming with code blocks was resulted both positively and negatively. Positive part was that using code blocks was easy but when it comes to design a complex game, it is time consuming. Because of its drag-and-drop feature, it was easy for learners. Smith & Duncan (2011) indicated that drag-and-drop feature provides additional feedback with an auditory "click" when code blocks are placed together. Therefore, students don't have to worry about syntax error.

Combination of text-based programming language and designing 3D games affected learners mostly in a good way. Students had the whole authority to choose genre, characters, and also design elements of their games. In this way, students were the designer of their own games. By becoming designers of their own game, students learn not only programming but also they gain critical and mathematical thinking skill, complex problem solving skill and deep understanding (Akcaoglu & Kale, 2016). More importantly, designing games provides students to learn by doing. It was also resulted that one of the advantage of StarLogo TNG was to be able to design, play, and edit simultaneously. Klopfer et al. (2009) indicated that having 3D visualisation of simulation output provides immediate feedback on the progress of the simulation as it being built. With this immediate feedback, students can notice their mistakes easily and fix (Begel & Klopfer, 2004). On the other hand, many disadvantages of designing 3D games in block-based environment have arisen. These were inadequacy of StarLogo TNG software: lack of character (Starlogo Breed) and no intervention on 3D visualisation of simulation output. Even though, StarLogo TNG was made an ideal software for educational purpose (Smith & Duncan, 2011), one student found it restricted for educational games.

5.6. Limitations

It's important to indicate that the findings of this study are limited to the participants of this case. They are the senior university students of Computer Education and Instructional Technologies who enrolled in an E-Game Based Learning course. Students had different level of prior programming knowledge. Another important limitation of this study is time. The fact that students' full-practice scores were low might be resulted from insufficient amount of time spent during worked-example and completion-example sessions. It suggests that in order to get higher scores in full-practice, duration of worked-examples and completion-examples can be increased.

5.6.1. Delimitations

This case study explored how difficult for students to transfer their previous programming knowledge into block-based programming while designing 3D game using Starlogo TNG program. Participants of this case were senior university students of Computer Education and Instructional Technologies (CEIT) who enrolled in an E-Game Based Learning course. This

study is not designed for basic programming skills, yet to observe the ability to transfer one's prior knowledge into new situation. CEIT students who had prior programming knowledge/skill were appropriate for this study. There is not much delimitation in this study. However, variety of examples provided during this study delimited. In this study, provided completion-examples are delimited. Fact that full-practice format was assumed as the most difficult format in this study is because it requires high/different level of mental load simultaneously. In order to decrease students' perceived mental effort level, variety of worked and completion example can be enriched.

5.7. Implications

5.7.1. Theoretical Implications

The use of examples is also very common in introductory programming courses. Our findings are on the efficiency of a learning environment that combines three different example formats with backwards-fading approach. During gaining of a new skill, it is beneficial to provide learners with worked-examples in order to form relevant schema in long term memory. Worked-examples are beneficial for novice learners with less/no prior knowledge. Once schema is formed, learner can solve similar problem in the way of near or far transfer. Educational transfer of learning can enable students to apply their knowledge and skills anytime and anywhere they need. In order to observe smooth transition from worked-example to full-practice problem, fading effect is used. Backwards fading effect provides learners with worked-example with full guidance and then followed by another worked example with one last step missing. It is beneficial to fade out worked-out solution steps using a backward fading approach (Atkinson, Renkl, & Merrill, 2003). Completion-example is beneficial when the learner has prior knowledge or non-novice learner. Full-practice is only effective when learner is expert. Providing learners with boundless practice of the performance may cause better transfer (Butler, 2010).

5.7.2. Practical Implications

The practical implication of our study is quite clear. Example formats are used a lot in Computer Science and similar fields. And in most courses use the instructional design of example-problem blocks. It's possible that some difficulties may occur during courses especially for the novice learners. It is may be due to intrinsic cognitive load imposed by sophisticated nature of programming tasks. So far, there is strong empirical evidence for the superiority of completion-example over worked-example or full-practice when designing 3D games using block-based programming environment. In order to make full-practice problems easy for learner, extraneous load must be minimized. This study suggests that future study can organize worked & completion examples in more effective way in terms of learning time.

5.8. Recommendations for Further Studies

Participants of this study practiced different types of example formats throughout the study. These example formats were *worked-example*, *completion-example*, and *full-practice*.

Students' scores of completion-example were resulted higher than their scores of worked-example. However, their scores of full-practice was resulted significantly lower than either worked-example or completion example. These findings suggest that, for students having prior (block-based or text-based) programming skill, worked example became redundant. However, to achieve sufficient knowledge transfer in order to accomplish full-practice example, duration of completion-example sessions can be increased. Duration can be beneficial for achieving sufficient schemas in order to be able to accomplish full-practice. In finding which arises from students' perceived difficulty, full-practice was also more difficult than either worked-example or completion example. Full-practice format was assumed as the most difficult format in this study because it requires high/different level of mental effort simultaneously. However, there was no significant difference on students' perceived difficulties between worked-example and completion-example. These findings suggest that in order to decrease students' perceived mental effort level, variety of worked and completion example can be enriched. Moreover, duration of completion-example sessions can be increased in order to make sure of whether students have sufficient skill of solving problems with no guidance. The results regarding the measures of cognitive loads that are based on efficiency metric revealed that efficiency value of all practice problems of Group-2 was higher that the efficiency value of other group (see Figures 23, 24). Findings suggested that future study can be conducted without worked-example format with non-novice learners. Overall suggestion for further studies, providing students with sufficient completion-examples who have prior knowledge and skill in order to make sure whether students achieved enough knowledge skill and whether students are able to complete the full-practice without any guidance.

5.9. Conclusion

It's assumed that using games have many benefits in education. It is not only to teach knowledge but also help them to learn to develop competences that will be useful to them in their future career endeavours. There are many educational games that are properly designed and developed in order to be fully exploited. However, in this study students became designers and developers of their own educational games instead of being consumers of educational games already developed. Since the participants had prior programming knowledge, students' ability to transfer their prior knowledge was observed. During game making process, different example formats were used: worked-example, completion-example, and full-practice. The evidence suggests certain worked example techniques with backwards fading effect are an improvement over conventional problem solving techniques, in terms of learning time and performance on far transfer for novices in block-based programming environment. In situations where the student is not novice, worked-example appears more to be redundant. However, providing them with completion-example appears more effective example format. For full-practice problems, it's assumed that providing students with sufficient time and variety of example can lead better learning outcome.

REFERENCE

Akcaoglu, M., & Kale, U. (2016). Teaching to teach (with) game design: Game design and learning workshops for preservice teachers. *Education*, *16*(1), 60-81.

Alhassan, R. (2017). The Effect of Employing Self-Explanation Strategy with Worked Examples on Acquiring Computer Programming Skills. *Journal of Education and Practice, Vol.8, No. 6*, 186-196.

Ali, A., & Smith, D. (2014). Teaching an Introductory Programming Language in a General Education Course. *Journal of Information Technology Education: Innovations in Practice, 13*, 57-67. Retrieved from http://www.jite.org/documents/Vol13/JITEv13IIPp057-067Ali0496.pdf

An, Y. J. (2016). A case study of educational computer game design by middle school students. *Educational Technology Research and Development*, *64*(4), 555-571.

An, Y. J., & Cao, L. (2017). The Effects of Game Design Experience on Teachers' Attitudes and Perceptions regarding the Use of Digital Games in the Classroom. *TechTrends*, *61*(2), 162-170.

Atkinson, R. K., Renkl , A., & Merrill, M. M. (2003). Transitioning From Studying Examples to Solving Problems: Effects of Self-Explanation Prompts and Fading Worked-Outs Steps. *Juornal of Educational Psychology, Vol. 95, No. 4*, 774-783.

Armoni, M., Meerbaum-Salant, O., & Ben-Ari, M. (2015). From Scratch to ""Real" Programming. *ACM Transitions on Compting Education, Vol. 14, No. 4*, Article 25

Backlund, P., & Hendrix, M. (2013, September). Educational games-are they worth the effort? A literature survey of the effectiveness of serious games. Games and virtual worlds for serious applications (VS-GAMES), Paper presented at the 2013 5th international conference on IEEE, (pp. 1-8).

Battistella, P., & Wangenheim, C. V. (2016). Games for teaching computing in higher education–a systematic review. *IEEE Technology and Engineering Education*, *9*(1), 8-30.

Begel , A., & Klopfer, E. (2004). Starlogo TNG: An Introduction to Game Development. 1-15.

Carroll, W., M. (1994). Using WorkedExample as as Instructional Support in the Algebra Classroom. *Journal of Educational Psychology, Vol. 86, No. 3*, 360-367.

Chang, C.-K. (2014). Effects of Using Alice and Scratch in an Introductory Programming Course For Corrective Instruction . *J. Educational Computer Research, Vol. 51 (2)*, 185-204.

Chapman, B. E., & Irwin, J. (2015). Python as a first programming language for biomedical scientists. In *Proceedings of the 14th Python in Science Conference (SCIPY 2015): Published online at http://conference. scipy. org/proceedings/scipy2015/(last accessed 23-09-2015)*.

Chen, O., Kalyuga, S., & Sweller, J. (2015). The Worked Example Effect, the Generation Effect, and Element Interactivity. *Journal of Education Psychology, Vol. 107, No. 3*, 689-704.

Choi, H.-H., Van Merrienboer, J., & Paas, F. (2014). Effects of the Physical Environment on Cognitive Load and Learning: Towards a New Model of Cognitive Load . *Educational Psychology eview*, 225-244.

Chu, H.-C. (2013). Potential Negative Effects of Mobile Learning on Students' Learning Achievement and Cognitive Load- A Format Assessment Perspective. *Educational Technology & Society*, 332-344.

Clark, R. C., Nguyen , F., & Sweller, J. (2006). *Efficiency in Learning: Evidence-Based Guidelines to Manage Cognitive Load.* San Francisco: Pfeiffer.

Cooke, N. L., Guzaukas, R., Pressley, J. S., & Kerr, K. (1993). Effects of Using a Ratio of New Items to Review Items During Drill and Practice: Three Experiments. *EDUCATION AND TREATMENT OF CHILDREN Vol. 16, No. 3*, 213-234.

Denner, J., Werner, L., & Ortiz, E. (2012). Computer games created by middle school girls: Can they be used to measure understanding of computer science concepts? *Computers & Education, 58*(1), 240–249.

Ebner, M., & Holzinger, A. (2007). Successful implementation of user-centered game based learning in higher education: An example from civil engineering. *Computers & education*, *49*(3), 873-890.

Ekeyi, D. N. (2013). Effect of demonstration method of teaching on students' achievement in agricultural science. *World Journal of Education*, *3*(6), 1.

Fernandez Leiva, A. J., & Civila Salas, A. C. (2010). Practices of Advanced Programming: Tradition Versus Innovation. *Computer Applications in Engineering Education*, 237-244.

França Batista, A., Connolly, T., & Peres Angotti, J. (2016). A Framework for Games-Based Construction Learning: A Text-Based Programming Languages Approach. *Academic Conferences & Publishing International Ltd.*, 815-823.

Fromme, J. (2003, 5). *Computer Games as a Part of Children's Culture*. Retrieved from Game Studies: http://www.gamestudies.org

Garris, R., Ahlers, R., & Driskell, J. E. (2002). Games, motivation, and learning : A research and practice model. *Simulation & Gaming*, 441-467.

Garvey, G. P. (2015). Fostering 21st Century Skills through Game Design and Development. *International Association for Development of the Information Society*.

Gray, S., St Clair, C., James, R., & Mead, J. (2007). Suggestion for Graduated Exposure to Programminh Concepts Using Faded Worked Examples. In *Proceedings of the third international workshop on Computing education research* (pp. 99-110). ACM.

Guibert, N., Guittet, L. T., Girard, P. (2005). A study of an efficiency of an alternate programming paradigm to teach the basics of programming. *Recuperado de http://www.lisi.ensma.fr/fr/equipes/idd/publications.html.*

Hayes, E. R., & Games, I. A. (2008). Making Computer Games and Design Thinking. *Games and Culture*, 1-24.

Kahn, K. (1996). ToonTalk: An Animated Programming Environment for Children . *Journal of Visual Languages and Computing* , 197-217.

Kalelioğlu, F. (2015). A new way of teaching programming skills to K-12 students: Code. org. *Computers in Human Behavior*, *52*, 200-210.

Kalyuga, S., Ayres, P., Chandler, P., & Sweller, J. (2003). The Expertise Reversal Effect. *Educational Psychologist, Vol. 38(1)*, 23-31.

Kelleher, C., & Pausch, R. (2006). Lessons Learned from Designing a Programming System to Support Middle School Girls Creating Animated Stories. *IEEE Symposium on Visual Languages and Human-Centric Computing*.

Kirriemuir, J., & Mcfarlane, A. (2004). *Literature Review in Games and Learning.A NESTA Futurelab Research report - report 8.*

Klopfer, E., Scheintaub, H., Huang, W., Wendel , D., & Roque, R. (2009). The Simulation Cycle: combining games, simulations, engineering and science using StarlogoTNG. *E-Learning, Vol. 6 No:1*, 71-96.

Lee, Y. J. (2011). Empowering teachers to create educational software: A constructivist approach utilizing Etoys, pair programming and cognitive apprenticeship. *Computers & Education*, 527-538.

Leutenegger, S. T., & Edgington, J. (2007). A Games First Approach to Teaching Introductory Programming. *ACM SIGCSEBulletin, 39(1)*, 115-118.

Lin, J. M. C., & Yang, M. C. (2009). Analyzing student performance and attitudes toward textual versus iconic programming languages. *Journal of Computers in Mathematics and Science Teaching, Vol.28, No:1*, 71-89.

Lui, T. C., Lin, Y. C., Tsai, M. J., & Paas, F. (2011). Split-attention and redundancy effects on moble learning in physical environments. *Computers and education, Vol.56, No:2*, 172-181.

Lye, S. Y., & Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12?. *Computers in Human Behavior, Vol. 41*, 51-61.

Maasoumi, M., Khazaei, B., & Shafiei, M. (2014). Programming The Way to Enrich Educational Foundations of E-Learning . *E-Learning and Software for Education*, 306-313.

Malan, D. J., & Leitner, H. H. (2007). Scratch for Budding Computer Scientists. *In Proceedings of the 38th SIGCSE technical symposium on Computer science education. ACM*, 223-227.

Malliarakis, C., Satratzemi, M., & Xinogalos, S. (2013). A Holistic Framework for the Development of an Educational Game Aiming to Teach Computer Programming. *Academic Conferences International Limited*, 359-368.

Malliarakis, C., Satratzemi, M., & Xinogalos, S. (2014). Designing Educational Games for Computer Programming: A Holistic Framework. *Electronic Journal of e-Learning*, Vol. *12*(3), 281-298.

Maloney, J., Peppler, K., Kafai, Y. B., Resnick, M., & Rusk, N. (2008). Programming by Choise: Urban Youth Learning Programming with Scratch. *In Proceedings of the 39th SIGCSE technical symposium on Computer science education. ACM*, 367-371.

Mann, B. D., Eidelson, B. M., Fukuchi, S. G., Nissman, S. A., Robertson, S., & Jardines, L. (2002). The development of an interactive game-based tool for learning surgical management algorithms via computer. *The American Journal of Surgery, Vol. 183 (3)*, 305–308.

Miller, G. A. (1956). The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information. *The Psychological Review, Vol. 63*, 81-97.

Miller, W. L., & Crabtree, B. F. (1999). The dance of interpretation. *Doing qualitative research*, *2*, 127-143.

Miller, K., Lasry, N., Chu, K., & Mazur, E. (2013). Role of physics lecture demonstrations in conceptual learning. *Physical review special topics-physics education research*, *Vol. 9*(2), 020113.

Moreno , R., Reisslein, M., & Ozogul, G. (2009). Optimizing Worked-Example Instuction in Electrical Engineering: The Role of Fading and Feedback during Problem-Solving Practice. *Journal of Engineering Education*, 83-92.

Najar, A.S., Mitrovic, A. & McLaren, B.M. (2016). Learning with intelligent tutors and worked examples: selecting learning activities adaptively leads to better learning outcomes than a fixed curriculum. *User Modeling & User-Adapted Interaction, Vol. 26*(5), 459-491 doi:10.1007/s11257-016-9181-y

Nievelstein, F., van Gog, T., van Dijck, G., & Boshuizen, H. P. A. (2013). The worked example and expertise reversal effect in less structured tasks: Learning to reason about legal cases. *Contemporary Educational Psychology*, 38, 118 – 125. http://dx.doi.org/10.1016/j.cedpsych.2012.12.004

Omwirhiren, E. M., & Ibrahim, K. U. (2016). The Effects of Two Teachers' Instructional Methods on Students' Learning Outcomes in Chemistry in Selected Senior Secondary School in Kaduna Metropolis, Nigeria. *Journal of Education and Practice, Vol. 7(15)*, 1-9

Paas, F. G. (1992). Trainig strategies for attaining transfer of problem-solving skill in statistics: A cognitive-load approach. *Journal of Educational Psychology, Vol. 84(4)*, 429-434.

Paas , F., & Ayres , P. (2014). Cognitive Load Theory: A Broader View on the Role of Memory in Learning and Education. *Educational Psychology Review, Vol. 26*, 191-195.

Paas, F. G., & Van Merriënboer, J. J. (1993). The efficiency of instructional conditions: An approach to combine mental effort and performance measures. *Human factors, Vol. 35(4),* 737-743.

Papert, S. (1980). *Mindstorm: Children, Computers, and poweful ideas.* New York: Basic Books.

Papastergiou, M. (2009). Digital game-based learning in high school computer science education: Impact on educational effectiveness and student motivation. *Computers & Education*, *52*(1), 1-12.

Pasztor, A., Lakatos Török, & Lakatos Török, E. (2010). Effects of Using Model Robots in the Education of Programming. *Informatics in Education, Vol.9, No. 1*, 133-140.

Peppler, K. A., & Kafai, Y. B. (2007). What Videogame Making Can Teach Us about Literacy and Learning: Alternative Pathways into Participatory Culture. *Online Submission*.

Perkins, D. N., & Salomon, G. (1992). Transfer of Learning. *International Encyclopedia of Education, Second Edition*, 1-13.

Petri, G., & Gresse von Wangenheim, C. (2017). How games for computing education are evaluated? A systematic literature review. *Computers & Education Vol. 107*, 68-90.

Reisslein, J., Atkinson, R.K., Seeling, P., & Reisslein, M. (2006). Encountering the expertise reversal effect with a computer-based environment on electrical circuit analysis. *Learning & Instruction, 16*(2), 92-103.

Renkl, A., & Atkinson, R. K. (2003). Structuring the Transition From Example Study to Problem Solving in Cognitive Skill Acquisition: A Cognitive Load Perspective. *Educational Psychologist, Vol. 38(1)*, 15-22.

Renkl, A., Atkinson, R. K., Maier, U. H., & Staley, R. (2002). From Example to problem solving: Smooth Transitions helps learning. *Journal of Experimental Education, 70(4)*, 293-315.

Skudder, B., & Luxton-Reilly, A. (2014). Worked examples in computer science. *ACE '14 Proceedings of the Sixteenth Ausralasian Computing Education Conference, Vol. 148*, 59-64.

Smith, V. A., & Duncan, I. (2011). Biology Students building Computer Simulations Using StarLogo TNG. *Bioscience Education, Special Edition, Vol. 18(1)*, 1-9.

Sweller, J., & Cooper, G. A. (1985). The Use of Worked Examples as a Substitute for Problem Solving in Learning Algebra. *Cognition and Instruction, Vol.2 (1)*, 59-89.

Sweller, J., van Merrienboer, J. J., & Paas, F. G. (1998). Cognitive Architecture and Instructional Design. *Educational Psychology Review, Vol. 10, No. 3*, 251-296.

Tabet, N., Gedawy, H., Alshikhabobakr, H., & Razak, S. (2016, July). From Alice to Python. Introducing Text-based Programming in Middle Schools. Paper presented at the Proceeding 21st Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE), Arequipa, Peru, (pp. 124-129).
doi: 10.1145/2899415.2899462

Ültanir, E. (2012). An Epistemologic Glance at the Constructivist Approach: Constructivist Learning in Dewey, Piaget, and Montessori.

Van Gog, T., Kester, L., & Paas, F. (2011). Effects of worked examples, example-problem, and problem-example pairs on novices' learning. *Contemporary Educational Psychology, 36*(3), 212-218.

Vasilateanu, A., Wyrazic, S., & Pavaloiu, B. (2016). A Science Fiction Serious Game For Learning Programming Languages. *eLearning and Software for Education*, 561-564.

Weidman, J., & Baker, K. (2015). The Cognitive Science of Learning: Concepts and Strategies for the Educator and Learner. *Neuroscience In Anesthesiology and Perioperative Medicine, Vol. 121, No.6*, 1586-1599.

Weintrop, D., & Holbert, N. (2017). From Blocks to Text and Back: Programming Patterns in a Dual-modality Environment. *SIGCSE.*, 633-638.

Wilson, A., Connolly, T., Hainey, T., & Moffat, D. C. (2011, October). Evaluation of Introducing Programming to younger School Children Using a Game Making Tool. Paper presented at the Proceedings of the 5th European Conference on Games-based Learning (ECGBL), University of Athens, Greece, 20-21 October (pp. 639-694). Academic Conferences International Limited.

Wilson, A., Hainey , T., & Connolly, T. M. (2013). Using Scratch with Primary School Children: An Evaluation of Games Constructed to Gauge Understanding of Pragramming Concepts. *International Journal of Game-Based Learning, 3(1),* 93-109.

APPENDIX

APPENDIX A-1: Full-Practice game hand-out of Red Riding Hood: Collecting Carrots (Turkish)

E-Oyun Tabanlı Öğrenme Dersi

LAB Uygulama: Oyun – 1

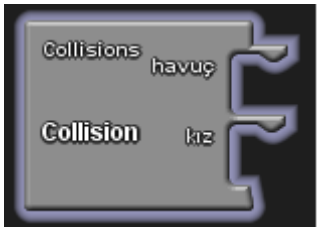
Senaryo: Kırmızı başlıklı kızın ananesine akşam yemeği için havuç toplayan oyun hazırlayınız. Sahne kırmızı başlıklı kızın ananesi olan ev, havuçlar ve ısırgan otlarından oluşacaktır. Kırmızı başlıklı kız havuca çarptığında hem kız büyüyecek hem de havuç sahneden kaybolacak, ve ısırgan otuna çarptığında ise küçülecektir. Sahnedeki bütün havuçları topladıktan sonra eve gidecek ve oyun bitecektir.

> **İpucu**: Başrolü büyüdüğünde boyutu evin büyüklüğünden büyük olmaması için büyüklüğü arttıran değere 1 yerine 0.5 veriniz.

Nesneler: Ev, Kırmızı başlıklı kız (başrol), havuç ve ısırgan otu(küçük yeşil ot)

Tasarlama aşaması:

- İçerikteki tüm objeleri sahnede rastgele yerleştiriniz. Ek olarak ağaç yerleştirebilirsiniz.
- Başrolünün nesnelere çarpma eylemini Collision (Çarpışma) kod bloğu ile oluşturabilirsiniz.



- Başrolün havuca çarptığında ise havucun sahneden kaybolmasını sağlayınız. Bunun için de Die kod bloğunu kullanınız.

- Başrolün büyüyüp ya da küçülmesi için Set Size(Boyut Ayarlama) kod bloğunu kullanınız.

APPENDIX A-2: Full-Practice game hand-out of Red Riding Hood: Collecting Carrots (English)

E-Game Based Learning Course

LAB Practice – Game 1

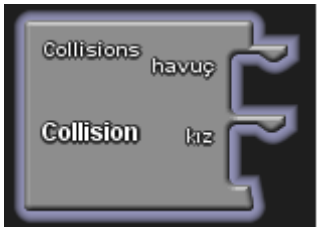
Scenario: Game consists of Red Riding Hood as a player, house of her grandmother, carrots, and nettles. Red Riding Hoods has to collect all carrots to prepare dinner for her grandmother. When the main role collides with the carrot, she gets bigger and carrot disappears. When she collides the nettle, she gets smaller. The game ends when she gets home after all the carrots are collected,

> **Hint**: Each time when the main role collides the carrot, set the enlarging value as 0.5 instead of 1 to avoid to become bigger the size of the house

Objects: Red Riding Hood (main role), house, carrot, and nettle

Designing:

- Locate all the objects randomly in the terrain. You can also locate trees additionally.
- Use "Collision" code block for action of the main role colliding with other objects.



- Use "Die" code block for disappearance of carrot when it is collided.

- Use "Set Size" code block to change the size of the main role.

APPENDIX B-1: Full-Practice hand-out of Crossing Bridge (Collecting coins) (Turkish)

E-Oyun Tabanlı Öğrenme Dersi

LAB Uygulama: Oyun – 2

Senaryo: Oyunun başrolü olan arabanın köprü yolunda karşıdan karşıya geçmesini sağlayan oyun hazırlayınız. Köprüde başroldeki arabanın dışında hareket halinde olan arabalar ve demir paralar olacak. Başroldeki araba köprüde ilerlerken diğer arabaları çarparsa puan azalacak ve demir paralara çarparsa artı puan kazanacaktır. Sonunda başroldeki araba köprüyü geçtiğinde ise ekranda 'Tebrikler, Köprüyü geçtiniz!' yazısını yazınız.

Nesneler: başrol arabası, hareket halinde olan arabalar, köprü ve demir para

Tasarlama aşamaları:

- Köprüyü ve demir paraları SketchUp programında tasarlayınız. Köprü çift şeritli olacaktır.
- Sahnede karşı karşıya olan iki dağı bağlayan bir köprü ve dağların arasında deniz olacak şekilde düzenleyiniz.
- Hareket halinde olan arabaları eşit sayıda köprünün iki tarafında olacak şekilde ve her biri için (başrol arabası dâhil olmak üzere) ayrı ayrı konum belirleyiniz.

  İpucu: Arabaların sürekli hareket halinde olacak şekilde ayarlanması için Run(Çalıştır) bloğunda hareket vermek istediğimiz karakterin(kendi oyunlarınıza göre) karşısında Forward(ileri gitme komut) bloğunu yerleştirebilirsiniz

- Demir Paraları köprünün üzerinde yerleştirilirken sürekli dönecek şekilde ayarlayınız ve her biri için de ayrı ayrı konum belirleyiniz.

  İpucu: Demir paraların sürekli dönme hareketini vermesi için Run(Çalıştır) bloğunda dönme hareket vermek istediğimiz karakterin(kendi oyunlarınıza göre) karşısında Left degs(sola dönme komut) ya da Right degs(sağa dönme komut) bloğunu yerleştirebilirsiniz

- Hareket halinde olan araba ve demir para sayısı: isteğe bağlı

APPENDIX B-2: Full-Practice hand-out of Crossing Bridge (Collecting coins) (English)

E-Game Based Learning Course

LAB Practice – Game 2

Scenario: Game consists of cars one of them as a main role, coins, bridge between two mountains, and a river. The bridge has two-lane road. On the bridge, there will be cars in motion and turning coins. Main role (car) has to cross the bridge. If it collides with other cars, score will decrease. If it collides with coins, score will increase. Game ends when the main role reaches to the end of the bridge Finally, sign as "Congratulations You've crossed the Bridge" pops up on the screen.

Objects: Main-role car, other cars, coins, and bridge.

Designing:

- Design coins and bridge using SketchUp. Be sure to create bridge with two-lane road.
- Locate the bridge between two mountains and draw river under the bridge between the mountains.
- Place every cars at different locations on the bridge.

  Hint: In order to make cars to be in motion, use Forward code block for each car in Run block

- Place the coins continuously turning way in different location.

  Hint: In order to make coins to be continuously turning, use Left degs or Right degs code block for coin in Run block

- It is optional to set the number of cars and coins.**APPENDIX C-1: Rubric (Sample)**

APPENDIX C1: Rubric Sample

Student Name:

Grade:

Game:

Code Blocks	Not tried	Incomplete	Completed
1. Edit Breeds			
2. Creating Procedure Block			
3. Using Logic-If Block (movement)			
4. Creating Setup Block (Creating characters and other objects)			
4.1. Set xy			
4.2. Set size			
5. Creating Run Block			
6. Terrain Screen			

APPENDIX C-2: Rubric for Worked Example

Session-1: Introduction to Starlogo TNG
Using IF Block

Student Name:

Grade:

Game 1: Mario

Code Blocks	Not tried	Incomplete	Completed
1. Edit Breeds			
2. Creating Procedure Block			
3. Using Logic-If Block (movement)			
4. Math Block			
5. Creating Setup Block 6. (Creating characters and other objects)			
7. Set xy & size			
8. Creating Run Block			
9. Terrain Screen			

APPENDIX C-3: Rubric for Worked Example

Session-2: Introduction to Collision Block

Student Name:

Grade:

Game 2: Treasure

Code Blocks	Not tried	Incomplete	Completed
1. Edit Breeds			
2. Creating Procedure Block			
3. Using Logic-If Block (movement)			
4. Creating Setup Block (Creating characters and other objects)			
4.1. Set xy & size			
4.2. Setting Camera			
4.3. Using Scatter Block			
4.4. Set Color Block			
4.5. Set & Show Score Blocks			
5. Collision Block			
5.1. Using If & Math Blocks			
5.2. Using Stamp Block			
5.3. Using Die Block			
6. Creating Run Block			
7. Terrain Screen			

APPENDIX C-4: Rubric for Completion Example

Session-3: Collision Block - Completion Example 1

Student Name:

Grade:

Game 3: Running Away From the Bears

Code Blocks	Not tried	Incomplete	Completed
1. Collision Block			
1.1.Using Say Block			
1.2. Creating Camera			
1.3.Using Die Block			
2. Creating Run Forever Block			

APPENDIX C-5: Rubric for Completion Example

Session-4: Collision Block – Completion Example 2

Student Name:

Grade:

Game 4: Police: Catching crashed cars

Code Blocks	Not tried	Incomplete	Completed
1. Edit Breeds			
2. Creating Procedure Block			
3. Using Logic-If Block (movement)			
4. Creating Setup Block (Creating characters and other objects)			
4.1. Set xy & size			
4.2. Setting Camera			
4.3. Using Scatter Block			
4.4. Set Color Block			
4.5. Set & Show Score Blocks			
4.6. Reset & Show Clock			
4.7. Set Status			
5. Collision Block			
5.1. Using If & Math Blocks			
5.2. Using Stamp Block			
5.3. Using Die Block			
6. Creating Run Block			
6.1. Forward blocks			
7. Terrain Screen			

APPENDIX C-6: Rubric for Full Practice

Session-5: Collision Block – Full Practice 1

Student Name:

Grade:

Game 5 : Red Riding Hood: Collecting Carrots

Code Blocks	Not tried	Incomplete	Completed
1. Edit Breeds			
2. Creating Procedure Block			
3. Using Logic-If Block (movement)			
4. Creating Setup Block (Creating characters and other objects)			
4.1.Set size			
4.2.Setting Camera			
4.3.Using Scatter Block			
5. Collision Block			
5.1.Using Die Block			
6. Creating Run Block			
Additional Note:			

APPENDIX C-7: Rubric for Full Practice

Session-6: Collision Block – Full Practice 2

Student Name:

Grade:

Game 6: Crossing Bridge (Collecting coins)

Code Blocks	Not tried	Incomplete	Completed
1. Edit Breeds			
2. Creating Procedure Block			
3. Using Logic-If Block (movement)			
4. Creating Setup Block (Creating characters and other objects)			
4.1.Set size			
4.2.Set Attitude			
4.3.Setting Camera			
4.4.Set & Show Score			
5. Collision Block			
5.1. Set Score + Math			
5.2.Using Die Block			
6. Creating Run Block			
6.1.Forward Block			
6.2.Right Block			
7. Editing Terrain (Creating mountain and lake)			
Additional Note:			

APPENDIX C-8: Rubric for Final Project

Final Project

Student Name:

Grade:

Code Blocks	Not tried	Incomplete	Completed
1. Edit Breeds			
2. Creating Procedure Block			
3. Using Logic-If Block (movement)			
4. Set Heading			
5. Creating Setup Block (Creating characters and other objects)			
5.1.Set xy			
5.2.Set size			
5.3.Scatter Block			
5.4.Set & Show Score			
5.5. Set & Show Clock			
5.6. Set Status			
5.7. Set Camera			
6. Creating Run Block			
7. Collision Block			
7.1.Set Score			
8. Terrain Screen			
Additional Note:			

APPENDIX D-1: Online Participant View Form Questions (Turkish)

- Daha önce programlama dili dersi aldınız mı? Hangi programlama dilini/dillerini biliyorsunuz?
- Programlama dillerinin size en zor gelen 3 yönü nedir?
- Daha önce herhangi bir görsel programlama (Scratch, Alice, Starlogo,... vs) kullandınız mı? Kullandıysanız hangilerini kullandınız?
- Starlogo programını nasıl buldunuz? Kullanırken zorlandınız mı? Ne tür görevlerde zorlandınız?
- Starlogo programı görsel açıdan anlaşılabilir mi?
- Starlogo'da kod yazarken önceden bildiğiniz komutları oluşturmada ne tür zorluklar/kolaylıklar yaşadınız?
- Hem kod yazarak hem 3 boyutlu ortam yaratarak oyun tasarlamanın olumlu ve olumsuz yanlarını yazınız.

APPENDIX D-2: Online Participant View Form Questions (English)

- Have you ever taken a programming language course before? Which programming languages do you know?
- What are the three difficulties of programming languages?
- Have you ever experienced visual programming language before? If so, which visual programming language have you experienced?
- How do you find Starlogo TNG? Were there any difficulties while you were working on it? Which tasks were difficult to accomplish?
- Is Starlogo TNG comprehensible visually?
- What kind of difficulties have you experienced on Starlogo TNG while you were creating code block which you have known already?
- Please write advantages and disadvantages of designing game with both coding and designing 3D environment.

CURRICULUM VITAE

Nomin Boldbaatar

Personal Details

Phone	+90 552 228 3593
E-mail	nominboldbaatar12@gmail.com
Nationality	Mongolian

Education

2014 - 2017 **Ondokuz Mayıs University, Samsun, Turkey**

MSc. in Computer Education and Instructional Technology

2010 - 2014 **Ondokuz Mayıs University** Bc. in Computer Education and Instructional Technology

Conferences

Kaynarca, İ., Kandin, E. & Boldbaatar, N. (2015, May). Kamu Personeline Yönelik Yapılan Hizmet içi Eğitimlerin değerlendirilmesi: Ondokuz Mayıs Üniversitesi Örneği. 9th International Computer & Instructional Technologies Symposium (ICITS), Afyonkarahisar, Turkey.

Boldbaatar, N. (2016, May). Transferring Desktop Experience into Mobile Experience: A Case Study from Cognitive Load Perspective. 10th International Computer & Instructional Technologies Symposium (ICITS), Rize, Turkey.

Sendurur, E., Efendioglu, E., Caliskan, N. Y., Boldbaatar, N., Kandin, E., & Namazli, S. (2017, April). The M-Learning Experience of Language Learners in Informal Settings. 13th International Conference on Mobile Learning, Budapest, Hungary.