

ADVANCING SCOOOL - GAME TYPE RESEARCH AND DEVELOPMENT

A Thesis
presented to
the Faculty of California Polytechnic State University,
San Luis Obispo

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in Computer Science

by
Chanelle Kaith Mosquera

December 2020

©2020
Chanelle Kaith Mosquera
ALL RIGHTS RESERVED

COMMITTEE MEMBERSHIP

TITLE: Advancing sCool - Game Type Research and Development

AUTHOR: Chanelle Kaith L. Mosquera

DATE SUBMITTED: December 2020

COMMITTEE CHAIR: Christian Eckhardt, Ph.D.
Professor of Computer Science

COMMITTEE MEMBER: Dr. Christian Gütl, Univ.-Prof. Dipl.-Ing.
Technical University Graz, Austria

COMMITTEE MEMBER: Foaad Khosmood, Ph.D.
Professor of Computer Science

ABSTRACT
Advancing sCool - Game Type Research and Development
Chanelle Kaith Mosquera

The proposed project, sCool, is an adaptive game-based learning experience designed for STEM education. In this work, we present a new iteration of sCool in efforts to further examine contributing factors of engagement, usability, and comprehension. The newly developed game experience for acquiring object-oriented programming skills is divided into two parts: concept learning and practical challenge. The concept learning part teaches students theoretical lessons of programming through fun gameplay. The practical challenge part allows students to practice programming by completing tasks. This project presents several new game types for both the concept learning and practical challenge parts. The development of these game types spreads across two phases. The first phase introduces two new game types and focuses on extending sCool to support learning object-oriented programming and improve student's learning comprehension. The second phase builds off of the first phase, introducing another new game type to improve the object-oriented programming learning experience and the game's overall usability and engagement. During the first phase, three experiments were conducted in a classroom setting with a computer science teacher. Conducting a study involving a total of 39 school students and three teachers, we are able to successfully display an enhanced understanding of different programming concepts. During the second phase, a single experiment was held remotely among a wide group of people, and the participants were self-guided by an instruction document and the sCool application. Conducting a study with 25 participants, we are able to show a significant improvement in the game's usability and engagement. For future works, further evaluations in-classroom and over a longer course will be useful in assessing the new game type's effectiveness in teaching object oriented programming. Furthermore, the game should be expanded to support learning more complex concepts in object oriented programming.

Keywords: Game-based learning, Explorative Learning, STEM Education,
Computer Programming Education, Engagement

ACKNOWLEDGMENTS

Christian Eckhardt, a supportive and hardworking advisor and professor whom I've had the privilege of working with throughout my time at Cal Poly. You believed, encouraged, and supported me every step of the way.

Christian Gütl, for being a part of my committee and my co-advisor and host during my stay in Austria. You continue to welcome me into your research team and give me the wonderful opportunity to work on sCool.

Foaad Khosmood, for being a part of my committee and professor. You've pushed my learning capability farther and always provided a safe space for me to make mistakes and learn.

Alexander Steinmaurer, my colleague and partner throughout this project. You welcomed me to Austria and continued to advise me throughout the entire process of this Masters thesis.

** Alexander also led some of the experiments of this project at schools in Austria.*

My family and friends for their continued support throughout my education and encouragement to do what I love.

Marshall Scholarship Plan and the Technical University of Graz, for giving me the opportunity to study in Austria where I began my Masters program.

TABLE OF CONTENTS

	Page
LIST OF FIGURES	x
LIST OF TABLES	xi
CHAPTER	
1. INTRODUCTION	1
1.1 Objectives	2
1.2 Expected Outcome	3
2. BACKGROUND AND RELATED WORK	4
2.1 Exploratorative Learning Pedagogy	4
2.2 Explorative Learning and Coding Games	8
2.3 Elements of Effective Game Design	11
2.3.1 MDA	11
2.3.2 Flow	12
2.3.3 Player Types	13
2.4 Coding Games	16
2.5 About sCool	17
2.5.1 Gameplay Modes: Concept Learning and Practical Challenge	18
2.5.2 Earlier Game Types	18
2.5.3 Web Platform	20
2.5.4 Teaching with sCool	21
2.5.5 Motivation for Further Development	21
3. DEVELOPMENT AND IMPLEMENTATION	23
3.1 Terminology	24
3.2 Technologies	24
3.3 Assets	25
3.4 sCool Logo	25
3.5 Base Code Revision	27
3.6 Phase 1: Development	30
3.6.1 Narrative	31
3.6.2 Concept Learning: Scavenger Game Type	32
3.6.2.1 Word Puzzle	34
3.6.3 Practical Challenge: Smartbox	36
3.6.4 Supporting Object Oriented Programming in Python	38

3.6.5 Adaptive Learning	38
3.6.6 Block Builder Prototype	39
3.7 Phase 2: BuildNBreak (GUI and Code) Game Types	42
3.7.1 Concept Learning: BuildNBreak (GUI)	42
3.7.2 Practical Challenge: BuildNBreak (Code)	44
3.7.3 Level Difficulty	46
3.7.4 Revising Object Oriented programming in Python	46
4. EVALUATION PROCEDURE	48
4.1 Instruments	48
4.1.1 Modified Game Engagement Questionnaire	49
4.1.2 System Usability Scale	50
4.1.3 General Questionnaire	51
4.1.4 sCool Game Related Questionnaire	51
4.2 Phase 1: Evaluation	52
4.2.1 Participants	53
4.2.2 Procedure	54
4.3 Phase 2: Evaluation	59
4.3.1 Participants	59
4.3.2 Procedure	59
5. RESULTS AND DISCUSSIONS	62
5.1 Phase 1: Results and Discussion	62
5.1.1 Engagement and Usability	62
5.1.2 sCool Game Related Questionnaire	64
5.1.3 Effectiveness in Teaching	64
5.1.4 Teachers' Impressions of sCool	65
5.2 Phase 2: Results and Discussion	66
5.2.1 Engagement and Usability	67
5.2.2 sCool Game Related Questionnaire	69
5.2.3 Remote Learning with sCool	70
6. CONCLUSION AND FUTURE WORK	71
6.1 Limitations	71
6.2 Future Work	72
REFERENCES	73
APPENDIX	78

LIST OF TABLES

Table	Page
1. Modified Game Engagement Questionnaire	49
2. System Usability Scale	50
3. General Questionnaire	51
4. sCool Game Related Questionnaire	51
5. Overview of participants throughout three experiments of phase 1	53
6. Overview of interviewed teachers	54
7. Phase 1 Experiment schedule	54
8. Overview of concepts and practical challenges in phase 1 experiments	56
9. Phase 2 Experiment Schedule	60
10. Overview of concepts and practical challenges in phase 1 experiments	61
11. Answers to General Questionnaire	66
12. Responses to the sCool Game Related Questionnaire in phase 2	69

LIST OF FIGURES

Figure	Page
1. de Freitas' Exploratory Learning Model [13]	6
2. de Freitas' framework for evaluating games-based education [14]	7
3. Example of Experiential Learning Theoretical Concepts [15]	9
4. GUI-centered approach to teach OOP concepts [16]	10
5. MDA Frameworks [17]	12
6. Flow Diagram [18]	13
7. Bartle's Four Player Types [23]	14
8. Marczewski's Player Types and Motivations [24]	16
9. sCool in-game shot of Top Down Shooter game type	19
10. sCool in-game shot of Robot Missions game type	20
11. sCool web platform, designing a practical challenge	22
12. sCool's history of logos	26
13. Game manager scripts were decoupled using inheritance	28
14. Improving sCool base code using Unity's ScriptableObjects	30
15. In-game shots of sCool's Scavenger game type	33
16. sCool Scavenger game type features a word puzzle	35
17. Smartbox practical challenge - at the start of the game	37
18. Smartbox practical challenge - practicing object-oriented programming	37
19. Initial prototype of BlockBuilder game type	41
20. In-game shot of BuildNBreak (GUI)	43

21. In-game shot of BuildNBreak (GUI)	43
22. In-game shots of BuildNBreak (Code) - coding part	45
23. In-game shots of BuildNBreak (Code) - after code is compiled	45
24. A sample of cards in BuildNBreak game type	46
25. Python Programming Revision	47
26. Practical task on the worksheet after working with sCool	58
27. Comparison of game engagement questionnaire	63
28. Phase 2 - MGEQ results	68

Chapter 1

INTRODUCTION

Many children are born into this digital era — with their first skills being tapping, swiping, and absorbing information through a colorful screen. Even teenagers and young college students managed to catch the digital wave at an early age and are well-immersed in its mass of communication, entertainment, and information. To keep up with the pace of the world around us, we develop digital literacy skills. Many common households have PCs, laptops, or smartphones; it's our main source to develop skills such as processing hypertextual or image-based information, learning through online collaboration, and creating digital content. With young students' early proficiency in digital literacy skills, education and pedagogical methods must shift and align themselves with the demands of the digital era. [1]

The quality of a person's fundamental education is of utmost importance, but is also one of the most challenging experiences. Oftentimes, educational subjects are so intricate and complex that traditional forms of teaching can make students feel overwhelmed, bored, or discouraged. These hindrances can be overcome through motivation and engagement. [2] The video game industry are experts in motivation and engagement. They have a solid grasp on how to help players unleash their newfound skills. This inspires research to propel motivation in education through game-based learning. [3] [4] Game-based learning through technological platforms is highly engaging for students. They provide hands-on learning experiences and offer immediate feedback on the skill

students are practicing. By incorporating these learning tools in classroom settings, teachers can facilitate their students' educational experience, increasing their engagement and motivation in learning. [5] This pedagogical approach can be especially useful in teaching STEM subjects - topics that are heavy with abstract concepts and can benefit from illustrative and interactive explanations. The player's engagement in a game-based learning environment has a positive impact on their learning and information retention. [6]

The situation mentioned above has motivated us to start the research project sCool, which was originally initiated by research groups from the Technical University of Graz in Austria and Westminster University in the United Kingdom. sCool is an educational tool for instructors to teach STEM subjects in a fun and engaging way. It features different types of games where the students can play, learn educational concepts, and test their knowledge. Its corresponding web platform also enables teachers to adapt the game to teach their customized curriculum. [7], [8], [9]

1.1 Objectives

For the further development of sCool, we explore additional game-types and methods for teaching computational thinking and programming. To summarize, the key contributions of this project work are:

- Development of new game types presented in sCool
- Exploration of game design for teaching object-oriented programming concepts

- Improvement of sCool's usability and engagement with the new game types

1.2 Expected Outcome

The overall expected outcome of this project is to provide a game-based learning experience that delivers educational content effectively. We explore game elements that improve the player's knowledge retention and design engaging experiences to improve the player's motivational level.

Chapter 2

BACKGROUND AND RELATED WORK

This section covers methodologies for game-based learning, concepts related to game engagement, and some background history of sCool. We also take a look at other works similar to sCool and establish sCool's importance in the research community. sCool has been an ongoing project since 2018, so much work has been done to bring sCool to its current state. sCool has seen iterations of development and experiments, building off of prior work to improve its effectiveness as a fun and educational tool.

2.1 Exploratorative Learning Pedagogy

Explorative learning is an effective strategy for learning new systems. Without a goal or intended use and application, there is little motivation to learn something. In addition, concepts that were learned through a beginning-to-end tutorial without a need to apply such concepts leads to little retention in the learner. A more effective approach to teaching is using a task-oriented learning methodology, challenging learners with goals to strive for. Another strategy for learning is exploring aspects of the topic - trying things out, discovering challenges, failing, learning from others, and finally overcoming the challenge with a new found knowledge. [10]

Video games are a modern platform for education, thus the standards used to create educational content in traditional methods cannot be transferred over. de Freitas et al. introduces the Exploratory Learning Model (ELM), a framework to help support teachers evaluate educational games and simulations. [11] This model extends Kolb's model of learning by taking into consideration contexts and situations of more common e-learning practices in modern times. [12] There are five components to the ELM as shown in Figure 1. The *Experience* aspect includes both lived and virtual experiences, as well as *transactional* experiences - learning through transactions between task and activity, tutor and learner, or between peers. *Exploration* is the next key aspect in learning - it urges learners to dig deeper and try new things. It also promotes opportunities of engaging with others for collaborative learning. *Reflection*, the third aspect, is necessary for effective learning transfer - meaning that the knowledge learned in this context can be applied in another context. Throughout the experiential learning cycle, learners will begin to *form abstract concepts*, the fourth aspect. Understanding abstract concepts may be quite difficult for learners, therefore depending on the context of the learning event, additional support may be needed to further illustrate the abstract concepts. Finally, in the *Testing* aspect, the learning experience is assessed to determine the game's effectiveness as an educational tool.

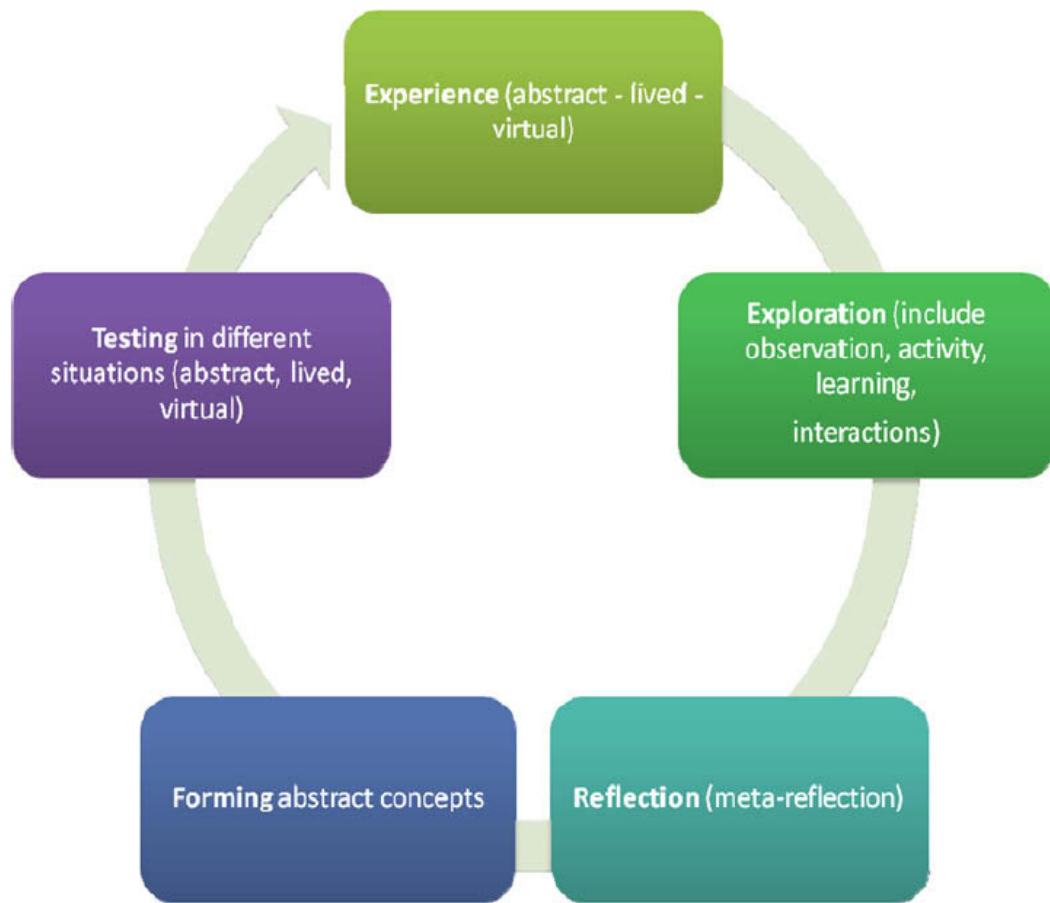


Figure 1: *de Freitas' Exploratory Learning Model* [13]

For this project, we use the ELM to guide and evaluate this version of sCool and ensure its capabilities of introducing programming concepts in a classroom setting.

Alongside the ELM, de Freitas' introduces a four-dimension framework to help teachers evaluate educational games and simulation. [14] As shown in Figure 2, the framework considers four different aspects in an iterative and reflective process. In the *context* dimension, the instructor must consider the resources available in this learning experience. This can include the instructor's own knowledge background, the place or

platform the learning is delivered, or any applicable education background. The *Learner* dimension includes the group of learners themselves and their backgrounds, styles, and preferences. The *Mode of representation* dimension is about the evaluated system's immersion and interactivity; it bridges the immersion from within the game with the process of learning and reflection that happens outside of the game. Finally, the *Pedagogic considerations* dimension is concerned with the instructor's methods of reflecting on the learned material. This framework plays a significant role in the design of sCool's game and the design of the learning experiments with sCool.

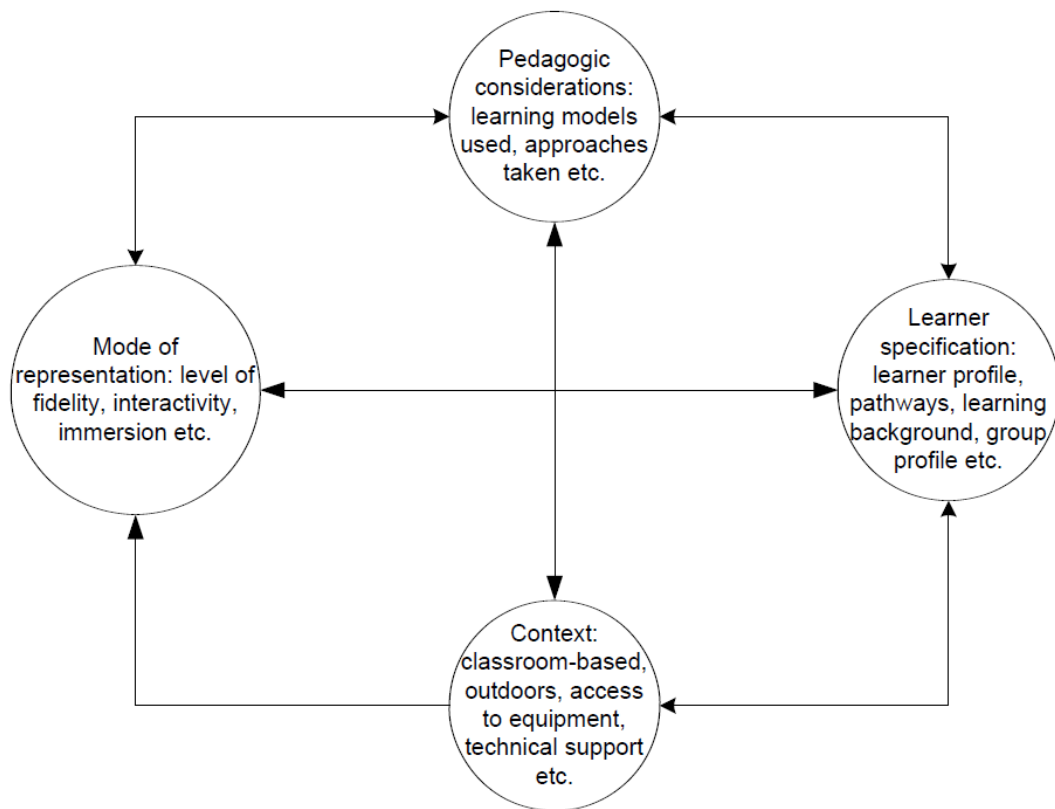





Figure 2: de Freitas' framework for evaluating games-based education [14]

2.2 Explorative Learning and Coding Games

In this subsection, we examine an assortment of educational tools and game-based learning experiences that are comparable to sCool. A handful of research teams across a variety of disciplines are incorporating the explorative learning pedagogy into their projects. Understanding how others are implementing this game-based learning experience is viable knowledge to ensure its implementation in sCool continues to further research in this topic. sCool's continued development ensures it is up to date with current explorative learning pedagogies. There are also many coding games on the market that have influenced and motivated sCool.

Falloon's work uses a game-based learning approach to teach young students science concepts. [15] Similar to sCool, this recent project was developed to further STEM education using the exploratory learning pedagogy. Figure 3 shows a practical challenge that the players must complete to reinforce their knowledge on the learned theoretical concepts. The success of this project provides evidence that given teacher support and appropriate technical tools, this is an effective and engaging way of teaching young students. It also proves that the ELM is a sound framework for designing a learning experience regarding complicated, abstract learning concepts. As opposed to sCool however, this work focuses on teaching scientific theoretical concepts, while sCool focuses on teaching programming concepts.

Challenge 4...

Can you build a circuit so that when a  is pressed the  turns, the  rocks and the  glows?

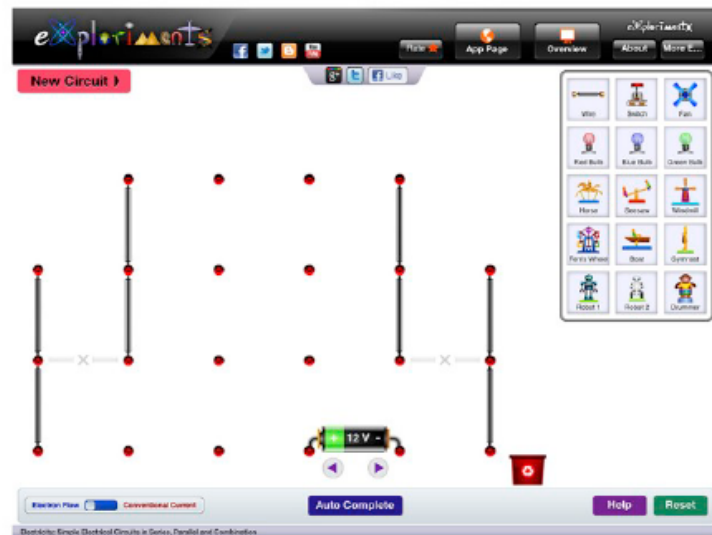


Figure 3: *Example of Experiential Learning Theoretical Concepts* [15]

Yan proposes similar research in teaching object-oriented programming with games. [16]

As shown in Figure 4, Their proposed project uses a GUI-centered approach to teach OOP concepts. The GUI was designed to show the hierarchy of objects in OOP design. In the game, the players practice programming by first exploring the game and observing the different game behaviours, understanding what programming commands lead to those game behaviours, and learning how to create a new world by instantiating objects. The game includes an integrated development environment for scripting using the Java programming language, a compiler, and debugger. The students played the game alongside teacher support. sCool shares a similar goal in providing a game-based tool

that is fun and effective in teaching OOP. sCool aims to provide a more engaging and immersive play experience by considering the MDA frameworks throughout the game design process.

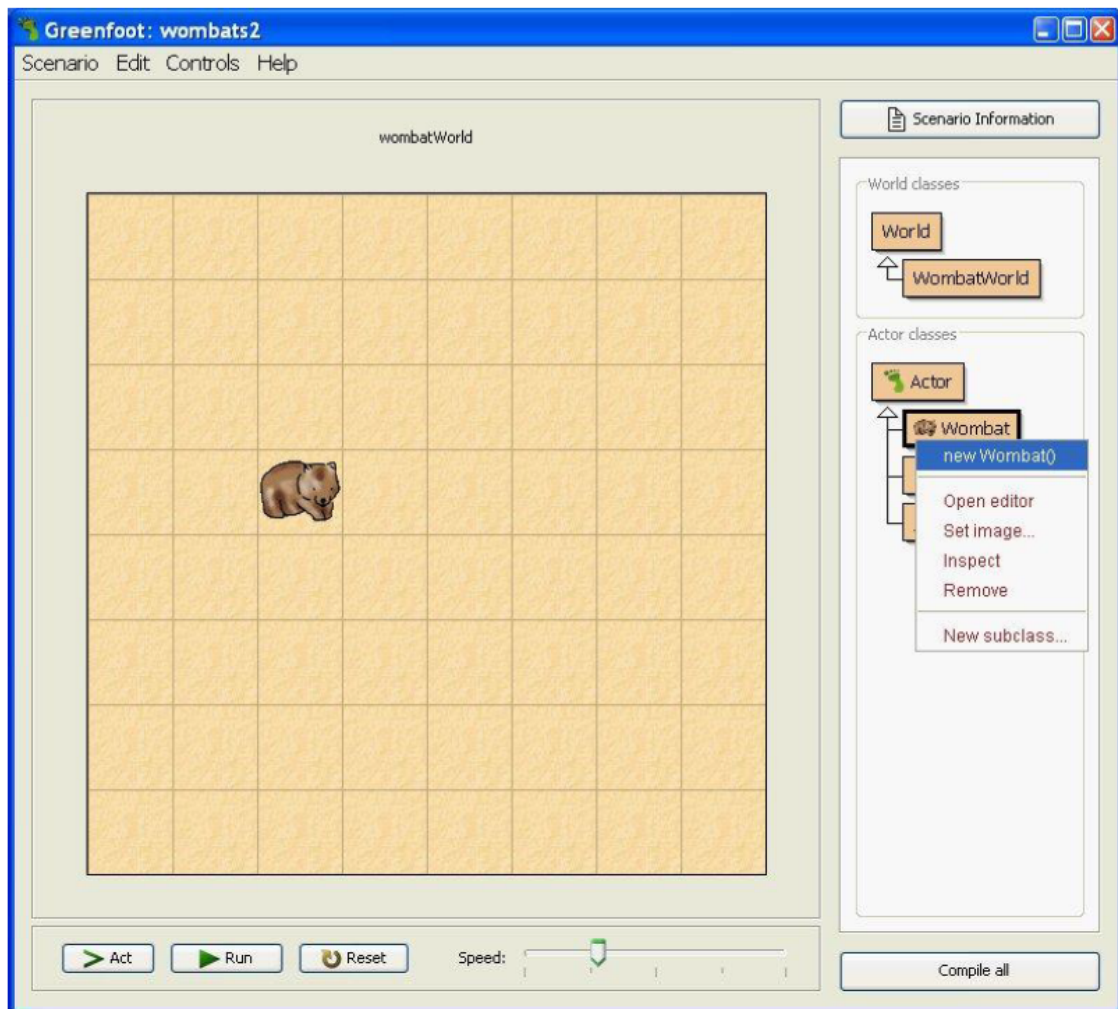


Figure 4: *GUI-centered approach to teach OOP concepts* [16]

2.3 Elements of Effective Game Design

There are many elements in a game that leads to a more engaging and meaningful game design. In this section, we cover the MDA framework, which is used to guide the game design process. Flow describes the state of focus players may experience during the game; it's a desired state that game developers aim to elicit from the players.

Furthermore, different player types play games for different reasons. Thus, it's valuable to know what game dynamics are attractive to each player type.

2.3.1 MDA

The MDA frameworks (Mechanics, Dynamics, and Aesthetics) provides a formal approach to game design and game research. [17] As shown in Figure 5, the MDA framework guides developers through the game design process and provides a user-facing perspective of how the player experiences it. At the foundation of every game are the core *mechanics* that give the player control-- for example, movement, clicking, etc. The *dynamics* of a game are concerned with the interactions between the game mechanics and the player -- for example, timer, dialogue system, etc. The *aesthetics* of a game are concrete descriptions of what makes the game *fun* -- for example, challenge, narrative, etc. Ideally, the game dynamics help elicit the desired aesthetics. A game's aesthetics can also be enhanced through sensual properties, such as graphics and sounds. To tie all three components together, consider the following example. A game that has a *narrative* aesthetic may be fun because of the story it tells. To achieve this aesthetic, the game dynamics would include a sequence of events and a

dialogue system. Finally, for the player to experience the game dynamics, the game's mechanics allow the player to walk through a map and interact with in-game characters. We closely follow the MDA framework and reference it at every step of development.

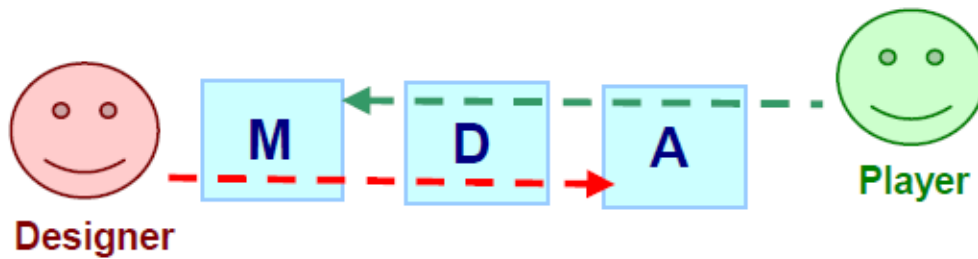


Figure 5: *MDA Frameworks* [17]

2.3.2 Flow

Flow describes the feeling of complete focus and attention when performing an activity. It's associated with high levels of enjoyment and fulfillment and leads to productive performance. According to Csikszentmihalyi, achieving the flow state requires an activity to be balanced between both levels of challenge and player ability. [18] [19] Flow is an integral part of the ELM; it's a reinforcing loop that sustains player engagement and motivates continued exploration. [13]

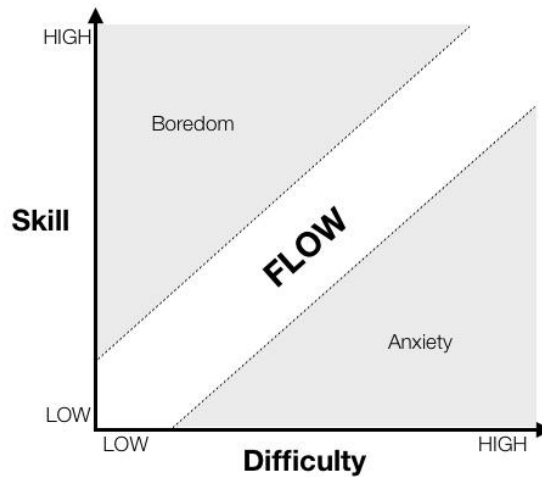


Figure 6: *Flow Diagram* [18]

Kiili et al. proposes a flow framework for game-based learning experiences. [20] The framework was tested with the RealGame business simulation game and proved to be a useful tool for designing engaging game elements and evaluating players' experience and flow state. Based on their findings, flow leads to more effective learning outcomes and promotes exploratory behavior. Game elements that help trigger flow include clear goals, immediate feedback, and a sense of control. When executed smoothly these elements lead to greater concentration, a loss of self-consciousness, a rewarding experience, and time distortion.

2.3.3 Player Types

Player types refers to the categorization of players based on their motivations of play. Within a given context, player types can be a useful tool during the process of game design. [21], [22] By understanding different player types, games can be designed to reach a wider variety of players.

The Bartle Test of Psychology, a well-accepted player type model, was introduced to help understand how players tend to approach games. [23] As shown in Figure 7, This model categorizes players according to four distinct categories: Achiever (10% of player population), Explorer (10%), Socializer (80%), and Killer (1%). For example, achievers are highly motivated by rewards and status, explorers play to discover new things, and socializers enjoy experiencing the game with others. Killers are similar to achievers in that they are motivated to be the best, but are distinguished by their desire for others to lose; this makes less than 1% of the gamer population. It's important to note that although Bartle's model is widely accepted, it represents a generic model. Players tend to show qualities of multiple categories and can differ depending on context.



Figure 7: Bartle's Four Player Types [23]

Marczewski's player type model goes more in-depth with six player types: Achievers, Players, Free Spirits, Socializers, Philanthropists, and Disruptors. As outlined in Figure 8, these player types are motivated by relatedness, autonomy, mastery, and purpose. [24]

Different player types are motivated to play games for different reasons. It could be to achieve goals, collect things, explore the unknown, or socialize and collaborate with others. Analyzing one's player type can be quite complex. However, understanding a generalization of the different reasons for a players' motivation is useful in game design because it guides you to incorporate those elements in your game that will motivate different types of players.



Figure 8: *Marczewski's Player Types and Motivations* [24]

2.4 Coding Games

In recent years, many are looking towards video games for teaching programming. A majority of these games are meant to introduce beginner concepts such as problem solving, algorithmic thinking, and familiarity of coding syntax. Scratch [25] , Blockly [26], and Code.org [27] are programming games that focus first on logical thinking skills and second on coding syntax. Players write code by dragging building blocks into the correct sequence. The game play styles are mostly creative, allowing players to use code to create different visual projects.

CodeCombat [28] and its successor, Ozario [29], offer an online game for teaching many aspects of programming through a fun, fantasy driven game. The game uses a more task-oriented approach. The players must use code to accomplish different tasks and go through the game's narrative. It also offers a web platform for teachers to review the students' work. These two games are very similar to sCool in that they are designed to enhance teacher-student learning and offer a task-based computer science curriculum. All of the games mentioned here are focused on teaching procedural programming — which is programming a sequence of computational steps. This version of sCool aims to explore game-based education for teaching object oriented programming.

2.5 About sCool

sCool is a game-based tool that enhances young students' education in STEM topics, specifically for computational thinking and coding. The project was initiated as a collaboration between TU Graz and Westminster University, UK, and continues its development with RMIT, Australia, and in this project, with Cal Poly SLO. sCool is designed to facilitate the transfer of learning between educators and students. It's system is split into two parts: the mobile/desktop game application combines fun games and educational content for the students, and the web application provides a tool for the instructors to design the educational content embedded in the game and to oversee their students' progress.

2.5.1 Gameplay Modes: Concept Learning and Practical Challenge

There are two modes of play in sCool: a concept learning, or theoretical mode and a practical challenge mode. In the concept learning mode, the players play and explore their virtual environment while learning about the theoretical programming concepts. Afterwards, the students apply the concepts they've learned in practical situations to solve tasks and challenges. This pedagogical approach improves the students' understanding and computational skills in a way that is playful, engaging, and motivating. Both modes of gameplay are task-oriented and story-driven, which are key elements of game-based learning. Students are motivated through clear goals, immediate feedback, and immersion. These two modes of gameplay were well-received by students and instructors as an effective learning tool. [7]

2.5.2 Earlier Game Types

Prior to this version, sCool featured two game types for the theoretical mode, *top down shooter* and *platformer*, and one game type for the practical mode, *robot missions*. The top down shooter features a procedurally generated map that creates terrains of varying sizes with mountains and crevices. The motivation behind this feature is to enable non-repetitive playability and to focus the educators solely on creating the educational content. Figure 9 is an in-game shot of the top down shooter game type showing the procedurally generated map. The map generation also varies based on the difficulty level that the educator defines through the web application. This allows the game to adapt to the teachers' curriculum and their students' learning goals. As shown in Figure 10, The

robot mission game type is part of the practical mode; it adds to the player's learning cycle by challenging them to apply the newly learned concepts. In this mode, the player's goal is to move the robot to the destination using code. Graphical code block templates are provided to assist the players when coding. The available code templates can be configured by the teacher through the web application to better suit the classrooms' needs. Most of the students found sCool's gameplay to be a fun and engaging learning process. Furthermore, after playing the game, they felt more motivated to learn about programming. [8]



Figure 9: *sCool in-game shot of Top Down Shooter game type*



Figure 10: *sCool* in-game shot of *Robot Missions* game type

2.5.3 Web Platform

Teachers use sCool's web platform to create customized learning content for their students. The learning content is grouped in a *course*, which then contains different *skills* for the students to work on. To teach the different skills, teachers create *Theory* and *Practice Tasks*, which define and configure the gameplay levels and learning content for the concept learning and practical modes of play. Figure 11 shows a form for designing practical challenge tasks using the web platform. The web platform was rated sufficiently high in usability (SUS score = 84.25) and nine (out of ten) of the instructors agree they would use this tool in the future. [7]

2.5.4 Teaching with sCool

As for the educational content, studies with sCool have been focused on teaching introductory programming elements in Python, such as variables, functions, and loops. In the latest version of sCool, we expand the Python programming curriculum to include object-oriented programming concepts. A major topic of this paper outlines the design of a game-based learning experience that is well-suited for teaching object-oriented programming.

2.5.5 Motivation for Further Development

Other studies have been conducted with sCool to assess its effectiveness as a learning tool across a variety of young students. The demographics taken into consideration includes age (ranging from 10-20), gender, and school systems. sCool's overall success motivates its continued development presented in this paper. [9] [24]

This project occurs in two phases and will be referred to as *phase 1* and *phase 2* in the following sections. Studies were done after the first phase and motivated its continued development in the second phase. [31]

New Practical Task





Title 

The Title field is required.

Description 

The Description field is required.

Solution 


Difficulty 

The Difficulty field is required.


Enable/Disable code shortcuts in the video game

Print 




Variable 




If statement 



For loop 




Move Left 




Move Right 



Move Up 



Move Down 

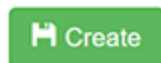


Figure 11: sCool web platform, designing a practical challenge

Chapter 3

DEVELOPMENT AND IMPLEMENTATION

This section covers the design and development process of this version of sCool. A majority of the focus during development time was dedicated to designing and developing new game types for sCool. A large portion of attention was also dedicated to improving the project's base code and framework. This was necessary to keep sCool's base code compatible with updated dependent softwares, as well as to upgrade the framework to support multiple game types.

sCool has undergone two phases of development; we will refer to them as *phase 1* and *phase 2*. To differentiate the contributions of each phase of development, we summarize them as follows:

Phase 1

- New game type for the concept learning game mode, called *Scavenger*
- New game type for the practical challenge game mode called *SmartBox*
- Adding support for object oriented programming in Python

Phase 2

- New game type for the concept learning game mode, called *BuildNBreak (GUI)*
- New game type for the practical challenge game mode called *BuildNBreak (Code)*
- Improving object oriented programming support

3.1 Terminology

It's important to clarify the particular terminology used to explain the development process.

- *Game mode* -- refers to two learning activities in sCool: the concept learning or "theoretical" mode and the practical challenge mode.
- *Game type* -- refers to the different game genres available for each game mode.

3.2 Technologies

At this stage, sCool has gone through a lot of growth and development, thus the project code maintenance is of utmost importance to ensure the project's stability. The sCool game was originally developed using Unity Engine version 2018.3. Midway through, the game was updated to use Unity Engine version 2018.4.23f LTS. This updated version of Unity contains a stable, long-term support, which is ideal since sCool is an ongoing project that will see future developments. With this update, we did a clean manual sweep through the whole project, resolving conflicts in the code and with the entities and components in the Unity project.

We tried upgrading sCool to version 2019.x, however there was a lot of overhead when updating the sCool game project. In addition, version 2019.x led to issues with connecting the sCool game to the online server.

3.3 Assets

A majority of the 3D environment assets from the previous version of sCool are reused. The procedurally-generated landscape tool has many properties that can be tweaked to generate different sized maps and place an array of objects. In order for the tool to be used in the new game types, minor adjustments were made to the base code. Other 3D assets for creating the environment were from the Unity Asset Store. [32] Furthermore, sounds were from Kenney's Assets [33] and some 2D graphics are from FlatIcon [34]. Other systems from earlier versions of sCool which have been adapted and reused are the checker-board grid system, and the tutorial system. Reusing these assets gave the advantage of style familiarity in the game.

3.4 sCool Logo

As sCool continues to expand and reach more classrooms and learners, it's time for sCool to establish its trademarked logo. The new logo's bubbly and jagged typography reflects sCool's aim of being a fun, educational game targeted towards young learners. Its color scheme is influenced by the robot in the original Cool game. sCool's history of logos is shown in Figure 12. It showcases sCool's original logo and icon, the multiple drafts of sCool's new logo, and the official new logo.



Figure 12: *sCool's history of logos*

3.5 Base Code Revision

With two years of development and experimentation, sCool already contains a lot of base code. In efforts to expand sCool with more game types, it was necessary to update sCool's original framework. The main features in the base code that we update includes

- Game type menu to select which game mode and game type to play
- Game manager for the concept learning and practical game modes
- Connection to the sCool database server to get the theoretical and practical tasks
- Tutorial system to guide the player through the game type

One of the main motivations to improve sCool's framework is for code reusability when creating new game types. Each game type has its own set of scripts to run the game, however all of their scripts share the following common functionalities:

sending/receiving data to the server, managing the tutorial, and initiating the popup quiz in the concept learning mode. Decoupling these code elements promotes code reusability and a more consistent code organization. Figure 13 shows a UML diagram of how the base code was refactored to decouple the game type's game manager scripts. In this revision, the LevelManager script is the base class that handles the core functionalities of any game type belonging to the concept learning mode. Earlier game types (grey), as well as the new game types presented in this paper (blue) inherit from LevelManager. Similarly, the PlaygroundManager script is the base class that handles the core functionalities of any game type belonging to the practical challenge mode. The earlier game type, as well as the new game types inherit from PlaygroundManager. The

functions that connect the game to the sCool database server are the same regardless of the game type. Thus, those functions are in the base classes: LevelManager and PlaygroundManager. The child classes are the game type's game manager which contains the rules of the game.

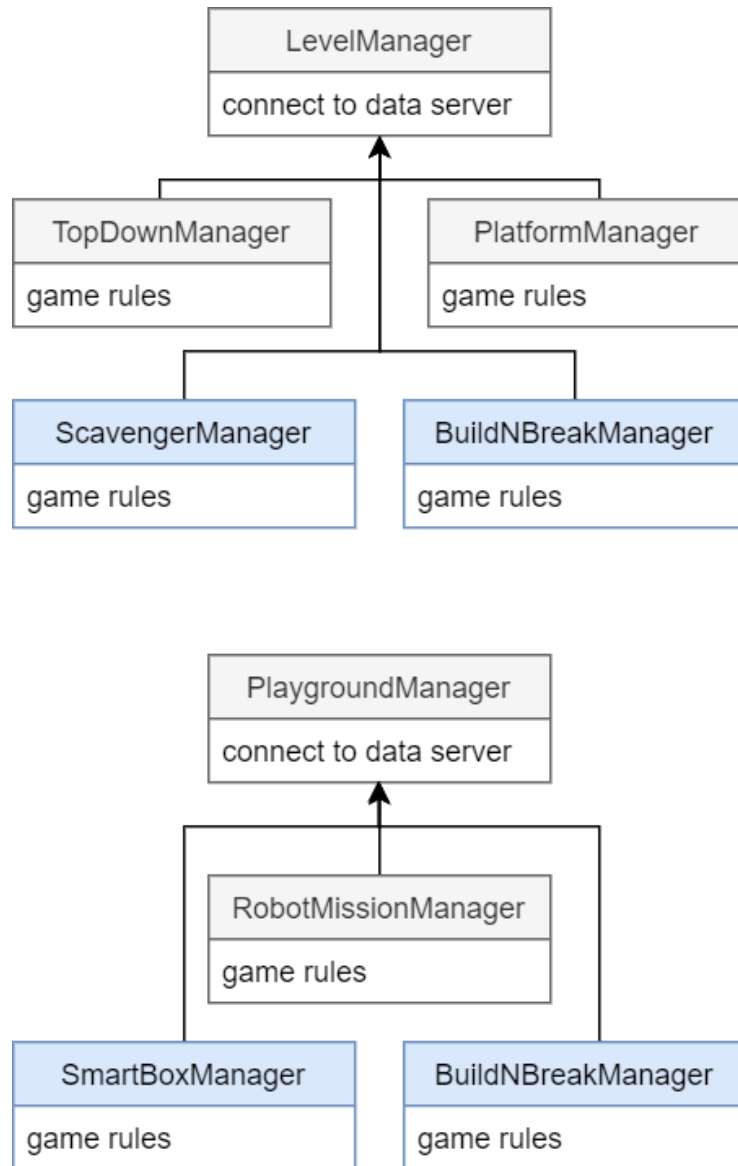


Figure 13: *Game manager scripts were decoupled using inheritance*

One of sCool's earlier game types features a tutorial system which contains a linear sequence of instructions that get triggered throughout the game. This was developed in earlier versions of sCool and its main limitation is that the instructions were hard-coded. To make the tutorial system reusable, we use Unity's ScriptableObjects. The main function of Unity's ScriptableObject class is to save large amounts of data in a container. These objects can then be loaded at run-time. Instead of hard-coding instructions, the instructions are saved in a ScriptableObject, which the tutorial system contains a reference to.

Furthermore, to practice code reusability, we use Unity's ScriptableObjects. The main function of Unity's ScriptableObject class is to save data as an Asset during the development stages, which can then be used in the game at run-time. An example of this use appears in the code for sCool's tutorial manager. During the practical challenge part of the game, sCool's tutorial system contains a linear sequence of instructions that get triggered throughout the game. In previous versions of sCool there was only one game type, therefore instructions for the tutorial were hard-coded in the tutorial manager. Instead, the instructions are stored in a scriptable object. Then, the hard-coded instructions in the tutorial manager script are replaced by a reference to that scriptable object. To visualize this, Figure 14 shows how sCool's Tutorial Manager can be reused in multiple scenes. Each Unity scene has their own Tutorial Manager instance which references data stored in the varying Tutorial ScriptableObjects. With this enhancement, multiple game types can now use the same Tutorial Manager system and have their own set of instructions.

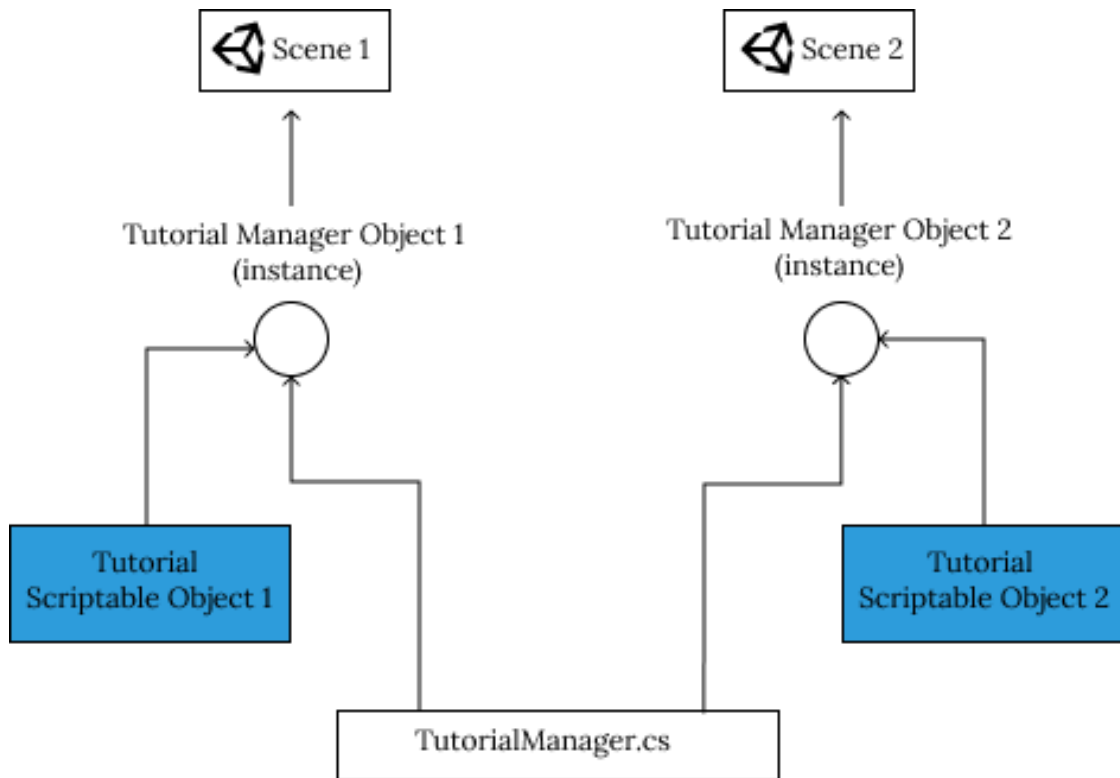


Figure 14: *Improving sCool base code using Unity's ScriptableObjects*

3.6 Phase 1: Development

In phase 1, we introduce a new game type for both the concept learning and practical modes: *Scavenger* and *SmartBox*. The game types from earlier versions of sCool introduced the player to an outdoors/space theme with *Rob* as the main character. Prior experiments with sCool showed that the theme and narrative were fun and well-accepted by the players. Thus, the new game types build off of this theme and narrative by continuing the adventure in an outdoor space.

3.6.1 Narrative

The game is presented as a narrative to immerse players in the environment. In the concept learning part of this game type, the narrative follows an explorer, Rob, who has discovered a vast terrain filled with mountains and crevices. The map contains an abundance of foliage with trees and rocks. Some of those trees appear more unique than the others, a technique used to hint to the player that they are interactive. Players can chop down the tree and collect the scattered wood. A treasure chest appears and Rob finds a scrap of paper that contains only fragments of a sentence. There are more treasure chests spread across the map. Once Rob has found all the scraps of paper, their next task is to piece those scraps together and uncover what it says. The message in the paper is the educational lesson created for this level of the game.

The narrative continues in the practical challenge mode, SmartBox game type. The wood collected from the Scavenger game type is used to build SmartBoxes. These are box-like objects which the player will need to program instructions into. In this game type, the land becomes infested with bugs and the player must program their SmartBoxes to squash the bugs. The narrative is provided through a 3D environment, popup text messages, and feedback cues.

3.6.2 Concept Learning: Scavenger Game Type

The concept learning part of the game has a top-down view of the character in a vast terrain filled with forest-like foliage. The core interactions in this game type are chopping

trees, collecting items, and unscrambling sentences in a word puzzle. The wood-collection system is developed to support the in-game currency feature in sCool. The collected wood appears again later in the practical challenge mode, where wood can be used to build SmartBoxes. Figure 15 contains in-game shots of the Scavenger game type. The game's core mechanics is moving the character around and interacting with other objects. In a mobile device, the controls appear as two input buttons on either end of the screen, one being a joystick for movement and the other a button to trigger interaction. These controls were introduced in an earlier version of sCool and has been well-accepted by users for its ease-of-use and high usability. [8]

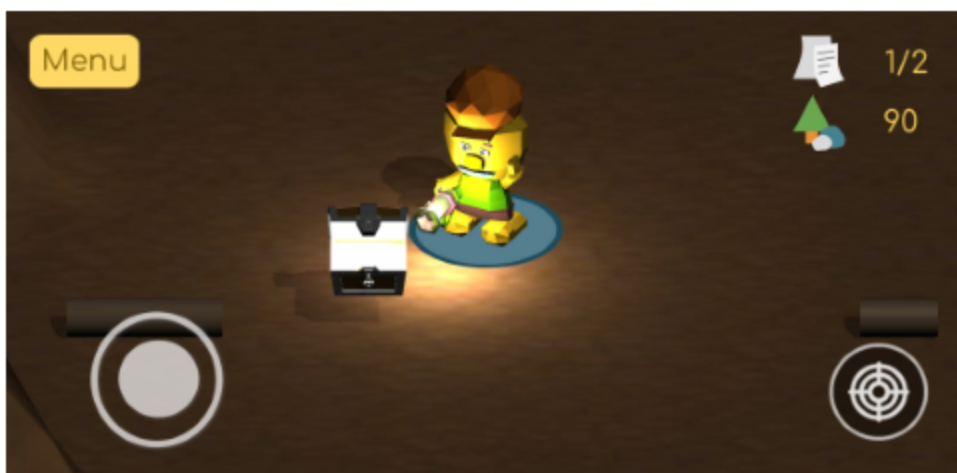


Figure 15: In-game shots of sCool's Scavenger game type

3.6.2.1 Word Puzzle

After all the scraps of paper have been collected, the player is done exploring the map and the scene switches to a word puzzle. This is the part of the game where the player must decipher the message by unscrambling the sentence fragments. As shown in Figure 16, The scene is split into two sides, the left listing all the sentence fragments in random order and the right listing text blocks which will eventually contain the sorted sentence fragments. In this example, the first, second, and fourth sentence fragments have been unscrambled; the third text place-holder (on the right side) is still empty and reads "Place text fragment here...". The puzzle uses a drag-and-drop mechanism where the player must drag the sentence fragments on the left side and drop it onto place-holders on the right side. The bottom of the screen refreshes with the complete sentences that the player is constructing. Once the player is done unscrambling, they press the 'Submit' button at the bottom of the screen. If the sentences were correctly ordered, then the game introduces the programming concept to learn in this level, followed by a quiz to test the students' comprehension. Since the programming concept is taught in a text-based way, the word puzzle is included to better establish concept absorption in the players. This prevents players from proceeding past the educational content without first reading it and having enough understanding such that they were able to piece together the words into cohesive sentences.

Programming concepts are created by instructors using the sCool's web-platform and are stored in the database server. At the start of a level, the game connects to the server

to get the associated programming concept for this level, which is represented as a string. For this specific game type, the programming concept string is divided into several strings and distributed into an array of objects rendered in the scene. The game keeps track of which strings have been found and how many are left.

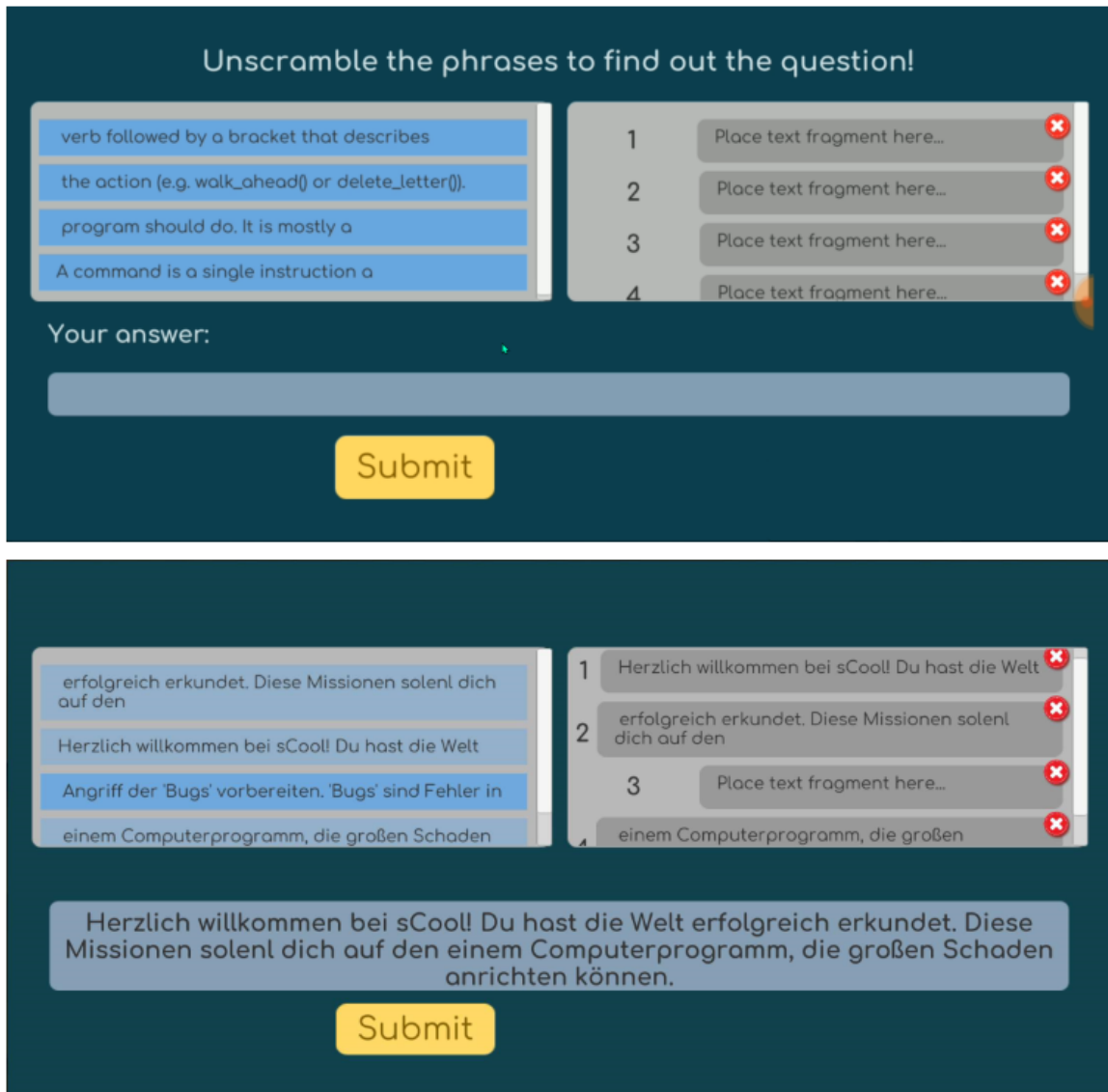


Figure 16: sCool Scavenger game type features a word puzzle

3.6.3 Practical Challenge: Smartbox

In the practical challenge part of the game, the player's goal is to protect their land by catching the enemies using code. The game environment involves a checker-board play area with a combination of boxes and enemies positioned arbitrarily as shown in Figure 17. The play area uses an adaptation of the checker-board from the previous version of sCool. As time passes, bugs are popping up on the field and the player has a set time to catch them all.

The player's main controls in the game is the keyboard and the drag-and-drop coding system. As shown in Figure 18, the keyboard spans across the bottom of the screen and includes all alpha-numeric and punctuation characters. Along the left side of the screen is a vertical bar containing code-blocks (represented as buttons) which can be dragged and dropped onto the scripting area. Code-blocks contain a string that acts as a template for a Python expression. When the player drags it onto the scripting area, that code displays as a string and the player can use the keyboard to edit desired parts, such as variable names or input parameters. For this challenge, the available code-blocks the player can use contain templates for changing a box's color and for moving it.



Figure 17: *Smartbox practical challenge - at the start of the game*



Figure 18: *Smartbox practical challenge - practicing object-oriented programming*

3.6.4 Supporting Object Oriented Programming in Python

The system's support for object-oriented programming was developed by creating a C# class to represent a box object in the game and wrapping it with another class that contains a Python engine. The C# class allows access to the Unity libraries which are necessary for displaying and interacting with the boxes in the game, whereas the Python class allows interaction with this C# class using Python code in-game. Each box is an instance of this class and contains the functions to move the blocks or change their color. At the start of the game, the blocks have no assigned color. The code in Figure 18 shows how the player can access the blocks to change their color. Furthermore, the players can move the blocks to a point in the grid by calling the block's Up(), Down, Right(), or Left() functions.

This game type begins teaching object-oriented programming by showing how to work with multiple instances of the same object. It emphasizes that all blocks have the same functions, but the functions act on the block from which it was called. In phase 2, we improve the game's methods for teaching object oriented programming by introducing constructors and properties.

3.6.5 Adaptive Learning

The sCool game offers an adaptive learning experience by allowing instructors to customize the game difficulty.

In the scavenger game type, the difficulty level is set using the *difficulty* field in the database. This integer value is taken as a parameter in the game when setting up the

level. For example, a higher difficulty will generate a larger map, such as in previous game types. The difficulty will also increase the amount of scraps of paper spread across the map, thus increasing the number of sentence fragments the player must unscramble. Lastly, an increase in difficulty rewards the students with more in-game currency.

The practical challenge's difficulty is also defined by the difficulty field in the database, as well as three other fields that define settings in the game: `NumberOfBoxes`, `NumberOfHidden`, and `NumberOfCoins`. These three fields were created for tweaking some settings in the previous game-type and are thus labeled specific to that context. However, for this game-type, those fields tweak the following settings respectively: the number of boxes the player can program, the number of enemy spawners on the map, and the number of wood the player can collect.

3.6.6 Block Builder Prototype

We began development on the Block Builder game type, which continued into phase 2 and is known as *BuildNBreak*. This section goes over the initial prototype of Block Builder.

Block Builder is a feature in the game that provides a platform for players to be creative and expressive. In this part of the game, the player builds 3D structures made out of a combination of the 3D shapes: cube, cone, sphere, and cylinder. These 3D shapes can be moved, scaled, and rotated to form an unlimited combination of structures. The 3D game environment continues to take place outdoors, decorated with foliage in the

background to continue the immersion of the narrative. An example of a block structure built using the Box Builder can be seen in Figure 19. Along the right side of the user interface are five buttons, the first and top-most are used to create an empty box and the remaining four to create the smaller 3D shapes (referred to in the game as "box bits"). After creating the box, the player can press one of the other four buttons to create a box bit. The box bit appears with a label above it. This label is a property of the box bit class and will be used when referring to it in code.

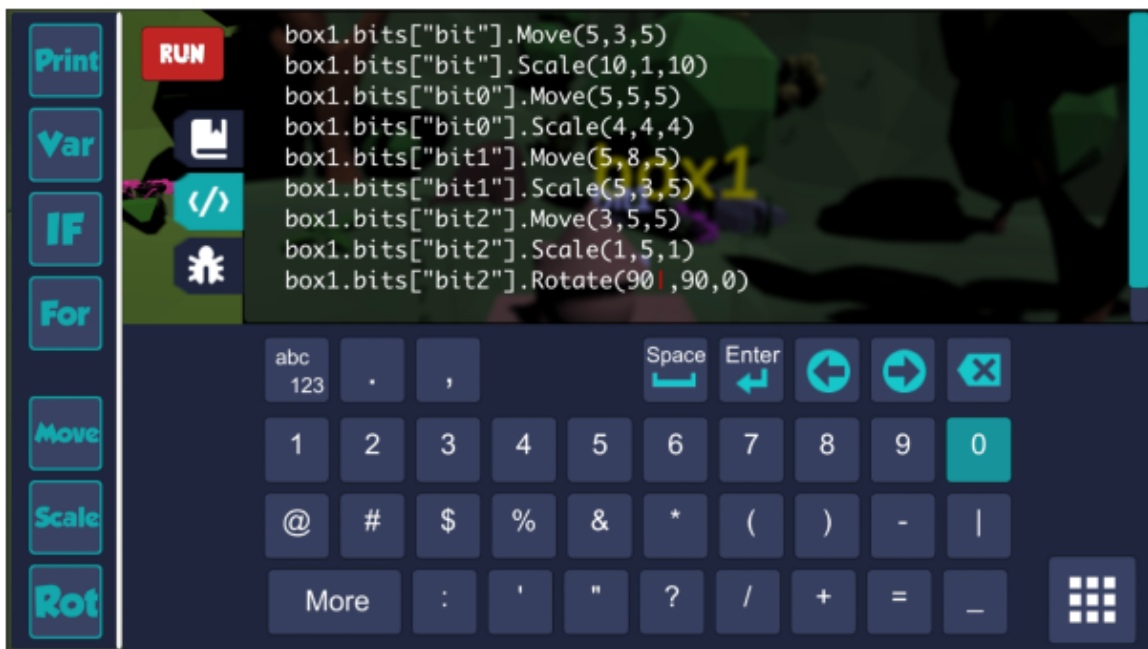


Figure 19: Initial prototype of BlockBuilder game type

3.7 Phase 2: BuildNBreak (GUI and Code) Game Types

In phase 2, a new game type is introduced for the concept learning and practical game modes, *BuildNBreak*. In BuildNBreak, the player is given a card that contains a pattern of blocks. The player must quickly recreate the pattern on the card, then move on to the next card. The goal is to reach the minimum amount of cards and then complete as many cards as possible before the timer runs out. The way BuildNbreak is played is different in both game modes. In the concept learning mode, the player builds blocks using the GUI. In the practical challenge mode, the player builds blocks by coding.

3.7.1 Concept Learning: BuildNBreak (GUI)

At the start of the game, the players are taught the rules of the game and how to interact with the user interface. As shown in Figure 20, the tutorial features a yellow arrow for guiding the user through each step of the tutorial. The left-hand side of the screen contains buttons used for building blocks and changing their color. Figure 21 shows how the player interacts with the game; the player must drag the block or color buttons to the bottom right-hand panel. The square in which they drop the block button reflects where in the grid the block will be placed. The top-right corner displays a yellow card with the pattern that the player must build. After completing the card, the player clicks on the 'Check' button to check their results and then the next card appears.



Figure 20: In-game shot of BuildNBreak (GUI)



Figure 21: In-game shot of BuildNBreak (GUI)

3.7.2 Practical Challenge: BuildNBreak (Code)

Since the players have already learned how to play BuildNBreak in the concept learning mode, they can focus more on the programming aspect in the practical challenge mode.

The coding system -- with the code block templates and the drag-and-drop mechanics -- is the same as the one in phase 1. The only things that have changed are the code

provided by the code block templates, which are the following:

- `cube = Cube()`
- `Sphere = Sphere()`
- `cone = Cone()`
- `__.color = Color.___` `{Red, Yellow, Orange, Green, Blue, Purple, Grey, Black, White}`
- `__.position = (__, __)` `# x,y position`
- `__.scale = (__, __)` `# width, height`
- `__.rotation = __` `#degree angle`

In Figure 22, the player drags and drops the code block templates into the script. They build the blocks by calling the constructors *Cube()* and *Sphere()* and change their properties, *position* and *color*. Figure 23 shows the result after the code is compiled.

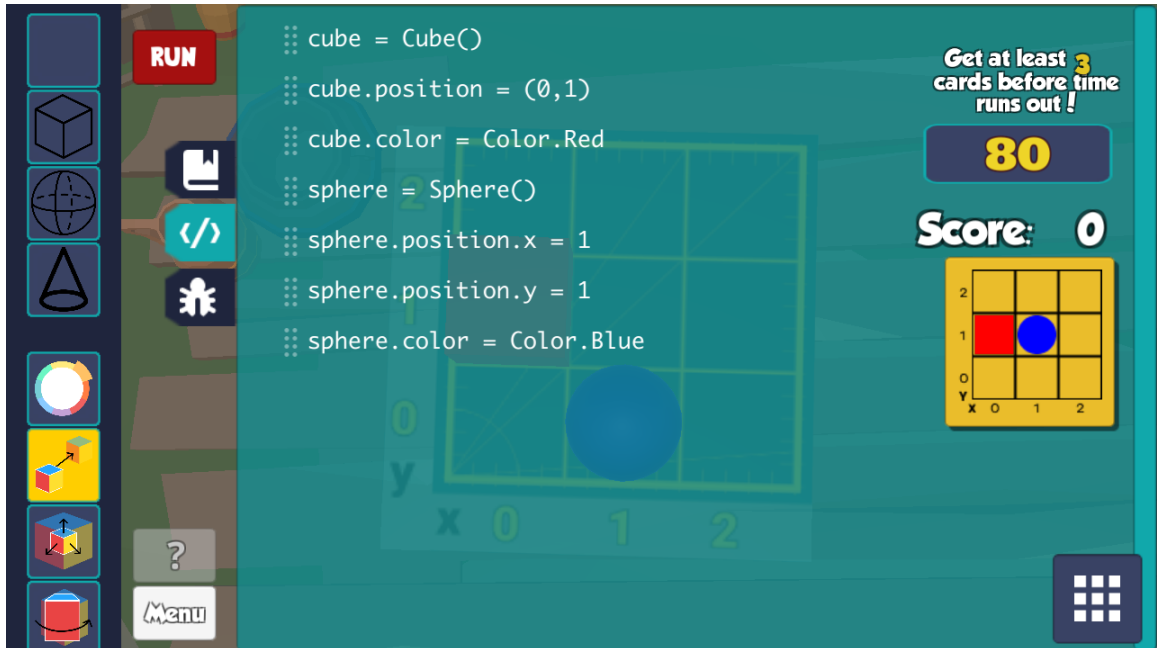


Figure 22: In-game shots of BuildNBreak (Code) - coding part

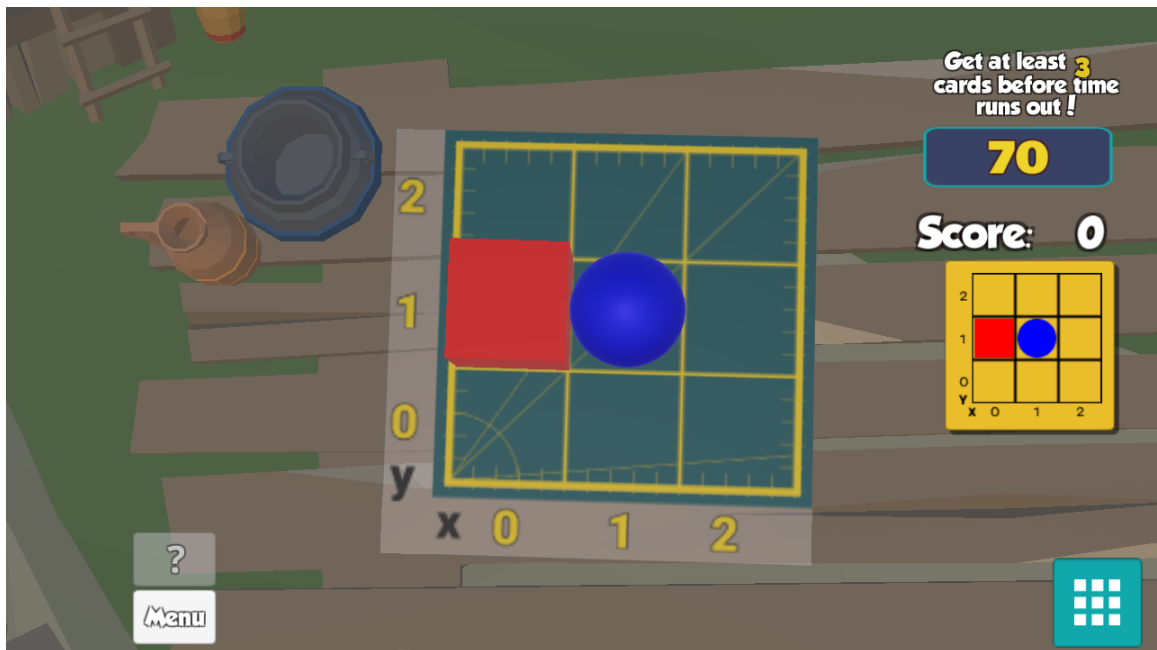


Figure 23: In-game shots of BuildNBreak (Code) - after code is compiled

3.7.3 Level Difficulty

The level difficulty in BuildNBreak varies based on the cards the player must recreate. As shown in Figure 24, The cards show different patterns of blocks that vary by features such as shape, color, position, scale, and rotation. With increasing difficulty, there are additional block features shown on the card. For the concept learning part, the cards vary only by the shape, color, and position. The practical challenge part adds scale and rotation, and all features are edited using Python code.

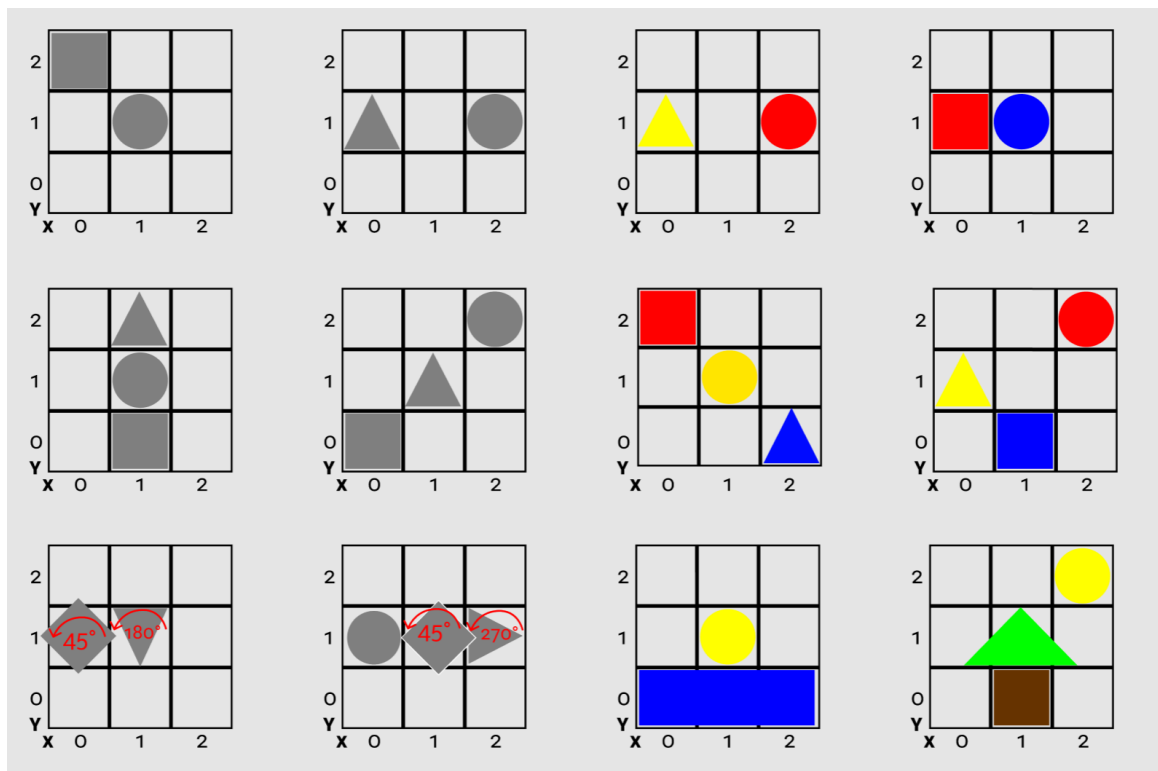
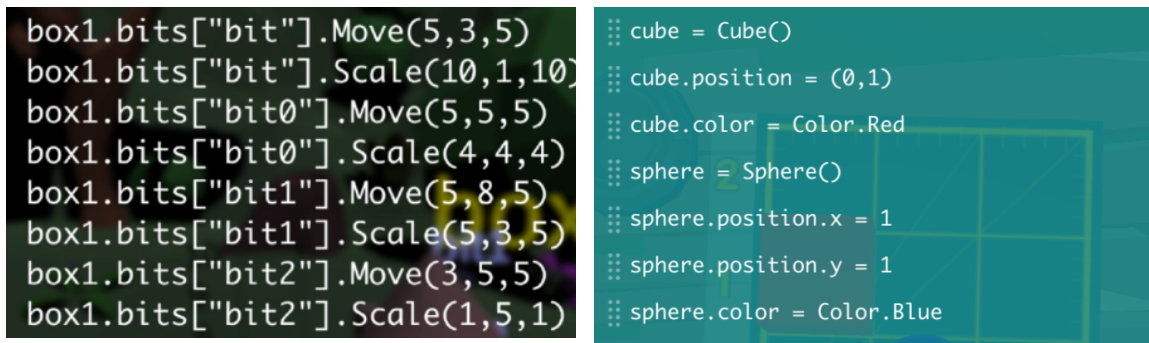


Figure 24: A sample of cards in BuildNBreak game type

3.7.4 Revising Object Oriented programming in Python

The way the Python engine is connected to the Unity game objects is revised in this phase. Previously, in phase 1, objects were defined and instantiated in a C# script and

simply referenced by a Python script. Now, objects are defined and instantiated using a Python script. This enables players to program in an object oriented way directly in Python. This also greatly simplifies the syntax the players need to code in Python. To compare, Figure 25 shows the python code needed to build blocks during phase 1 (left) and phase 2 (right). Notice that in phase 1, players do not have to instantiate the blocks, so they do not learn how to use constructors. To complicate it even further, blocks were accessed by using a dictionary (e.g. *box1.bits["bit"]...*). On the other hand, in phase 2, no blocks have been instantiated yet. The player learns how to use constructors, assign the instantiated objects to a variable, and use that variable to access the object's properties. See Appendix for the complete Python code that defines the classes and variables of a Block object. This script is hidden from the player and is the first to run when the player compiles their Python code. The script can easily be adapted to allow the students more functionality when programming.



```
box1.bits["bit"].Move(5,3,5)
box1.bits["bit"].Scale(10,1,10)
box1.bits["bit0"].Move(5,5,5)
box1.bits["bit0"].Scale(4,4,4)
box1.bits["bit1"].Move(5,8,5)
box1.bits["bit1"].Scale(5,3,5)
box1.bits["bit2"].Move(3,5,5)
box1.bits["bit2"].Scale(1,5,1)
```

```
cube = Cube()
cube.position = (0,1)
cube.color = Color.Red
sphere = Sphere()
sphere.position.x = 1
sphere.position.y = 1
sphere.color = Color.Blue
```

Figure 25: Python Programming Revision
[Left] Programming with the Box Builder Prototype in phase 1. [Right] Programming with the final BuildNBreak game in phase 2.

Chapter 4

EVALUATION PROCEDURE

The quality of sCool as an engaging, game-based learning tool is determined by conducting a series of experiments and using a multitude of evaluation instruments. Since this version of sCool underwent two phases of development and testing, this section is split into two parts: Phase 1 Evaluation and Phase 2 Evaluation. There were a total of four experiments: three experiments in phase 1 and one experiment in phase 2.

In all of the experiments, players first completed a preliminary questionnaire asking about their age, gender, initial opinions of learning in a game-based way, and previous experience with game-based learning tools. Afterwards, the students spent a majority of the time playing with sCool. Immediately after, they complete a questionnaire asking questions related to the game's engagement, usability, and other research-related questions.

4.1 Instruments

During the experiments, we use multiple instruments to measure the game's engagement and usability. We also use a questionnaire to collect general information about the students.

The data was collected in different ways depending on the age group and context in which the experiment was held. In phase 1, secondary school students filled out the questionnaire using a Google Form, which they filled out using their mobile phones or

computers provided in school. The data in the google form was exported into a spreadsheet and analyzed. In phase 2, participants filled out multiple questionnaires across several platforms. Some of the questionnaires were built into sCool, so participants were able to answer them in-game. The remaining questionnaires were accessible through a Google Form.

4.1.1 Modified Game Engagement Questionnaire

The Modified Game Engagement Questionnaire (MGEQ) is based off of Brockmyer et al.'s Game Engagement Questionnaire. [35]It provides a measure of the levels of engagement elicited while playing video games. It includes questions regarding absorption, flow, immersion, and presence. In the MGEQ, there are a total of 13 questions and some were rephrased for better understanding among the participants. Table 1 shows a complete list of the questions in the MGEQ.

Table 1: *Modified Game Engagement Questionnaire*

1	I was spacing out sometimes	Absorption
2	I was unaware of my surroundings while playing	Absorption
3	The game feels real	Flow
4	I get wound up with the game challenges	Flow
5	Playing makes me feel calm	Flow
6	I feel like I just can't stop playing	Flow
7	I can't tell if I'm getting tired	Flow
8	I felt like I was moving automatically while playing	Presence
9	I play longer than I meant to	Presence

10	I lost track of time while playing	Presence
11	I lost track of where I am	Presence
12	My thoughts go fast	Presence
13	I really get into the game	Immersion

4.1.2 System Usability Scale

The SUS is a questionnaire with ten items pertaining to the system's usability. The questionnaire is formatted as a scale, where the participant rates each item one of five responses, ranging from 'Strongly Agree' to 'Strongly Disagree'.

Table 2: *System Usability Scale*

1	I think that I would like to use this system frequently.
2	I found the system unnecessarily complex.
3	I thought the system was easy to use.
4	I think that I would need the support of a technical person to be able to use this system.
5	I found the various functions in this system were well integrated.
6	I thought there was too much inconsistency in this system.
7	I would imagine that most people would learn to use this system very quickly.
8	I found the system was very cumbersome to use.
9	I felt very confident using the system.
10	I needed to learn a lot of things before I could get going with this system.

4.1.3 General Questionnaire

The general questionnaire collects information about the students' age, gender, and their opinions or past experiences with game-based learning.

Table 3: *General Questionnaire*

	Question	Type
1	Please choose your gender.	Single-choice {Male, Female, Other}
2	How old are you?	Open answer
3	What was your name in the game?	Open answer
4	Do you think apps are a great way to learn something?	LikeRT
5	I would be more motivated to learn in a classroom using games	LikeRT
6	Do you already have programming experience?	Single-choice {None at all, A bit not much, I've taken a few classes, I have a lot of experience}
7	If so, in which programming language?	Open answer

4.1.4 sCool Game Related Questionnaire

After playing the game, students were asked questions related to their experience playing and learning with sCool. There are two versions of this questionnaire for phase 1 and phase 2. The questions are outlined in Table 4.

Table 4: *sCool Game Related Questionnaire*

Question
<i>Included in phase 1 and phase 2</i>

1	The game's structure is coherent
2	The game's control is easy to use
3	The usage of the keyboard was easy
4	The levels were easy to master
5	I learned something while playing the game
6	sCool encouraged me to learn more about programming
7	The type of game is fun
8	Playing the game was pleasing
9	Programming was fun
<i>Included in phase 1 only</i>	
10	The language is understandable
11	The work sheet was easy to solve
12	The character Rob is appealing
13	The theme space is interesting

4.2 Phase 1: Evaluation

The focus of the studies in phase 1 is to show how well the new game types are accepted as an engaging and motivating tool for learning and how effective they are in teaching in various classroom settings. We evaluate sCool to answer the following questions:

- How is the new game type perceived by the students from different school types?
- Can sCool's new game type help the students to understand certain coding concepts?

- How effective are sCool's new game types for teaching introductory object-oriented programming?

4.2.1 Participants

For this evaluation, we focused our target audience on secondary school students (ages 12-13) since this educational level includes students with adept reading comprehension and basic digital competence for navigating devices with touchscreen or mouse-and-keyboard mobility. Another priority for this project is sCool's scalability, therefore we performed the evaluations in varying classroom settings: one is a regular secondary school and the other an academic secondary school. The age of all participating students remained similar (7th grade in the regular secondary school and 8th grade in the academic secondary school). By reaching out to different school types, we were able to diversify the participants who may be coming from different social backgrounds and educational experiences. Table 5 provides an overview of the participants in each experiment.

Table 5: *Overview of participants throughout three experiments of phase 1*

	Group 1	Group 2	Group 3
Grade	7th	8th	8th
Age	12–13 years (M=12.27) (SD=0.22)	12–15 years (M=13.69) (SD=0.39)	13–14 years (M=13.5) (SD= 0.27)
Gender	10 boys, 2 girls	13 boys	6 girls, 8 boys
Coding Experience	0 persons	3 persons	8 persons

4.2.2 Procedure

The data for this study was collected from each participant who had given consent. (In this study, all participants gave their consent.) Furthermore, the data was assessed with anonymity since each students' data were listed under pseudonyms.

Table 6: *Overview of interviewed teachers*

	Teacher 1	Teacher 2	Teacher 3
Gender	female	male	male
Experience	15 years	1 year	30 years
Education	college of education	university	IT certifications
School Type	secondary	secondary	secondary/college

Each experiment in the study took place in a classroom setting with both the main instructor for that class and a computer science teacher present at all times. Table 6 gives an overview of the teachers in each experiment. All the students were instructed to work in pairs. Depending on the classroom's resources, the pairs of students completed their tasks either with their own device or with a shared one.

Table 7: *Phase 1 Experiment schedule*

Time	Task	Method of Instruction
5'	Welcome, Introduction of project	Instructor only
10'	Motivation for programming, Brainstorming	Altogether
5'	Introducing sCool and game types	Altogether
5'	Form groups, Installation	Per group
50'	Play sCool	Per group

10'	Complete worksheet	Per individual
10'	Answer questionnaire	Per individual
5'	Final discussion	Altogether

An overview of the schedule during the experiment is shown in Table 7. At the beginning of the study, the computer science teacher introduced the project to the students, as well as an introduction to the idea of programming. This introduction is necessary to give the students a brief overview of what to expect. Together, the class went over the setup process and installed them in all the devices used for this experiment. A majority of the time was spent playing sCool, completing the tasks and challenges. Accompanying the lesson plan taught in the game, the students practiced and transferred their learning by completing a worksheet. Finally, the students completed the questionnaire and ended the experiment with a class discussion.

The educational content presented in the game was created by the computer science teacher using the sCool web-platform for teachers. Table 8 shows the concepts and practical challenges in the course. The educational lesson was presented in the game in a text-based format. The lessons taught object-oriented programming concepts in a practical way by focusing more on what the students will code in order to accomplish certain tasks. There were a total of five tasks to complete in the concept learning part of the game - each task concludes with the educational lesson, followed by a quiz. The five tasks were completed linearly because their associated lessons were structured such that their comprehension is best understood in that order. Then, the students completed

two tasks in the practical challenge portion of the game. Again, the two challenges were completed in a linear order, the second being more difficult than the first. Programming was required for both of these challenges. The difficulty was made more challenging by increasing the number of enemies spawned, which translates to the students needing to write code in a more thoughtful and strategic way.

Table 8: Overview of concepts and practical challenges in phase 1 experiments

Mission	Concept	Question
<i>Concept Learning Part</i>		
Task 1	Introduction	What are bugs?
Task 2	Objects	Which two terms characterize objects?
Task 3	Properties	What means label in terms of objects?
Task 4	Properties	Which command renames an object?
Task 5	Methods	Which command will move an object?
<i>Practical Challenge Part</i>		
Task 1	Objects	Defeat five bugs using both blocks
Task 2	Objects	Defeat fifteen bugs using both blocks

To determine whether this version of sCool is a viable tool for helping students understand certain coding concepts, the students were given a worksheet to complete after playing sCool. As shown in Figure 26, The worksheet contains an assignment that is similar to the practical challenge the students had to work through in the game. It was intentional for both the practical challenge and the worksheet assignment to be similar and mainly differing by the medium it is presented in because we are interested in

seeing how well the knowledge transfers in another context. The two were similar in that the problem was on a ten-by-ten grid along the x and y axes and the students were responsible for guiding multiple objects to certain locations. In the in-game practical challenge, the students had to program the boxes to move towards the bugs by noting their positions on the XY-coordinates of the grid using the on-screen keyboard. Similarly, in this worksheet assignment, the students had to determine the correct commands that will move the bikers along the XY-coordinate to their correct house. However, instead of programming, they had to hand-write the code using the Python functions they learned while playing the game.

Towards the end of the evaluation, the students were given a post-questionnaire with questions concerning their general impressions and their experience with sCool's new game type, specifically its levels of engagement and its system usability. The questionnaire was answered through Google Forms. Game-related data was also sent from all the devices to the servers. This data is stored in a Microsoft SQL database and was reviewed in a visual format using sCool's web-platform. The data collected regarding the general and game-related questions were analyzed and visualized using Microsoft Excel, and both the game-engagement and system usability were analyzed using the open source programming language R to categorize the factors and calculate the mean and standard deviation.

Right after the experiment, we conducted an evaluation from all three teachers who were present the entire duration. The structure of the evaluation was a face-to-face

interview. The questions were prepared beforehand and were concerned with the teachers' perspective and opinions of sCool as a learning tool in the classroom. An important aspect of this project is its pedagogical approach in the classroom, therefore the centered focus of the interview was to gain insight from the teachers' opinions of sCool's new game type and how effective its approach is. At the end of the interview, the teachers provided additional comments they had. All of this data was collected and analyzed using Microsoft Excel. [29]

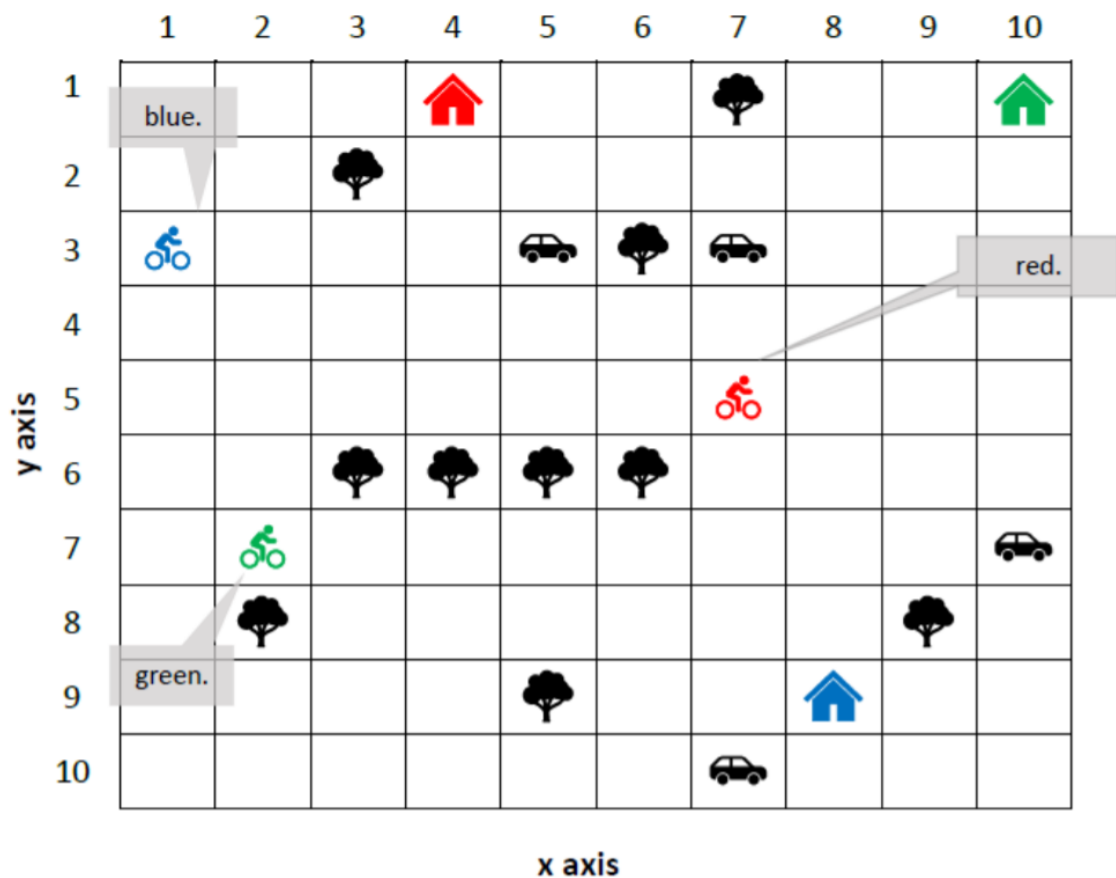


Figure 26: Practical task on the worksheet after working with ,sCool

4.3 Phase 2: Evaluation

Since phase 2 was motivated by our findings from the phase 1 evaluation, the focus of phase 2 is to improve the engagement and usability levels of the new game types. The game types in phase 2 were evaluated remotely and the participants were provided with a written document to self-guide them through the experiment process.

4.3.1 Participants

Since this study was held remotely and self-guided, we targeted older participants who are at least in high school and beyond. This ensured that the participants are capable of following written instructions throughout the experiment and are digitally competent to install and use the application. A total of 25 participants were involved in the phase 2 experiment, 12 male and 13 female. The age of the participants spread across varying age groups: two are ages 18 or less, ten are ages 19-24, ten are ages 25-29, and two are ages 30 or older.

4.3.2 Procedure

Table 9 provides an overview of the experiment schedule. At the start of the experiment, the participants were provided with a document to guide them through the experiment. Due to the remote, self-guided circumstance of this experiment, the document was revised multiple times in efforts to improve its accessibility and debug technical challenges with the installation process. The participants were given specific instructions for setting up sCool on their Windows or MacOS laptop/computer. After opening the

application and registering as a new player, the participants were prompted to fill out the general questionnaire in the game.

Next the participants played sCool and completed a total of six tasks: three tasks each for the concept learning and practical challenge modes. Table 10 outlines the tasks the players completed. The skills learned in each task builds off of the last, so the players were instructed to complete them in linear order.

After completing all of the tasks, the players completed the first part of the questionnaire to assess the engagement levels experienced during the game. This was provided in-game, so the players did not have to exit the game to complete the questionnaire.

The second part of the questionnaire assessed the system's usability and asked questions about the participants' overall experience with sCool. These questions were provided with the Google Form, *sCool Phase 2 - Questionnaire (part 2)* [36].

Table 9: Phase 2 Experiment Schedule

Time	Task	Method of instruction
1'	Project and sCool Introduction	Direct Message
1-3'	Setup and Installation	User Guide
1'	Pre-questionnaire	User Guide
15-30'	sCool Gameplay	User Guide
10'	Post-questionnaire (in-game)	User Guide
5'	Post-questionnaire (Google Forms)	Direct Message

Table 10: *Overview of concepts and practical challenges in phase 1 experiments*

Mission	Concept	Question
<i>Concept Learning Part</i>		
Task 1	Introduction	What is object oriented programming?
Task 2	Constructors	How do you create an object of the class Cube()?
Task 3	Properties	Let “myCube” be a reference to an object of the class Cube. How do you change the cube’s color to Green?
<i>Practical Challenge Part</i>		
Task 1	Constructors	Get as many cards as you can before the timer runs out! Make sure to get at least 2 cards.
Task 2	Properties	Get as many cards as you can before the timer runs out! Make sure to get at least 3 cards.
Task 3	Properties	Get as many cards as you can before the timer runs out! Make sure to get at least 3 cards.

Chapter 5

RESULTS AND DISCUSSIONS

5.1 Phase 1: Results and Discussion

We performed initial tests during phase 1 which included a total of 39 students (8 girls and 31 boys). The tests were completed in 3 separate groups and the experiment setups were identical. We assessed the students' performance in completing tasks and assignments in the game and used a post-questionnaire to assess the game's usability and the levels of engagement and absorption felt by the players.

5.1.1 Engagement and Usability

The results of the MGEQ is summarized in Figure 27 . The distance between the level of immersion and presence is higher at the first experiment. This may be due to the fact that students in the first group are not familiar with game-based approaches in a computer science class and are more engaged in virtual environments. The level of absorption is similar in all groups.

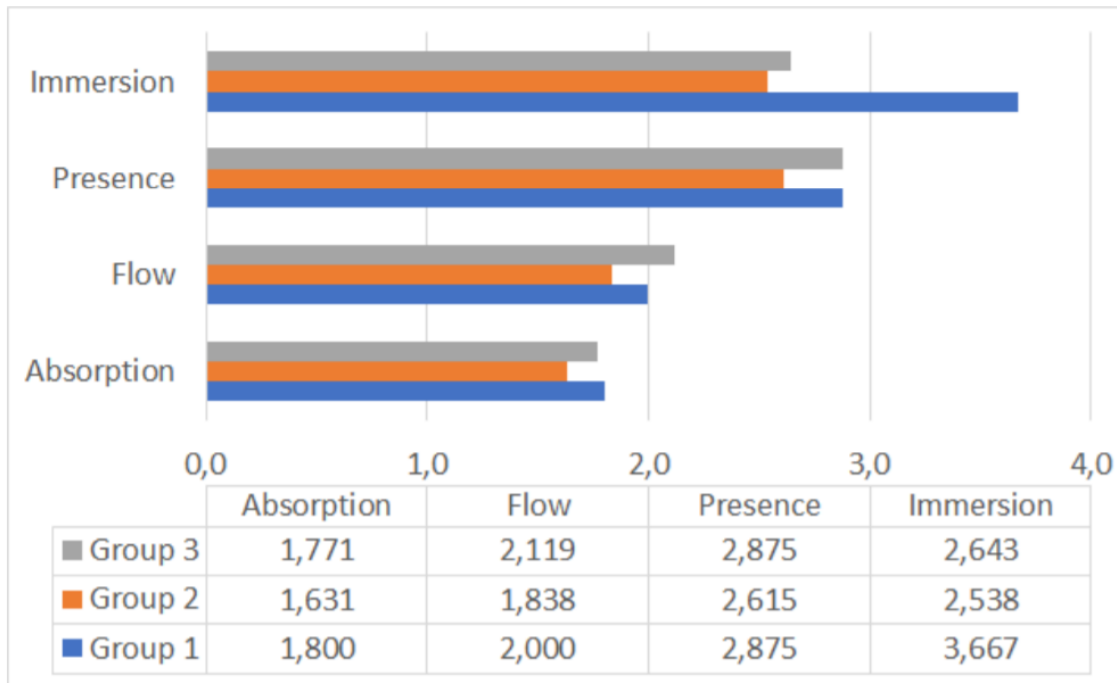


Figure 27: *Comparison of game engagement questionnaire*

The system's usability score (SUS) is above average, reaching a value of approximately 70.8 across all three groups:

- SUS Group 1: 70 (SD=13.7)
- SUS Group 2: 67.9 (SD=14.6)
- SUS Group 3: 74.5 (SD=17.3)

The system usability score was determined based on the students' answers to a post-questionnaire. For context, all the students are comfortable with mobile devices (smartphones) and have even played mobile video games before and at least 30.8% have practiced programming in a game-based way before. Therefore, the students' perspective on the systems' usability is reliable and informative. In the questionnaire,

the students were asked to rate the in-game controls and the keyboard used for the programming portions of the game. 87.18% of the students agreed that the in-game controls were user-friendly and easy to grasp. Furthermore, 92.31% agreed that the keyboard was also straightforward to use. Improvements to the system's usability were suggested by other students such as adding mouse cursor support and extra keyboard support. These features were provided during experiments in phase 2.

5.1.2 sCool Game Related Questionnaire

The game's level of engagement was assessed based on the students' responses in the sCool game related questionnaire. 31 students (79.49%) replied sCool provided a fun experience for learning and practicing programming concepts. Since this version of sCool introduces a new game type, we also inquired about this game type's impression and 33 students confirmed that this game type is enjoyable. 30 students (76.92%) felt confident that they have learned something during this experience and at least more than half of the total students (56.41%) stated that they felt encouraged to learn more about programming.

5.1.3 Effectiveness in Teaching

sCool's effectiveness as an educational learning tool is assessed based on the students' in-game performance and their ability to transfer the knowledge in the post-game worksheet. All students were able to solve the concept learning tasks of the game. 35 out of 36 students were able to pass the first practical task and 28 students proceeded to pass the second, more difficult task. Group 3 was the most successful group with a

100% student completion of both tasks. The worksheet was graded based on Python syntax and correctness. 38 of all students solved at least one task correctly and 26 of these students continued to solve all the other tasks.

5.1.4 Teachers' Impressions of sCool

As for the three teachers' speculation, all of whom have experience with incorporating apps in their class. They recognize that sCool's most essential strengths are

- the ability to add and create their own content into the app
- its integrability in class
- its overall simple usage (accessibility, installation process, etc.)

Only one of the teachers instructs a programming class, but each trusts that they would use sCool as a tool for introducing computational skills and coding. All the teachers believe sCool is best fit for introducing new topics and learning to apply concepts and that it is best suited for students in secondary school, specifically from grade 5 to 8. In regards to sCool's strengths and weaknesses, the teachers agree sCool serves a strong approach to motivating students to learn programming in a different way. They also agree with sCool's design allowing individual assessments of students in-game, focusing on a student-centered class where each student can be coached. The current disadvantages of the game are its user interface and design. The application should provide a more self-explanatory experience. We explore solutions to this concern in phase 2.

5.2 Phase 2: Results and Discussion

Studies in phase 2 included 25 participants doing remote, self-guided participation. We assess sCool's ability to be a self-explanatory experience through the participants' task completion and correctness. We also use a post-questionnaire to assess the game's usability and the levels of engagement and absorption felt by the players. Most of the participants agree that education can be expanded and enhanced using technological tools. More specifically, 92% believe that apps are a great way to learn something and 72% believe they would be more motivated to learn in a classroom in a game-based way. A majority of the participants (19 out of 25) have had at least a little bit of programming experience before; whereas six participants had none at all. Answers to the general questions are reviewed in detail in Table 11.

Table 11: *Answers to General Questionnaire*

Question	Answers
Do you think apps are a great way to learn something?	6 -- Strongly Agree 17 -- Agree 0 -- Neither agree nor disagree 0 -- Disagree 2 -- Strongly disagree
I would be more motivated to learn in a classroom using games.	4 -- Strongly Agree 14 --- Agree 4 - Neither agree nor disagree 1 -- Disagree 2 -- Strongly Disagree
Do you already have programming experience?	6 -- None at all 6 -- A bit not much

	4 -- I've taken a few 9 -- Yes I have a lot
If so, in which language?	C#, Python, C++, CSS, HTML, Typescript, Javascript, Java

Providing the modified game engagement questionnaire (MGEQ) in the game strengthens the accuracy of the results. Since the players didn't have to exit the game to complete the questionnaire, the players remained immersed and present in the virtual environment. To further encourage immersion, the visual style in which the questions were presented remains consistent with the rest of the game.

5.2.1 Engagement and Usability

The game type's engagement levels were assessed based on the six participants who did not have prior coding experience. This provides insight into sCool's engagement levels not just as a game, but as one with an educational goal. Figure 28 shows the detailed results of the Modified Game Engagement Questionnaire (MGEQ). According to the results of the MGEQ, there was a high level of immersion ($M=3.68$, $SD=0.69$) and presence ($M=3.289$, $SD=0.689$), meaning the participants were highly aware of the virtual environment. This can be attributed to the game's style and gameplay consistency, as well as game element enhancements such as sounds, particle effects, and narrative aesthetic. Flow is a very important aspect of the player's engagement — it describes the player's focus and attention, thus driving their motivation levels to

complete the educational task. The level of flow ($M=3.139$, $SD=0.716$) increased by 62% as compared to phase 1.

game engagement (phase 2)

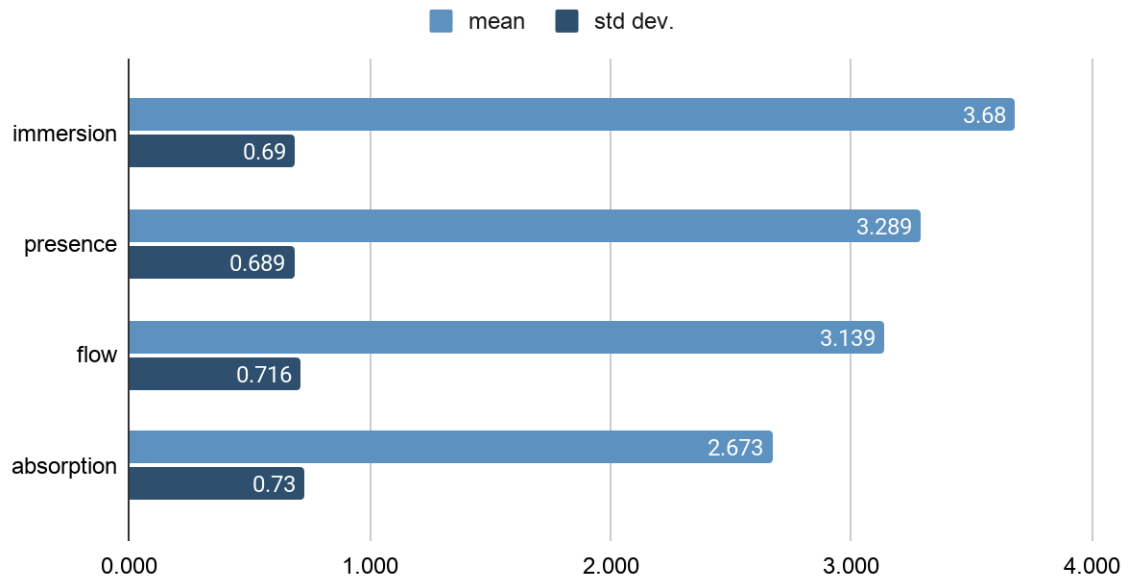


Figure 28: Phase 2 - MGEQ results

A total of 11 participants responded to the System Usability Scale (SUS). The SUS score has improved since phase 1 (score=74.0, $SD=12.9$). The improvement in usability can be attributed to the target group being an older age and thus more digitally competent. However, the SUS score is still significant because the experiments were self-guided and participants had to navigate through sCool without any assistance. Since no clarifying instructions were provided during the experiment, it was very important for the system's use to be clear and understandable. All of the participants were able to complete all six tasks. Thus, the SUS score is highly reflective of sCool's success in usability.

5.2.2 sCool Game Related Questionnaire

According to the sCool Game Related Questionnaire, 10 out of 11 participants agreed or strongly agreed that the game is logical and understandable, and 8 agreed or strongly agreed that they learned something during the game. These findings positively reflect on the organization of the learning concepts in the game, that each lesson built off of the other in a logically sound way. Overall, more than half of the participants claim to have learned something during the game and enjoyed both the game type and programming. Responses to this questionnaire are further detailed in Table 12.

Table 12: *Responses to the sCool Game Related Questionnaire in phase 2*

Question	Strongly Agree	Agree	Neither	Disagree	Strongly Disagree
The structure of the game is logical and understandable	4	6	1	0	0
The levels were easy to master	3	4	2	1	1
I had learned something during the game	3	5	2	1	0
sCool motivated me to learn more about programming	2	4	5	1	0
I found the type of game fun	5	1	5	0	0
I enjoyed the game	4	5	2	0	0
Programming was fun	4	5	2	0	0

5.2.3 Remote Learning with sCool

This is the first time experiments with sCool were conducted in a remote setting without an instructor. All participants, 12 (out of 25) having little to no coding experience, were able to successfully complete all the tasks. This shows great improvement since phase 1, where students were given an introduction to programming before the play experience. The improvement in understandability can be attributed to the phase 2 game types' improved tutorial system and game usability.

Chapter 6

CONCLUSION AND FUTURE WORK

In this paper, we present the continued design and development of sCool. The goals of this project is to explore the educational impact of the new game types. The impact of a game-based learning experience depends on the learning pedagogy, as well as the engagement levels of the game. Phase 1 of this project is focused on introducing a game type for teaching object oriented programming. The game's effectiveness as a learning tool was evaluated by the students' ability to complete the practical challenge tasks, and by testing how their knowledge transfers over in a final worksheet assessment.

Conducting a post survey regarding the individual experience with middle school students as well as their teachers displayed a great acceptance of usability, high engagement, and an increased affinity to further investigate programming skills. They provided further recommendations on how to improve the game's usability and engagement, thus motivating phase 2 of this project. In phase 2, we introduce new game types and improve the object oriented programming experience. A self-guided experiment was conducted with participants across a wide age group and varying levels of video game experience and programming skills. The post survey shows an improvement in level of engagement and usability.

6.1 Limitations

It's important to note some limitations during the experiments in phase 2. Due to the 'work/learn from home' circumstances, we were not able to reach participants from our

primary target: secondary school classrooms. Since sCool is designed for teaching introductory programming, we would ideally like to reach classrooms with younger students who have little to no coding experience. This would provide results more comparable to the experiments in phase 1.

6.2 Future Work

Further evaluation of the game types in phase 2 of sCool will provide a more comprehensive analysis of the game's engagement, usability, and effectiveness as a learning tool. The circumstances in which sCool was evaluated for phase 1 and phase 2 were very different: the first in a classroom setting with teachers and students ages 12-13, while the second remote and self-guided, with participants of many age groups. To accurately assess sCool's improvements from phase 1 to phase 2, its necessary to evaluate phase 2 by conducting experiments in a similar setting as phase 1 (in a classroom, with young students and teachers). Furthermore, currently sCool supports the teaching of introductory object oriented programming concepts, such as instantiating objects and accessing their properties and methods. This can be expanded to support learning other concepts such as creating classes and inheritance.

REFERENCES

- [1] Y. Eshet, "Digital Literacy: A Conceptual Framework for Survival Skills in the Digital era," *J. Educ. Multimed. Hypermedia*, vol. 13, no. 1, pp. 93–106, Jan. 2004.
- [2] M. Prensky, *Digital game-based learning*. New York: McGraw-Hill, 2001.
- [3] K. Kiili, "Digital game-based learning: Towards an experiential gaming model," *Internet High. Educ.*, vol. 8, no. 1, pp. 13–24, Jan. 2005, doi: 10.1016/j.iheduc.2004.12.001.
- [4] C. Perotta, G. Featherstone, H. Aston, and E. Houghton, "Game-based learning: latest evidence and future directions," p. 49.
- [5] A. Hilliard and H. F. Kargbo, "Educationally Game-Based Learning Encourages Learners to Be Actively Engaged in Their Own Learning," *Int. J. Educ. Pract.*, vol. 5, no. 4, pp. 45–60, 2017.
- [6] J. Hamari, D. J. Shernoff, E. Rowe, B. Coller, J. Asbell-Clarke, and T. Edwards, "Challenging games help students learn: An empirical study on engagement, flow and immersion in game-based learning," *Comput. Hum. Behav.*, vol. 54, pp. 170–179, Jan. 2016, doi: 10.1016/j.chb.2015.07.045.
- [7] A. Kojić, "Design and Implementation of an Adaptive Multidisciplinary Educational Mobile Game," p. 174, 2017.
- [8] M. Kojić, "Procedural Content Generation in a Multidisciplinary Educational Mobile Game," p. 158, 2017.
- [9] A. Steinmaurer, J. Pirker, and C. Gütl, "sCool – Game-Based Learning in Computer Science Class: A Case Study in Secondary Education," *Int. J. Eng. Pedagogy IJEP*, vol.

- 9, no. 2, p. 35, Apr. 2019, doi: 10.3991/ijep.v9i2.9942.
- [10] J. Rieman, "A field study of exploratory learning strategies," *ACM Trans. Comput.-Hum. Interact. TOCHI*, vol. 3, no. 3, pp. 189–218, Sep. 1996, doi: 10.1145/234526.234527.
- [11] S. de Freitas and T. Neumann, "The use of 'exploratory learning' for supporting immersive learning in virtual environments," *Comput. Educ.*, vol. 52, no. 2, pp. 343–352, Feb. 2009, doi: 10.1016/j.compedu.2008.09.010.
- [12] D. Kolb, *Experiential Learning: Experience As The Source Of Learning And Development*, vol. 1. 1984.
- [13] S. de Freitas and T. Neumann, "The use of 'exploratory learning' for supporting immersive learning in virtual environments," *Comput. Educ.*, vol. 52, no. 2, pp. 343–352, Feb. 2009, doi: 10.1016/j.compedu.2008.09.010.
- [14] S. de Freitas and M. Oliver, "How can exploratory learning with games and simulations within the curriculum be most effectively evaluated?," *Comput. Educ.*, vol. 46, no. 3, pp. 249–264, 2006.
- [15] G. Falloon, "Using simulations to teach young students science concepts: An Experiential Learning theoretical analysis," *Comput. Educ.*, vol. 135, pp. 138–159, Jul. 2019, doi: 10.1016/j.compedu.2019.03.001.
- [16] L. Yan, "Teaching Object-Oriented Programming with Games," in *Proceedings of the 2009 Sixth International Conference on Information Technology: New Generations*, USA, Apr. 2009, pp. 969–974, doi: 10.1109/ITNG.2009.13.
- [17] R. Hunicke, M. LeBlanc, and R. Zubek, "MDA: A Formal Approach to Game Design

- and Game Research,” p. 5.
- [18] M. Bonaiuto *et al.*, “Optimal Experience and Personal Growth: Flow and the Consolidation of Place Identity,” *Front. Psychol.*, vol. 7, Nov. 2016, doi: 10.3389/fpsyg.2016.01654.
 - [19] M. Csikszentmihalyi, “Motivation and creativity: Toward a synthesis of structural and energistic approaches to cognition,” *New Ideas Psychol.*, vol. 6, no. 2, pp. 159–176, Jan. 1988, doi: 10.1016/0732-118X(88)90001-3.
 - [20] K. Kiili, S. de Freitas, S. Arnab, and T. Lainema, “The Design Principles for Flow Experience in Educational Games,” *Procedia Comput. Sci.*, vol. 15, pp. 78–91, Jan. 2012, doi: 10.1016/j.procs.2012.10.060.
 - [21] D. Dixon, “Player Types and Gamification,” p. 4.
 - [22] L. S. Ferro, S. P. Walz, and S. Greuter, “Towards personalised, gamified systems: an investigation into game design, personality and player typologies,” in *Proceedings of The 9th Australasian Conference on Interactive Entertainment: Matters of Life and Death*, New York, NY, USA, Sep. 2013, pp. 1–6, doi: 10.1145/2513002.2513024.
 - [23] “Richard A. Bartle: Players Who Suit MUDs.” <https://mud.co.uk/richard/hcds.htm> (accessed Nov. 06, 2020).
 - [24] A. Marczewski, “User Types,” in *Even Ninja Monkeys Like to Play: Gamification, Game Thinking and Motivational Design*, 1 edition., CreateSpace Independent Publishing Platform, 2015, pp. 65–80.
 - [25] “Scratch - Imagine, Program, Share.” <https://scratch.mit.edu/> (accessed Nov. 29, 2020).

- [26] “Blockly,” *Google Developers*. <https://developers.google.com/blockly> (accessed Nov. 29, 2020).
- [27] “Learn today, build a brighter tomorrow,” *Code.org*. <https://code.org/> (accessed Nov. 29, 2020).
- [28] “CodeCombat - Coding games to learn Python and JavaScript.”
<https://codecombat.com/> (accessed Nov. 29, 2020).
- [29] “Ozaria - Computer science that captivates.” <https://www.ozaria.com/> (accessed Nov. 29, 2020).
- [30] A. Kojic, M. Kojic, J. Pirker, M. Mentzelopoulos, D. Economou, and C. Gütl, “sCool - A mobile flexible learning environment,” *ILRN 2018 Mont.*, doi: 10.3217/978-3-85125-609-3-11.
- [31] C. K. Mosquera, A. Steinmaurer, C. Eckhardt, and C. Guetl, “Immersively Learning Object Oriented Programming Concepts With sCool,” in *2020 6th International Conference of the Immersive Learning Research Network (iLRN)*, Jun. 2020, pp. 124–131, doi: 10.23919/iLRN47897.2020.9155144.
- [32] “Low Poly Pack | 3D Environments | Unity Asset Store.”
<https://assetstore.unity.com/packages/3d/environments/low-poly-pack-94605>
(accessed Apr. 12, 2020).
- [33] “Kenney • Digital Audio.” <https://www.kenney.nl/assets/digital-audio> (accessed Dec. 01, 2020).
- [34] “Flaticon, the largest database of free vector icons,” *Flaticon*.
<https://www.flaticon.com/> (accessed Dec. 01, 2020).

- [35] J. H. Brockmyer, C. M. Fox, K. A. Curtiss, E. McBroom, K. M. Burkhart, and J. N. Pidruzny, "The development of the Game Engagement Questionnaire: A measure of engagement in video game-playing," *J. Exp. Soc. Psychol.*, vol. 45, no. 4, pp. 624–634, Jul. 2009, doi: 10.1016/j.jesp.2009.02.016.
- [36] "sCool Phase 2 - Questionnaire (part 2)," *Google Docs*.
https://docs.google.com/forms/d/e/1FAIpQLSet0kEhxEjfkPvE7iBmolCWhBML-vdzFkSEJZQKRU6IShB4AQ/viewform?usp=send_form&usp=embed_facebook (accessed Nov. 27, 2020).

APPENDIX

Python code that defines the classes used in phase 2 of the project.

```
import clr
import System
import UnityEngine
from UnityEngine import *

class Position(object):

    def __init__(self, x, y, model):
        self.x = x
        self.y = y
        self.model = model

    def apply(self):
        if hasattr(self, 'model'):
            self.model.transform.localPosition = Vector3(self.x, self.y, 0)

    def __repr__(self):
        return '(' + str(self.x) + ',' + str(self.y) + ')'

    def __str__(self):
        return '(' + str(self.x) + ',' + str(self.y) + ')'

    @property
    def x(self):
        return self._x

    @x.setter
    def x(self, new_val):
        self._x = new_val
        self.apply()

    @property
    def y(self):
        return self._y

    @y.setter
    def y(self, new_val):
        self._y = new_val
        self.apply()

class Scale(object):

    def __init__(self, x, y, model):
        self.x = x
        self.y = y
        self.model = model

    def apply(self):
        if hasattr(self, 'model'):
            self.model.transform.localScale = Vector3(self.x, self.y, 1)

    def __repr__(self):
```



```

        return '(' + str(self.x) + ',' + str(self.y) + ')'
    def __str__(self):
        return '(' + str(self.x) + ',' + str(self.y) + ')'
    @property
    def x(self):
        return self._x
    @x.setter
    def x(self, new_val):
        self._x = new_val
        self.apply()
    @property
    def y(self):
        return self._y
    @y.setter
    def y(self, new_val):
        self._y = new_val
        self.apply()

class Color():
    White = UnityEngine.Color(1,1,1)
    Red = UnityEngine.Color(1,0,0)
    Orange = UnityEngine.Color(1,0.5,0)
    Yellow = UnityEngine.Color(1,1,0)
    Green = UnityEngine.Color(0,1,0)
    Blue = UnityEngine.Color(0,0,1)
    Purple = UnityEngine.Color(0.5,0,1)
    Black = UnityEngine.Color(0,0,0)
    Brown = UnityEngine.Color(0.4,0.2,0)
    Grey = UnityEngine.Color(0.5,0.5,0.5)

class Block(object):
    def __init__(self):
        if not hasattr(self, 'model'):
            self.model = None
        if not hasattr(self, 'name'):
            self.name = 'Block'
        self.position = (1,0)
        self.scale = (1,1)
        self.rotation = 0;
        self.color = Color.Grey

    @property
    def color(self):
        return self._color
    @color.setter
    def color(self, new_col):
        self._color = new_col

```

```

        if self.model is not None:
            blueprintSystem.SetColor(self._color, self.model)
    @property
    def position(self):
        return self._position
    @position.setter
    def position(self, new_val):
        self._position = Position(new_val[0], new_val[1], self.model);
        if self.model is not None:
            self._position.apply()
    @property
    def scale(self):
        return self._scale
    @scale.setter
    def scale(self, new_val):
        self._scale = Scale(new_val[0], new_val[1], self.model);
        if self.model is not None:
            self._scale.apply()
    @property
    def rotation(self):
        return self._rotate
    @rotation.setter
    def rotation(self, new_val):
        self._rotate = new_val
        if self.model is not None:
            self.model.transform.localRotation = Quaternion.Euler(0,0, self._rotate)

class Cube(Block):
    def __init__(self):
        self.model = blueprintSystem.CreateBlock(1)
        self.name = 'cube'
        super(Cube, self).__init__()

class Sphere(Block):
    def __init__(self):
        self.model = blueprintSystem.CreateBlock(2)
        self.name = 'sphere'
        super(Sphere, self).__init__()

class Cylinder(Block):
    def __init__(self):
        self.model = blueprintSystem.CreateBlock(3)
        self.name = 'cylinder'
        super(Cylinder, self).__init__()

class Cone(Block):
    def __init__(self):
        self.model = blueprintSystem.CreateBlock(4)

```

```
self.name = 'cone'  
super(Cone, self).__init__()
```