2-14-2014

Lesson Plan and Workbook for Introducing Python Game Programming to Support the Advancing Out-of-School Learning in Mathematics and Engineering (AOLME) Project

Cherish Franco

Cherish A. Franco

Candidate

Electrical and Computer Engineering

Department

This thesis is approved, and it is acceptable in quality and form for publication:

Approved by the Thesis Committee:

Dr. Marios Pattichis, Chairperson

Dr. Ramiro Jordan

Dr. Sylvia Celedón-Pattichis

Dr. Carlos LopezLeiva

Lesson Plan and Workbook for Introducing Python Game Programming to Support the Advancing Out-of-School Learning in Mathematics and Engineering (AOLME) Project

By

Cherish A. Franco

B.S., Computer Science, New Mexico Institute of Mining and Technology, 2010

THESIS

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Science

Computer Engineering

The University of New Mexico
Albuquerque, New Mexico

December 2013

DEDICATION

All my work, both past and present, will forever and always be dedicated to my father, mother, and brother. Without their love and support, I wouldn't be where I am today. Also, a special thanks to Jason R., for believing in me and loving me.

ACKNOWLEDGMENTS

I would like to thank Dr. Lorie Liebrock of New Mexico Tech, whose compassion, understanding, guidance, and tireless efforts to keep me on track and advise me every chance she had, helped me to achieve my Bachelor's degree through much trial and tribulation. I would also like to thank Dr. Nasir Ghani for his encouragement when I was new to the program. His quick responses and well-advised guidance helped me to pursue what was best for me, and helped me to find my passion for teaching. Lastly, I would like to thank Dr. Marios Pattichis for his exemplary efforts to help me find the topic that worked best for me, and whose infectious excitement and attitude towards visual learning for children showed me that my passion for teaching and learning is something worth fighting for.

Lesson Plan and Workbook for Introducing Python Game Programming to Support the Advancing Out-of-School Learning in Mathematics and Engineering (AOLME) Project

By

Cherish A. Franco

B.S., Computer Science, New Mexico Institute of Mining and Technology, 2010

ABSTRACT

In recent years, research has started to show a distressing trend in the world of science, technology, engineering, and mathematic (STEM) fields. It has become apparent that students from under-represented groups get little to no experience or practice in the field of engineering. This apparent lack of exposure to engineering knowledge and practice is likely to be the cause as to why students from under-represented groups may not become interested in engineering careers.

The Advancing Out-of-School Learning in Mathematics and Engineering (AOLME) project was created specifically for providing integrated mathematics and engineering experiences to middle-school students from under-represented groups. The thesis presents a new approach to introducing game programming to middle-school students that have undergone AOLME-training while still maintaining a fun and relaxed environment. The thesis provides a discussion of three different educational, visual-programming environments that are also designed for younger programmers and provides motivation for the proposed approach based on Python.

The thesis details interactive activities that are intended for supporting the students to develop their own games in Python.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

1.1 Motivation

As children, we are often taught new concepts and ideas through the use of interactions and visualizations. Children from preschool through elementary school age are taught topics such as their ABCs, shapes, and colors by using images. This is predominantly done to reinforce the particular concept they are trying to learn. In my opinion however, as the age of the child, and the child's education increases, we begin to see that the use of visualizations is not so prevalent. Because of this lack of exposure to engineering experiences and practices through the use of these visualizations, it is possible that students become less inclined to pursue any sort of scientific or engineering-based subject for their educational future. Students are generally unprepared when it comes to engineering because they are required to learn difficult subjects in a setting that is not geared towards their learning needs at such a young age. A possible approach to address this dilemma was initiated in the Advancing Out-of-School Learning in Mathematics and Engineering (AOLME) project.

AOLME is an interdisciplinary project run by two distinct departments. The Electrical and Computer Engineering Department from the School of Engineering, and the Department of Language, Literacy, and Sociocultural Studies from the College of Education have worked together to bring AOLME to fruition. The

project's primary audience is middle school-aged children. The goal of AOLME is to promote awareness of both a set of mathematical practices different from, but still related to those experienced at school, and a set of meaningful experiences in engineering through the processing of digital images and video [1].

The basis for this thesis revolves around this idea of reaching out to students who have not been exposed to the elements of engineering and mathematics by teaching them using non-traditional means. Non-traditional means in this case would be either verbal or written instruction.

As a naive college freshman, I ventured into the world of computer science without any prior knowledge, except for a very basic programming class on Visual Basic. I had no idea what Linux was, no idea what object-oriented meant, and certainly had no clue as to how to program using a terminal. Because of this struggle, I began to develop an idea on how I could prepare students for what was to come, far in advance of the start of their college career. I felt that in order to stimulate more interest in engineering and computer science, I needed to be able to effectively teach younger aged students the basics of programming so that they could be far better prepared for more complex topics.

I learned very quickly that I am a visual learner. If I am given some code, and I can watch it run, I quickly learn how to make things work and make them better. So, I decided that in order to capture the minds and imaginations of younger students, while at the same time guaranteeing that they will be well prepared when it comes to programming, why not create a means for them to learn to program while still

having fun? This idea could be simply solved by introducing the concept of games and learning how to program them.

Programming games is by no means a new art form. Essentially every single video game created is made possible by programming. AOLME already has ventured towards the idea of having its students create animated videos with pixels, but introducing games would be a whole new area of interest for them.

In summary, the motivation behind this thesis is address the general lack of motivation and interest in engineering of middle school students by implementing a teaching design and workbook that instructs those students and retains their interest and attention.

1.2 Thesis Statement, Innovations, and Contributions

There are hundreds of different programming languages and applications in use in today's technologically advanced world. These programming languages, however, are created with the assumption that the potential user has some sort of basic knowledge when it comes to programming. These widely used programming languages are primarily geared towards young adults and beyond and rarely take into consideration a much younger audience. Recently, however, there has been a bit of a push in the direction of developing programming languages and environments that can be used to teach young children the intricate art of programming. Instead of exposing students to the underlying programming languages, most of these efforts provide programming environments that are centered on one key aspect, the use of interactive visualizations.

Educational programming environments are tools that try to approach instruction through the use of these interactive visualizations. These programs were largely created with the idea that they would help the student to understand the logistics of programming by using interactive visuals, instead of the more common practice of verbal and written instruction. Some environments that are particularly suited towards younger children incorporate more visual aids as well as much simpler explanations. Although most of these environments are not meant for the creation of real-world applications, they do allow the user to create programs that follow the same flow and syntax of an actual program.

The goal of this thesis is to introduce a Python-based framework for extending the current AOLME activities into game programming. This goal can be achieved by using my proposed three-step teaching process based on providing students coding examples and thorough line-by-line explanations. Another goal of this thesis is to explore a means to teach game programming in Python, by incorporating visual stimuli in the form of creating characters all the way to creating interactive games. This thesis will explore a way to develop a visual guide to teach the topic of game programming in Python, while maintaining an understanding that those students participating in AOLME, will have little to no experience with programming.

In order to teach students how to actually program, we must first be able to engage them through the use of visuals and interactions, while at the same time introduce programming using languages such as C or Python. This thesis will introduce a few key innovations and contributions, as listed below:

1. **Develop a 3-step approach on how to teach programming. The 3 steps are as follows:**
    a. ***Provide students with working code*** – This step allows the student to see the code in action without any errors, as well as give them a template of what their code may look like as they develop their own game. This template will aid them in finding errors when they began to write more complex programs. This step avoids having students type significant amount of code or having to work through syntax errors as outlined in [16].
    b. ***Provide students with line-by-line explanations*** – This step allows the instructor the ability to explain each and every parameter and function of a line of code, and gives the student an inside look as to how the code works as a whole. These explanations will also aide in enhancing the students' programming skills by enforcing good programming practices. This step will be compatible with AOLME practice of explaining the code that is being used in the tutorials and then having the students fill-in their own code.
    c. ***Provide and run programs with errors or missing code*** – This step is the key to the 3-step approach. The instructor has provided a working template, and explained what each line is doing at each step, but now we introduce errors. This step is here to ensure that the students understand that no error is a bad error. There is a lesson to be learned from each and every mistake. Some errors will need to be

fixed, but other so-called errors may actually produce an unexpected surprise when the program is run. Students may find that they discover some unintentional action that wasn't previously provided to them. They will also have to fill-in missing code based on material taught to them during previous activities. This template approach has also been tested and proven to be effective in AOLME where the students are asked to develop their own video representations based on the examples that are given [16].

2. **Develop a workbook that is based on the 3-step approach discussed above. This workbook will:**
    a. Introduce programming in Python;
    b. Use the 3-step approach in each lesson;
    c. Engage the students by teaching them how to program a very simple game, and
    d. Leave enough to the imagination to make the students want to learn more once they see how easy and fun programming can be.
3. **Develop an answer manual that aides the instructor.**

    Sample answers will be given, time amounts for each lesson, as well as specific instructions on how the students might find the answers they need in order to answer the question in the book.

1.3 Thesis Summary

The remainder of this thesis is organized in the following fashion:

1. **Chapter 2: Background**

   This chapter will provide background information on 3 widely used visual programming environments--Daisy the Dinosaur, Alice, and Scratch. These three languages were chosen specifically because of their unique approaches to introducing programming to a younger audience. Topics discussed for each programming environment will include an overview of the interface of the programming environment, a sample, block-based program, as well as a discussion on how the environments were used, and whether students are retaining useful programming skills. The discussion on these three programming environments will then be followed by a discussion on Python, its approach to game programming, and its use throughout the activity workbook created for this thesis.

2. **Chapter 3: Middle School Lesson Plans for Teaching Game Programming Using Python**

   In this chapter we will extensively discuss the 3-step approach highlighted in Section 1.2 above. Individual lessons will also be addressed in order to show how this 3-step approach is being utilized throughout the workbook. There will also be a discussion on how the workbook was developed, an explanation on how each lesson was created, as well as a discussion on why this particular game was chosen as an introductory programming lesson.

3. **Chapter 4: Future Work and Conclusions**

In this chapter we will discuss future projects, including a discussion on introducing 3D environment, as well as conclusions on the practicality of the Python workbook and 3-step approach.

4. **Appendix A: The Paddle Battle! Workbook**

   Appendix A will provide the actual workbook in its entirety. Lesson plans, questions, and code are shown throughout the workbook. Some lesson plans are step-by-step instructions on how to program the particular piece of code the student is being taught, and other lessons rely solely on the knowledge that the student has gained from previous lessons. At the end of the workbook, the students will have enough code to see results, but will be expected to finish their game on their own. Time limits on lessons will range from 15 minutes to 60 minutes. Students are encouraged to ask questions and make mistakes.

5. **Appendix B: The Paddle Battle! Instructor's Manual**

   Appendix B provides all the answers for the activity workbook. Instructors will use this manual as a guide for students and the questions they may have. Sample answers are provided, and all code is provided for each lesson. Lastly, all code needed to answer the questions at the end of the workbook are provided.

Chapter 2

Background

2.1 Existing Visual Programming Environments

As discussed earlier, there are many educational programming languages being used in classrooms all over the world. In the following sections, three educational programming environments will be discussed. The first environment, an app called Daisy the Dinosaur, is created for children ages 5 and up. The second and third programming environments, Scratch and Alice, are more well-known. These three environments were chosen because of their ingenuity and creativeness. Each section will give an overview, a sample program, as well as a discussion on how they were used, and whether students are actively learning how to program by using these languages. This discussion will then be followed by a proposal on how AOLME might approach this same concept, while still maintaining their unique goals of bringing awareness to mathematics and engineering, and using Python. The proposal will contain a discussion on three key elements for this teaching approach, as well as a discussion on the creation and use of the activity workbook created for this thesis.

2.2 Daisy the Dinosaur (ages 5-8)

Daisy the Dinosaur is a visual programming environment that is geared towards a much younger audience. Students as young as 5 years of age are able to use this application to get a basic introduction into the world of programming. Daisy the Dinosaur is currently only supported on the iPad and iPhone platforms, so it unfortunately only reaches those students who happen to have access to that

particular technology. Nevertheless, the concept of Daisy the Dinosaur is a rather interesting and engaging approach.

2.2.1 The Syntax for Daisy the Dinosaur

The basic idea for programming in Daisy the Dinosaur revolves around making Daisy do whatever the student has created in their program window. Students are given access to a finite set of commands that they drag and drop to their program window, as shown below in

Figure 1. Once they have created their program, they can then press play to see what they have made Daisy do.

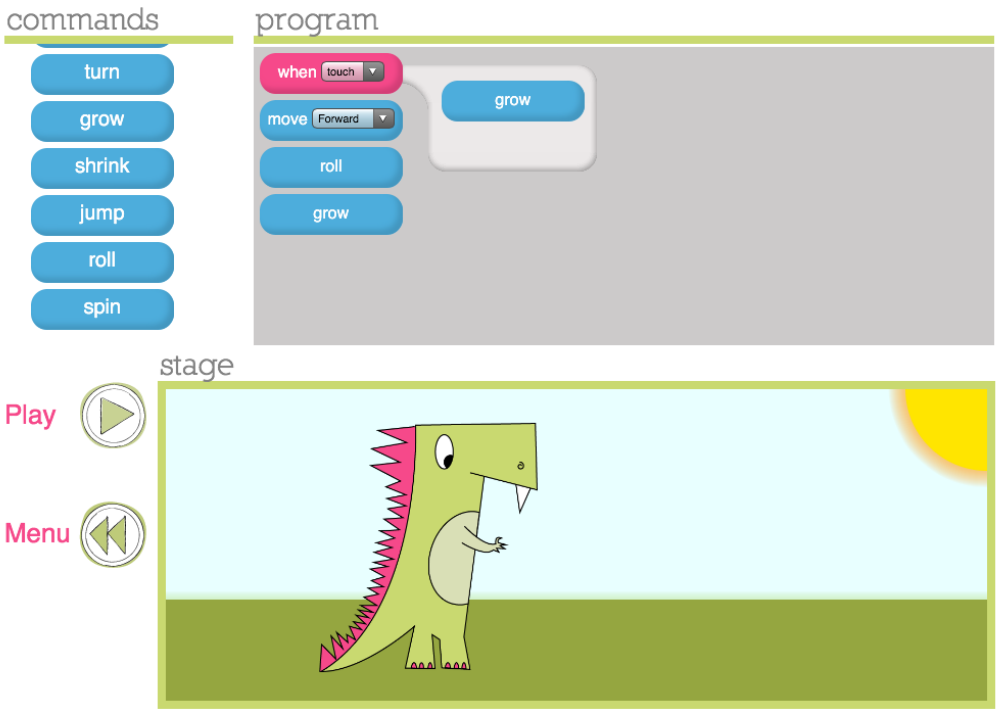


Figure 1: Example Program Using the Daisy the Dinosaur Application [2].

There are a few tutorials in the application; however, the entirety of the Daisy the Dinosaur application is separated into two simple categories: Freeplay

Mode and Challenge Mode. Freeplay Mode allows students to interact with the application and create their own programs to see what happens. As stated before, there is a finite set of commands that the students have access to, such as move or grow. In Challenge Mode, students are asked to write simple programs in order to pass the challenge. The first challenge is the easiest of all the challenges; however, as the student progresses through each challenge, the difficulty increases. To make the challenges a little easier to complete, only those commands that are needed for their program are shown in the commands window. Unfortunately, there are only 5 challenges at this point in time. It is my opinion that more challenges that created much more intricate programs to attempt would be very beneficial to a beginner.

Essentially, Daisy the Dinosaur programming is as simple as it can get, and yet, introduces more complex concepts such as loops. The `when` command in Daisy the Dinosaur is basically a simple loop command. Students can learn how to make certain actions happen at a set number of times that they determined. Being able to introduce a concept such as loops to kindergarten aged students is phenomenal, but do the students actually acquire sufficient skills that relate how to program real world applications?

2.2.2 Daisy the Dinosaur as an Introductory Environment

Daisy the Dinosaur introduces students to computer programming by engaging them with cute graphics and easy to learn syntax. Students not only learn basic computer programming skills, but also build their problem-solving and analytical skills, something they will need in their future educational careers [3].

Unfortunately, Daisy the Dinosaurs' layout and content is not enough to prepare students for more complex programming languages. There are no published studies that describe whether students actually learned any skills of value from using this application. However, a quick overview of the program has convinced me that there is a limited amount of information that students will learn. The process of programming, and the beginnings of conditional programming are addressed, but the lessons learned using this application may not be very useful when a student needs to start programming in something like Python for the AOLME project. If anything, Daisy the Dinosaur is a fun application that can be used to pique the interest of the students and get them to start exploring the world of programming.

2.3 Alice (ages 8+)

The next educational programming environment we will discuss is called Alice. Alice is an innovative 3D programming environment that makes it easy to create an animation for telling a story using the idea of programming. It allows students to learn fundamental programming concepts in the context of creating animated movies and games [4].

2.3.1 The Syntax for Alice

One of the key things that Alice does for its students is letting them see almost immediately what their programs are doing when they run them. The highly interactive and eye-catching 3D graphics that are readily available for the students to use opens up a whole new way to learn. Students can play around with their programs and can create their own environments and characters with very little

programming knowledge. For example, in Figure 2 below, we see a program where the student has made an ice skater that skates and moves around the screen. The steps it took to make this environment a reality are quite simple. The student created the environment, invoked the iceSkater object, the lake, and the ground, and then simply wrote their code using the drag and drop operations provided. Granted, the program described here is incredibly simple. As the student begins to develop more complex environments and characters, the code, and the knowledge of how Alice works becomes immensely more complicated.

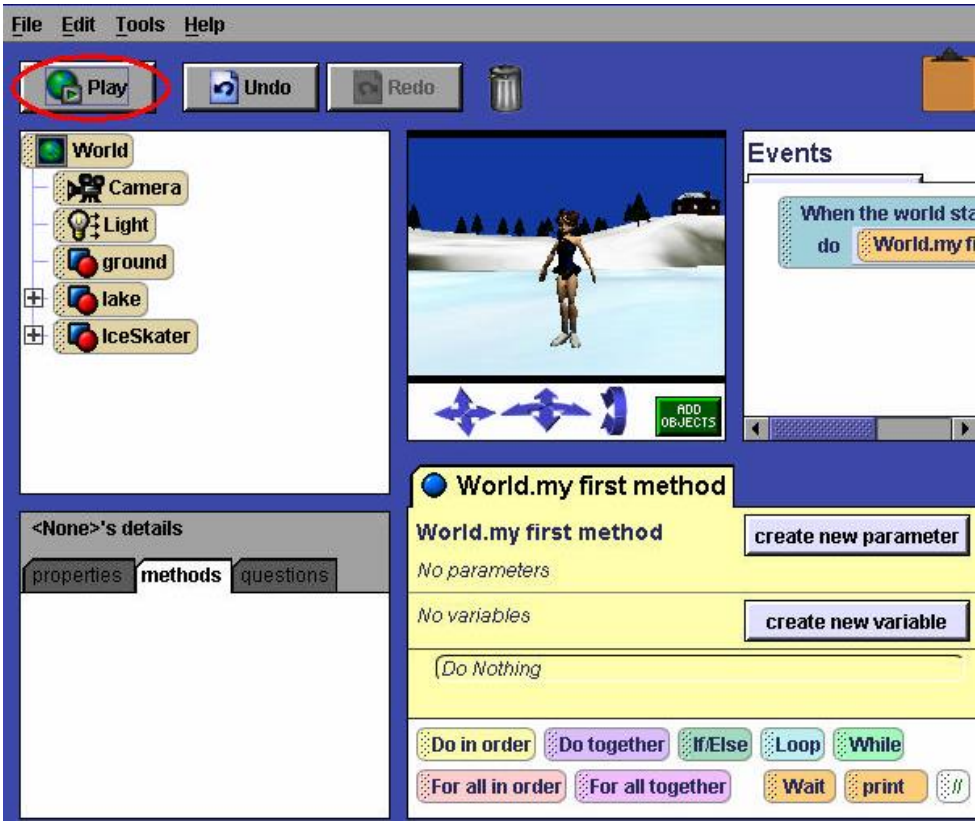


Figure 2: Example Program using Alice [5]

Alice is able to introduce complex programming concepts without over-burdening their students with memorization or requiring that they have prior

programming knowledge. Alice introduces the user to functions, flow control and loops, as well as recursion, which is not an easy topic to introduce. Another great feature of Alice is the fact that the Python language is interwoven into Alice. This opens the door to the beginnings of teaching younger students more complex and real-world programming languages.

2.3.2 Alice as an Introductory Environment

In a published study authored by a group from multiple universities, they found that although there are many benefits to using Alice, there are still many issues and problems that arise from using it [6]. One of the main issues they found was the fact that errors in Alice were either incredibly cryptic, or no error or warning was displayed. For example, if a user wrote a program to move their object 5 spaces, and then back 5 spaces, they might try something like the following:

```
someObject.Move(Forward, 5)
someObject.Move(Back, 5)
```

Unfortunately, this will produce no movement at all, primarily because Alice, by default, causes all animations to occur simultaneously [6]. Although this is not necessarily an error, it might have been beneficial to throw a warning telling the student that since they did not include a control structure, then their animation may not work as they expected.

Another issue that can occur is the fact that in order to properly use Alice, the student must have a complete understanding of how Alice statements work, as well as how Python works [6]. Students may run into the issue of non-functioning

programs purely because they were unable to distinguish between Alice statements and Python statements. If a student accidently puts a Python statement within an Alice statement, their function will fail because Alice will not recognize the Python code. In order for the program to run correctly, the student would have to move their Python code outside of the Alice code. This would seem to indicate that a student must understand two languages, Alice and Python, before they could even begin to program. Understanding two different languages is not a bad things, however, asking that a middle-school aged student to learn and memorize two languages may be pushing their limits a bit much.

After viewing the Alice environment, it becomes apparent that the particular learning style most predominantly used is interactive visualizations. The Alice developers wanted the users to be able to have fun while at the same time learn programming. In my opinion, interactive learning is very important when trying to teach a difficult problem. By engaging the user, the instructor can delve farther into more complex topics. Alice does a wonderful job of simplifying the topic of programming and allows the user to learn at a comfortable pace through the power of inquiry and creativity. However, learning Alice is not quite that simple and requires a little more background in programming than Daisy the Dinosaur. Alice is another programming environment that has the ability to increase more interest in programming by engaging their audience with graphics and games.

2.4 Scratch (ages 8+)

One of the most highly used early-education programming language environments for children is a programming language called Scratch. Scratch is a type of educational program that makes it easy to create interactive stories, animations, games, music, and art [7]. It was developed in 2003 by the Lifelong Kindergarten group, led by Mitchell Resnick at the MIT Media Lab [7].

2.4.1 The Syntax for Scratch

The Scratch syntax is based on a collection of graphical "programming blocks" children piece together to create programs. The blocks are shaped to fit together only in ways that make syntactical sense [7]. In other words, if a block does not fit another block, then the flow of the program does not make sense and will therefore not run as the user expects. The blocks have many shapes, sizes, and functions. They also are only pieced together in an obvious way based on these shapes and sizes. This unique visual programming method limits the amount of syntax and grammar of the language [7]. In Figure 3 below, we see a simple Hello, World example being created using the distinctive Scratch syntax.

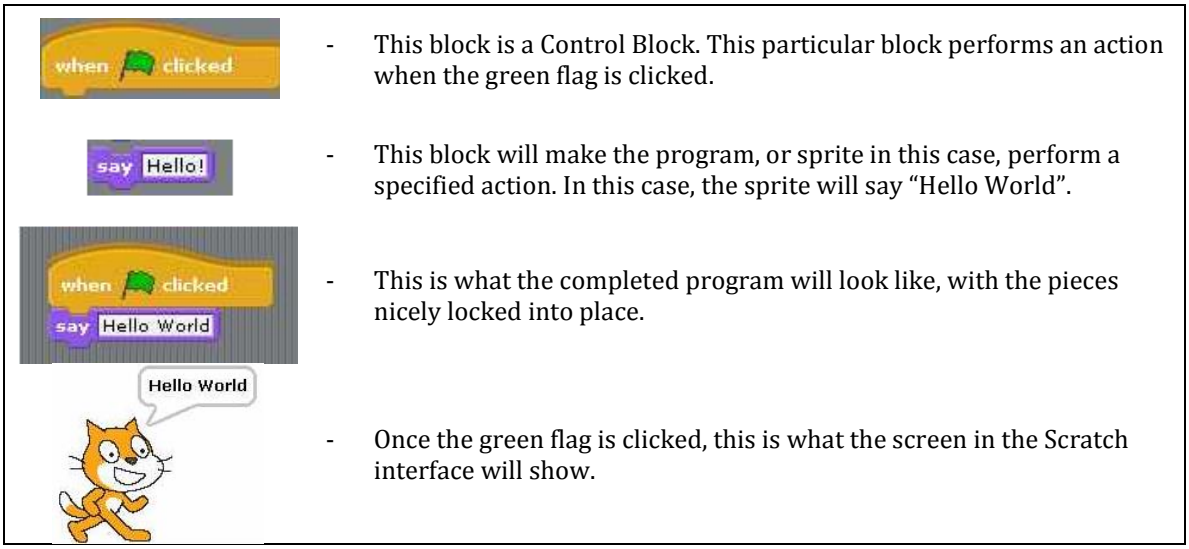


Figure 3: Scratch Syntax [8]

This style of programming is very beneficial to the users. It allows them to experience the process of programming, but eliminates the need for memorization of syntax, rules, and formatting.

2.4.2 Scratch as an Introductory Environment

Outreach programs, such as AOLME, are designed to attract and increase the amount of students in the science and engineering disciplines. In order to do so, these programs must come up with ways to engage their students by using tools, such as Scratch, to stimulate more interest in subjects that are not always taught well. One example of an outreach program that used Scratch as its base programming language is a two-week camp known as Animal Tlatoque [9]. This particular camp was created not only to increase interest, but also to research whether or not using tools such as Scratch were at all beneficial to learning how to actually program.

Animal Tlatoque was initially designed with two specific goals: (1) to attract a target audience of middle school students from underrepresented groups with non-CS backgrounds that appeal to both parents and children, and (2) to engage participants in interdisciplinary activities that allow them to learn about computer science and develop skills for computational thinking [9]. They also took on an added goal of assessing the amount of information retained of any actual CS content that took place during the camp. They placed some constraints on the camp that determined that it would be a two-week, non-academic, interdisciplinary, "fun-oriented" camp [9].There were lessons for each of the two weeks of the camp. The first sets of lessons for week 1 were setup to only introduce those concepts that the students would need to complete a final project. The set of lessons for week 2 were primarily to reinforce topics such as conditionals and events, but were not necessarily needed for the final project. By the end of the camp, students were tested on what they had learned.

What this group found was that even with those constraints listed above, the students were more or less able to comprehend and retain the material that they were given. They found that the student were able to utilize several computer science concepts such as variable usage, changing the sprite, or even manipulating it with ease; however, it did appear that concepts such as if blocks were not well known. The group believed that this was probably due to some extraneous issues such as the students being rushed or being far more interested in other, more fascinating lessons. Though these issues may have been a factor in the students not retaining this particular concept, it still needs to be considered that perhaps the

programming tool itself is not structured well enough to teach a concept such as if blocks effectively. Though no age specifications were given in their final report, the group did indicate that their research was based on those students who were first year students to the camp, thus they were students who had no background in computer science. Nevertheless, the camp was effective in its approach to teach programming concepts while determining whether or not using Scratch was useful. Students did learn some programming concepts that would benefit them in programming with real-world languages such as variable usage, correct implementation of some functions and controls, and correct use of events.

2.5 Python

All three of the programming environments mentioned earlier approach programming in largely different ways; however, none of them focus on the direct use of a general purpose programming language. The three programming environments, though fun and engaging do not sufficiently expose them with the source-code behind the programming concepts. Exposure to the source-code and development of programs at the source-code level is an essential skill for access to computing careers. AOLME's objective is to create a learning environment where students are learning both concepts of mathematics, engineering, and computer science concepts while still having fun. In order to guarantee that a student is actually learning applicable material, a general-purpose programming language, such as one that supports object oriented programming, needs to be utilized. Object oriented languages are powerful languages that are used throughout the world of programming. Some languages are difficult to learn and understand, but then there

are languages like Python, which in my opinion are very powerful and relatively easy to comprehend.

2.5.1 The Syntax for Python

Python's syntax is widely used and easy to understand. Its syntax is based on the idea that it be as readable as possible. Another key aspect of the language is that it boasts the ability to allow programs to be built using fewer lines of code than those of other programming languages [10]. As an example, let's take a look at a simple Python program from the workbook in APPENDIX A: The Paddle Battle! Workbook.

```
1. class Ball:
2.     def __init__(self, canvas, color):
3.         self.canvas = canvas
4.         self.id = canvas.create_oval(10, 10, 25, 25, fill=color)
5.         self.canvas.move(self.id, 250, 250)
6.
7.     def draw(self):
8.         pass
```

Figure 4: Example Code from Workbook

This example shows a class that students will work with in a lesson. The class simply creates a small oval that will later be drawn on the canvas or window the student has created. Though the syntax for Python is immensely more complicated than those of the previous visual programming environments discussed, the Python language itself provides direct access to the practices that computer science and engineering students need to be successful.

Python has many capabilities from being used as a scripting language, to being used in conjunction with other tools to create more complex programs. An example of this is Python's ability to support a module called Pygame. Pygame is specifically written and used to develop games in Python. The use of this module opens the door for much more interactive and visual games than can be produced with the standard packages of Python, and works by introducing a relatively small number of additional programming constructs.

2.5.2 Python as an Introductory Environment

As an introductory programming language, Python seems to work for children trying to learn programming for the first time. The book primarily used for research in this thesis, "Python for Kids: A Playful Introduction to Programming", by Jason R. Briggs [10], is an excellent example of how Python can be taught to even the youngest of students.

The book's aim is to introduce programming by keeping the lessons incredibly short and interesting. Lessons are broken up into short pieces of information the students will need as they progress through the book. Finally, the last few chapters are dedicated to putting together all the knowledge gained and creating a couple of games. These lessons on creating the games are broken up in the same way the book is, and take measures to keep the content low level and fun. Students are provided all the code that they need to create a finished product, and then asked to improve upon it at the end. The students are also allowed to view the answers to the challenge questions, just in case they are having a difficult time,

although Python is still considered an easy enough programming language to be an effective tool for teaching programming.

2.6 A Summary of AOLME

AOLME is a good indicator that teaching introductory programming using Python can be done succesfully. The projects primary goal is to establish a pipeline of support and motivation for underrepresented middle school students to pursue a career in STEM [16]. AOLME has been successful in teaching students programming by using Python and have been able to stimulate more interest in the field of programming by developing a curriculum that jointly approaches a design that would combine mathematical ideas with engineering concepts. The curriculum includes but is not limited to proportional reasoning, geometry, and algebra for the mathematics portion, and digital image and video through computer programming for the engineering concepts. In keeping with this trend, this thesis is a way to further the reach of the students and keep them interested in learning an actual programming language.

2.7 Programming Environments and Languages Summaries

In summary, the table below is provided to showcase each language discussed in Chapter 2. It must be noted that all of these programming environments and languages are unique and useful in their own ways. It is my opinion that each language has its own way to get students thinking about computer programming and in a sense, teaches them the basic concepts that they will need to understand in order to pursue a future in engineering and computer science. The goal of this thesis

however, is to reach even farther than what these environments are aiming for. The goal of this thesis is to teach the concept of programming using a general purpose programming language, not a programming environment. The table provides a summary of each language and what they tried to accomplish, as well as provide examples of students gaining useful programming skills using these languages.

Programming Environment	Summary	Skills Gained?
Daisy the Dinosaur	This is an introductory programming language that teaches students as young as 5 years of age how to program by controlling the character known as Daisy the Dinosaur. Students are limited to just a short list of commands, and the GUI blocks support a rather limited number of specific uses. Students can either create programs on their own, or take the challenges that prepare them for more intricate programs.	No studies have been conducted using this language. The syntax of Daisy is a little too simplified to support teaching more complex programming concepts (e.g., variables). This would best be used as perhaps an example of what a student could make when they learn how to program.
Alice	A 3D environment that allows the student to create environments, characters, movies, and games. This powerful language is very fun and engaging. It has a set of commands to be used, but also allows the students the ability to enhance their programs by changing and modifying those commands.	Studies done on Alice show that for the most part, Alice is an effective tool to teach introductory programming. Colleges and universities currently use Alice in some of their classes; however, it is noted that the syntax and usage of the programming are often times considered difficult to learn right away. Another issue is that error messages are not readily given, thus confusing the student more when something doesn't work right.
Scratch	An interesting and fun programming language that builds its programs through the use of puzzle-like code pieces, the approach to writing programs is fun and interesting. It is powerful and has the ability to create very fun programs; however, its application in the real-world is not incredibly useful. If anything, students learn how the functionality of code works, but not how to actually write their own programs.	Animal Tlatoque found that although the students were limited in what they could make, they were still able to retain a lot of useful information and showed signs of comprehending many difficult computer science topics. There were some topics that were not as easily retained, such as if statements, but overall, this language was able to open the door for the students to begin to comprehend programming.
Python	Python's syntax is much more difficult to learn than the other programming languages; however, its main goal is to be able to write functional, real-world applications with as little lines of code as possible. It requires the use of functions and understanding of the syntax for the ability to create general-purpose applications..	AOLME students have created images and videos in classes so far. Preliminary results indicate that students' interest in programming has grown [16], but more research is needed on ways to support student engagement. The workbook in this thesis is a proposal on how to better engage the students using game programming.

Table 1: Summary of Visual Programming Languages

Chapter 3

Middle School Lesson Plans for Teaching Game Programming Using Python

3.1 Using Python to Create Games

The proposed activities assume that the students have already learned the basics of Python and how to create digital images and videos in the AOLME project. The next logical step to better their programming skills, and pique their interest is to introduce game programming. This thesis is a proposal for how this step might be approached by AOLME. The following sections will highlight three key elements of this educational approach which I developed as part of this project. The three key elements discussed are (a) providing students with working code, (b) giving line-by-line explanations, and (c) having the students run code that has been purposely tampered with. There will be a discussion on all three of these elements and why they were chosen and considered important. There will also be examples provided on particular lessons that students will come across in the proposed activity workbook that is part of this thesis. Finally, there will be a discussion on the layout and content of the activity workbook, future work, and how this proposal might be used by AOLME in the future.

3.2 Providing Students with Working Code

One of the most beneficial ways to help students learn the art of programming is by giving them examples of running code. Although some might consider this a form of

cheating, providing working code to the students helps them to visualize how the code works, and what it should be doing. Students who are being introduced to a new programming language need to first be able to understand those basic concepts of the language that will be used in any program, simple or complicated. Providing working examples of these basic concepts gives the students a piece of working code to fall back on if they find themselves struggling to write a more complicated program in the future.

This kind of approach could be considered a sort of reverse engineering approach to programming. This method allows the students to see and modify existing code, and then watch what happens when those changes are compiled and implemented. By taking this reverse engineering stance, we begin to open up new doors for the students to enter such as seeing what happens when code is changed, learning what errors are caused by changes and how to deal with them, as well as giving them the potential to discover some new concept that might make their programs even better.

This teaching method is what is to be considered the main concept for this proposed approach introduced in this thesis. To better understand this proposed approach, the example below is a possible lesson that a student might be asked to complete using the first step of this 3-step approach.

3.2.1 Lesson Plan Example for Section 3.2

One of the first lessons in the workbook that accompanies this thesis is to have the students run the code for the game they will eventually be creating. This workbook

will provide each student with the complete source code for a Pong-like game. As shown in Figure 5 below, the students are asked to run and watch the code, and then answer a few questions about it.

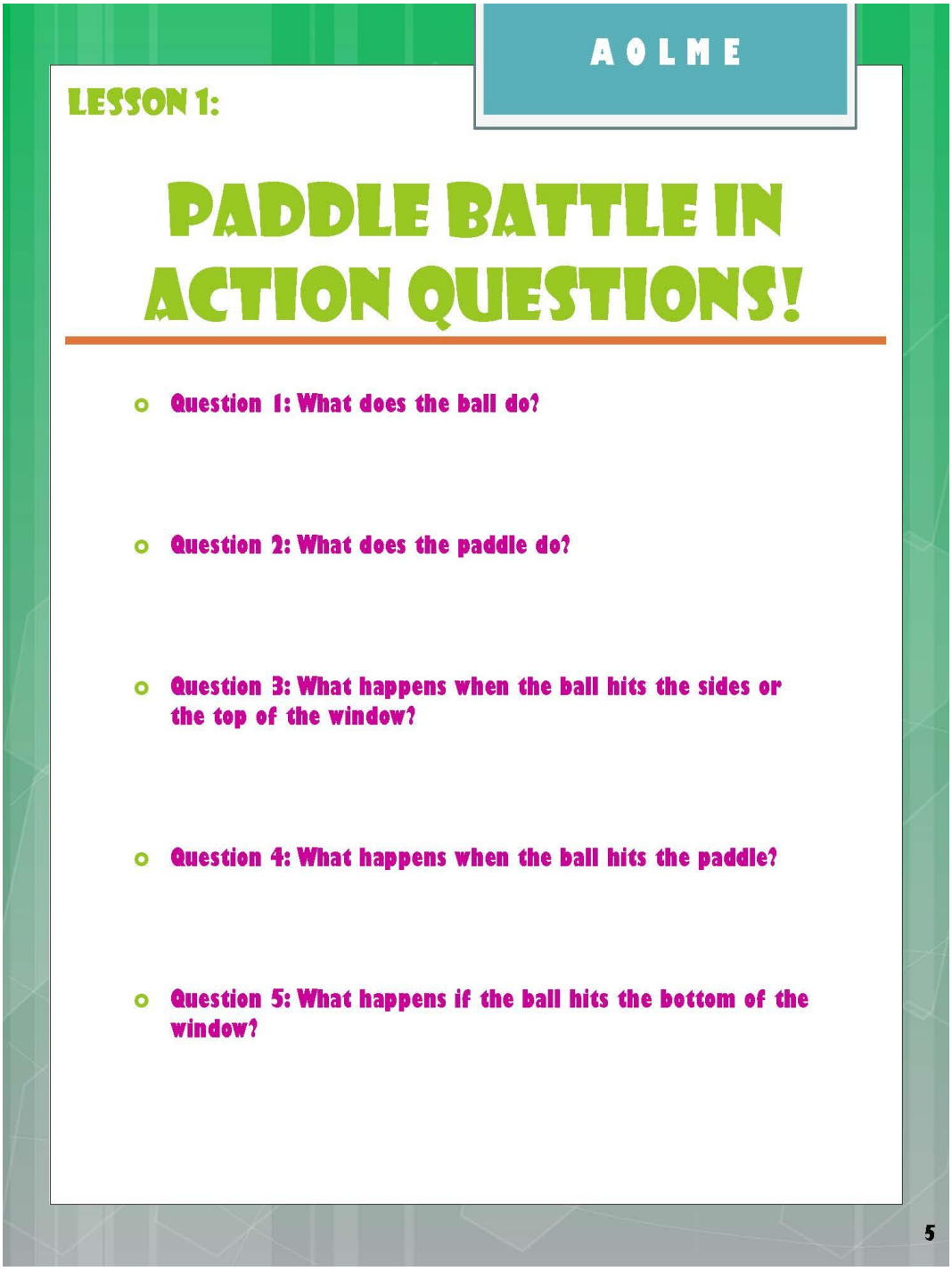


Figure 5: Lesson 1 Questions

The purpose of this lesson is to make sure the students understand how the code works before they start to actually create it themselves. The lessons that follow in the workbook continue with this approach by first having the students run the individual pieces of code they are working with, learning how it works, and finally experimenting with it.

3.3 Providing Students with Line-by-Line Explanations

In my opinion, providing the students with line-by-line explanations is essential to teaching effective programming skills. There is no greater feeling than writing a program and then watching it run. However, before a student can get to this point, the student needs to first understand the flow of the code as it runs. This knowledge will help the student not only to better understand how his or her program works and what it is doing, but also to understand where things may be broken if errors occur.

The purpose of providing a line-by-line explanation of what is happening at each step of the program is to reinforce good programming practices. These practices are strengthened because the student will not only be learning what each line of code is doing, but also be observing how each line impacts the flow of the code as it is run. For example, if we provide the students with a simple "Hello, World" program, and then provide them with line-by-line instructions, we find that we are able to teach at least three different topics using this one simple program. The student will be learning how the canvas works and how to implement it, how

the print functions works, and how we can manipulate the text that is displayed when using the print function.

In order to make sure that the students understand the flow of a program, we must first provide them with an explanation of the program code we already have provided them with in their lesson plans. The code supplied is already correct, so the first step is to show the students what happens when the program is run. This gives the student a visual idea of what the code does. The next step will then be to run this same code, but show the students what happens at each line, even if the explanations seem a little redundant. By explaining the concepts over and over again, we hope to instill these good programming practices for later use.

Once these explanations have been given, and a step-by-step walk through has been completed, the students will then be expected to write their own piece of code, while using the code given previously as their guide. Students will be expected to write their program, and provide their own explanations as to how they think their code should run. The students will then run their own code and figure out whether their explanations and understanding are reflected in the way that their program is running. For a better understanding of this proposed approach, the example below will highlight steps a student might have to take when attempting this lesson.

3.3.1 Lesson Plan Example for Section 3.3

In the workbook created for this thesis, we see that each lesson is broken down into small steps and small pieces of code. Each lesson displays the sample code given to the student for that particular lesson, and then steps line-by-line through the code

with them. In Lesson 3, the students are learning how to create a class called Ball, which will allow them to draw a ball on the window that they just created in Lesson 2. The first step in this lesson is to have them run and watch the sample code to see what happens. Then we begin our explanations as seen in Figure 6 below.

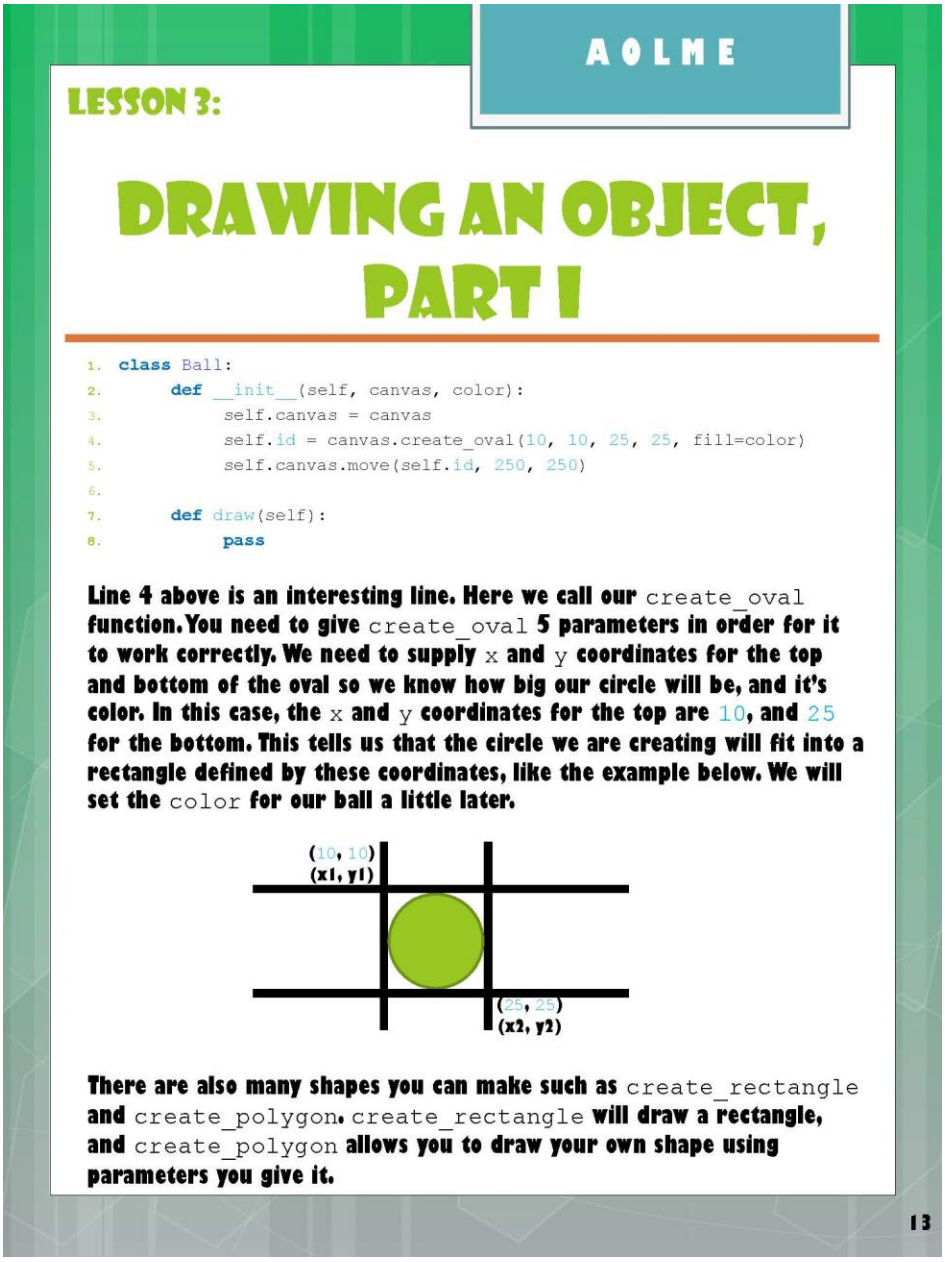


Figure 6: Lesson 3 Example of Line by line Explanations

In the case of Figure 6, we have spent a little bit more time on this particular line's explanation. This is because this line requires a lot of parameter inputs from the students. We want to make sure that the students understand how to use this particular function in this line, and that they know what parameters they will need. We also want to make sure that they know that there are other functions like it, so that the students can explore with the different functions available to them. The introduction of similar functions also makes an appearance in this lesson as a setup for a question that must be answered in the following lesson, Lesson 4.

As we continue to step through the code for Lesson 3, each line is explained as thoroughly as possible. However, the lines of code are not expanded upon. Questions asked at the end of each lesson are designed to help the students further their knowledge on some of these concepts. For example, let's take a look at Questions 2 and 3 of Lesson 3 shown in Figure 7 below.

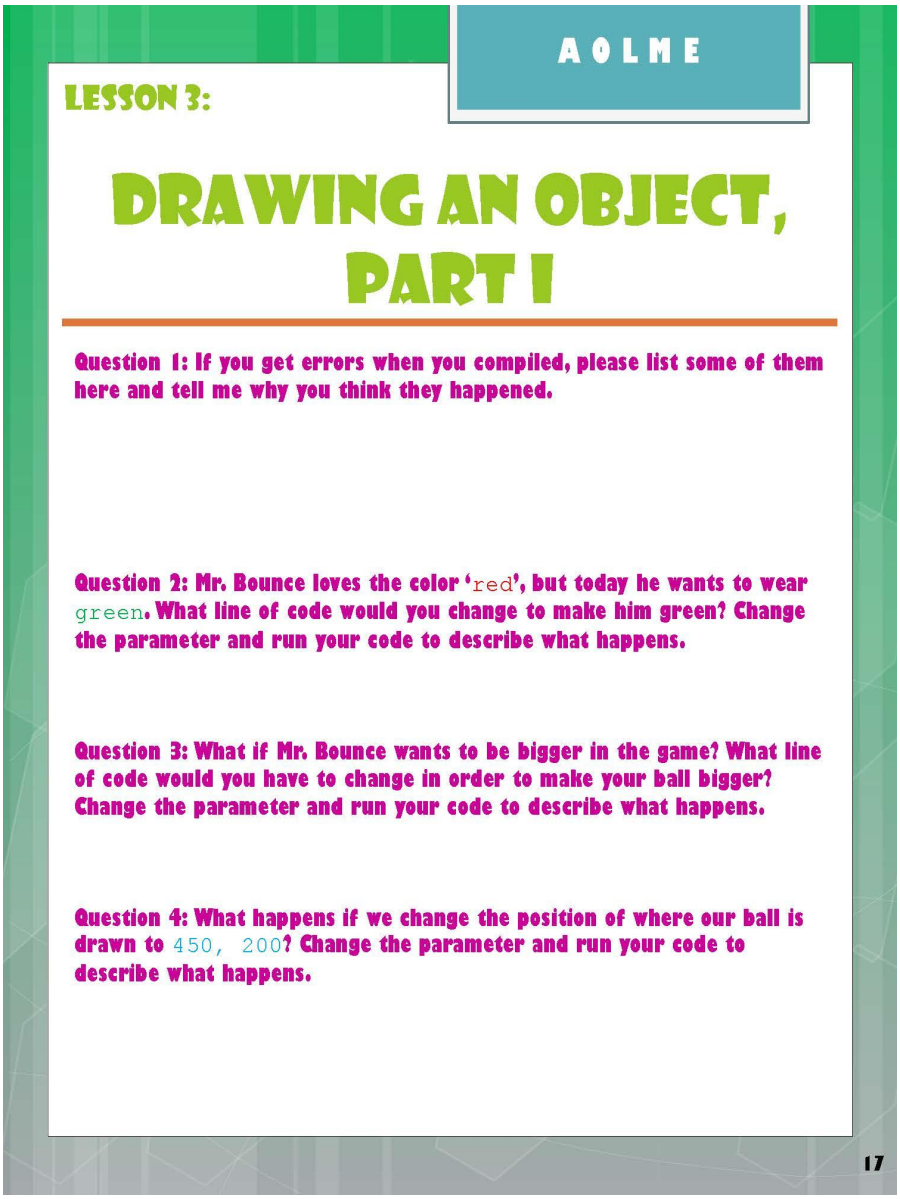


Figure 7: Lesson 3 Questions

Question 2 asks the students to change the color of their ball from red to green. Students are expected to go back through their workbook and look at the explanations to find which line needs to be changed. If they review the explanations, they will find that on slide 15, there is an explanation on how we give the ball its color, and that there are many other colors they can choose from. If they change this line, they will answer the question with ease.

In much the same way, Question 3 asks the student to figure out what line of code needs to be changed in order to make their ball bigger. If they have read the explanations, they will find that slide 13 (shown in Figure 6 above) describes how we make a ball by providing x and y coordinates to make it a particular size. If the students change these x and y coordinates, and then compile and run their code, they will see that they are adjusting the size of their ball.

3.4 Providing and Running Programs with Errors or Missing Code

In order to teach the students valuable programming skills, we must be able to show them what to expect when things go wrong. In following the 3-step approach, we have already given the students a program with working source code; however, this only teaches the student what to expect when everything is correctly done. If we provide the student with working code, and then purposely break this code by introducing errors or removing pieces of code, we will be able to show them what happens when these errors occur, and what they need to do to fix the issue.

There are two lessons that can be learned from this approach. The first lesson that a student will encounter is the ability to compare working code against the code that is not performing as expected. This is very beneficial because the student will be able to see where they may have gone wrong in their own rendition of their program.

The second lesson that the student will learn, however, is the most valuable. The point of providing working code and then introducing errors into it is to teach the student what to expect when things go wrong. This will aid in their future ability

to create better, more efficient programs. By showing students what errors and issues they may run into, we can preemptively provide them the tools to debug their own programs when issues arise, while at the same time, potentially introduce some new, and interesting results. These new outcomes are the result of a hidden lesson to be learned from these errors. For example, let's say we have provided the students with the piece of working code shown in Figure 8 in Section 3.4.1. In keeping with the format introduced in the previous sections, we will first run the code to show the students what happens and what to expect when everything is correct. Then we will step line-by-line through the code and explain to the students what each line is doing, and how it affects the program. Finally, we will ask the students to write their own version of the program provided and ask them to play around with the parameters and inputs. Some of the changes implemented by the students will cause errors. This will allow the student to go back and figure out where they went wrong. However, some of the changes might produce a good and unexpected result. Using our example in Figure 8, let's say that during one of these sessions, a student changes some parameters and suddenly discovers that they have found a way to change the direction of their object. The original code only had the object moving up and down, however, with just a small change in the code, a student might be able to discover the code needed to make their object move left to right.

In my opinion, by writing code through trial and error, the student is allowed to use their imagination and their own intuition to make their program even better than what it was once before. The student will learn how to implement new results, while at the same time learn where they may have gone wrong when errors occur.

By providing a working piece of code for the student to work with, and then introducing errors, we have given them the tools to be creative, without the added stress of asking them to create a program entirely from memory. The working code and the error-ridden code will be there to reinforce their knowledge and help them to better their programming skills.

3.4.1 Lesson Plan Example for Section 3.4

As shown in Figure 8, Lesson 5 has the student learning how to make their ball move up. There is no further discussion on how a student might make the ball move in a different direction. The goal of this workbook is to provide the student with enough information that they can figure out what line of code needs to be changed, and then change this line to experiment with different parameters.

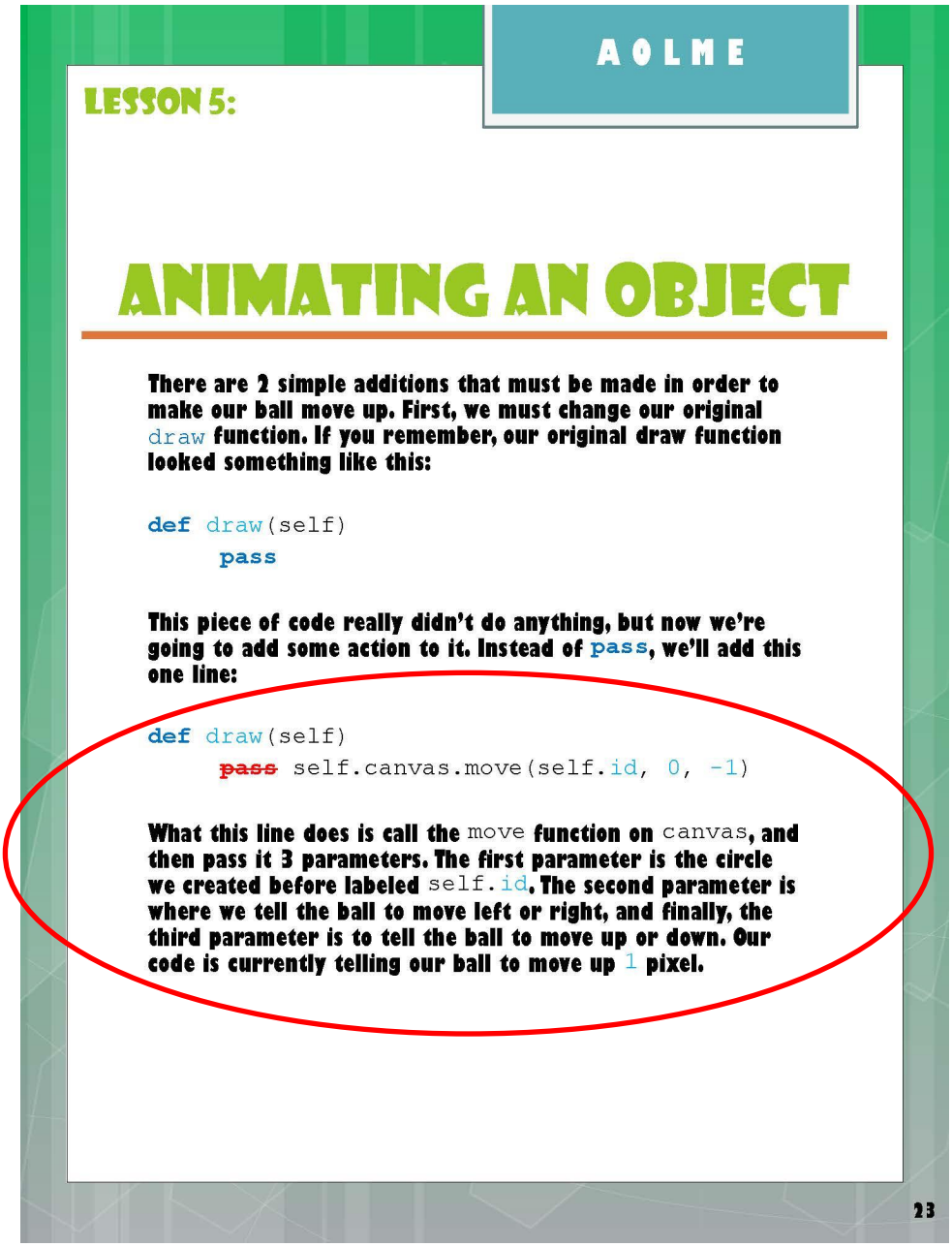


Figure 8: Lesson 4 Example

The red circle in Figure 8 is showing us the explanation given for this line. We have purposely told the student that this line is where we tell the ball what direction to move; however, we only provide the up direction for our sample code. It is up to the student to figure out what needs to be done in order to change the direction. The

questions for this section are the areas in which the student will change parameters. These questions are shown in Figure 9 below.

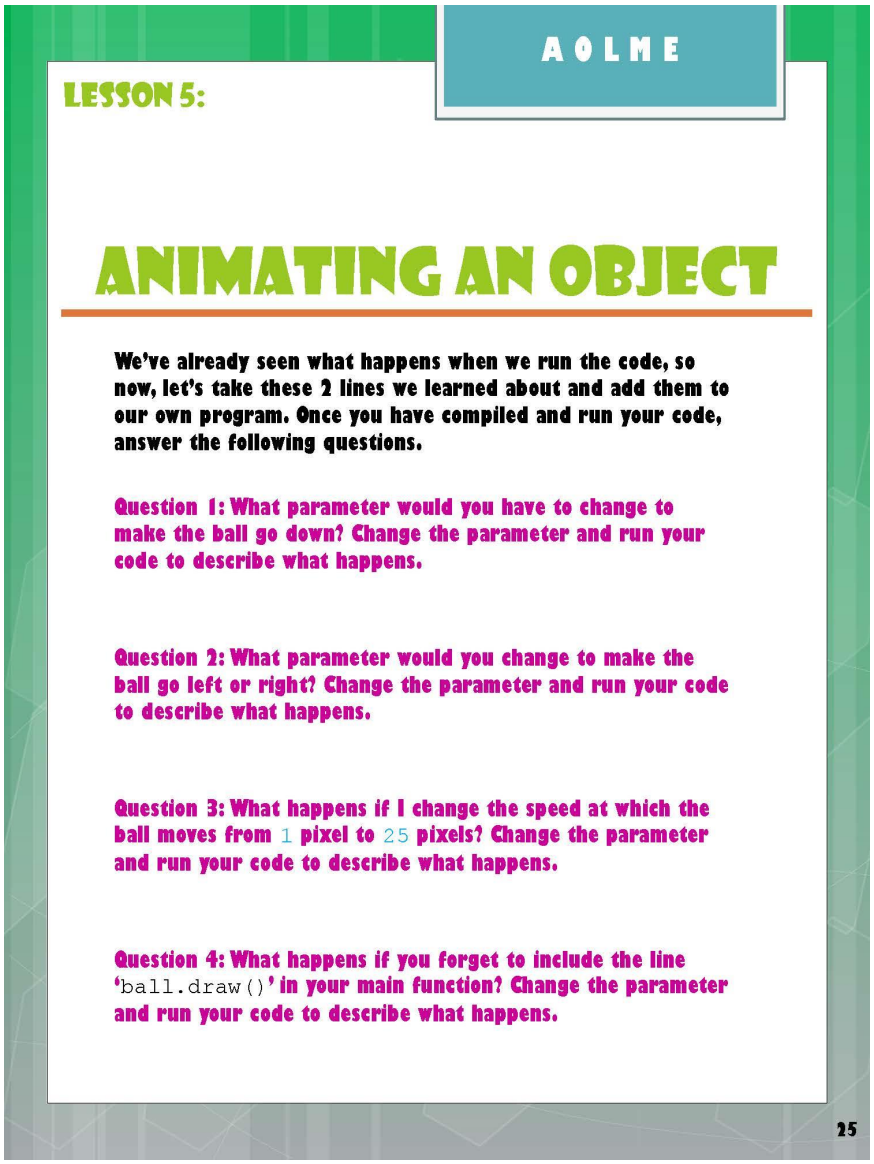


Figure 9: Lesson 4 Questions

Questions 1 and 2 ask the students to figure out how to change the direction of their ball. If they have read the explanation on slide 23 (Figure 8), they will find a discussion on how to make their ball move left, right, or down, although the code to do that is not explicitly given to them. It is up to them to figure that part out. There

may be errors as the students try different inputs, but there may be a discovery as well. Aside from making their ball go up, down, left, or right, the students may happen upon the parameters that they need to make the ball go diagonal. This can only happen if the student decides to put a value in both parameters. Increasing the value will also increase the speed at which the ball travels, another gem that we purposely did not provide the students. These examples were not discussed in the hope that the students will discover them on their own.

Question 4 asks the student to purposely remove a key line in their code and then describe what happens when they run the code. This may not cause an error, but it will cause the program to not run correctly. This question is there to make sure the students know what to expect and how to fix it if the issue should ever come up again.

3.5 The Paddle Battle! Activity Workbook

We've already seen some examples of what the activities will look like in the workbook in the previous section. We will now discuss what each lesson will include, the layout of the workbook overall, and how to use it.

Instructors are expected to let the students program their own programs with as little assistance as possible. The explanations in the beginning of each lesson will be taught by the instructor, but students are expected to take those explanations, and the templates provided to create their own version of the game called Paddle Battle! The code for the Paddle Battle! Game is an adaptation of the code found in the book titled Python for Kids by Jason R. Briggs [10]. There are a few changes to

the original code to make the game a little easier to explain and program, as well as to rid the original program of any excess code. For a more visual and thorough reference, please refer to APPENDIX A**: The Paddle Battle! Workbook** for the activity workbook.

3.5.1 Introduction

In the beginning of the activity workbook the student is introduced to the main character of the Paddle Battle! Game named Mr. Bounce. Mr. Bounce is a colorful character who has decided that he would like to teach our students how to program his favorite game Paddle Battle! Mr. Bounce understands that programming can be difficult at times, so he starts the students off with the basics of Python.

3.5.2 Lesson 1: Paddle Battle in Action! (15 min.)

Lesson 1 is the first, and most important lesson to get the students started. The instructor is expected to run the completed and error-free code for the students to watch and see what they should expect when their version of the game is complete. At this point in time, the instructor will not provide any explanations on the actual lines of code. This lesson is just to introduce the students to the game, and then ask them to answer a few questions on what the game does while it runs. This particular lesson should only take about 15 minutes to allow for adequate time viewing the code in action, and answering the questions provided in the workbook. The following lessons will be where the instructor will start getting into the explanations of code, but in much smaller segments.

3.5.3 Lesson 2: Building a Canvas (30 min.)

In this lesson, students are introduced to the tkinter package. This package is what makes the entire program work. Students will be shown eleven lines of code for this section. When complete, these lines of code actually do a lot to prepare the program for future code that will be introduced earlier, even though all that happens is a window appears. Just like Lesson 1 before, the instructor will run the snippet of code to show the students what to expect. The instructor will then go line-by-line and explain what each line of code is doing. Finally, after all explanations are complete, the students will be asked to write their own code modeled after the code given and explained to them earlier. This lesson should take about 30 minutes to complete.

While the students are writing and running their programs, the instructor is expected to walk around and assist students who are struggling. Students are encouraged to change and experiment with any parameters in their code. Once the students have completed their small program, and it runs correctly, the students are then expected to answer questions on what happens to the program when they change specific parameters given in the workbook questions for Lesson 2. These questions are to ensure that students actually understand the concepts and fundamentals of the lines of code they were just introduced to.

3.5.4 Lesson 3: Drawing an Object, Part I (45-50 min.)

Now that the students have a small piece of working code for their Paddle Battle! Game, they will now continue to build on it by creating their first character in this

lesson. Lesson 3 introduces the concept of the class and the object. Once again, students will be shown what the working code will do when run. Instructors will then go line-by-line and explain the 8 lines of code that will be introduced, as well as explain where in their existing code, these lines will be located. The students will then need to write their own code into their own programs and answer questions.

The first question asks the students to describe what happened when they got any errors and to explain why they think they happened. The remaining questions are modeled to see if the students understand what certain lines of code actual do. There are already explanations that the instructor will give that provide hints on what the lines specifically do, so the answers to these questions should not be difficult to implement and answer. Because the students are learning new concepts, this lesson should take about 45 -50 minutes to complete to allow for adequate time for questions.

3.5.5 Lesson 4: Drawing an Object, Part II (30-45 min.)

In this lesson, students will be expected to write about 95% of the code with little instruction. The Paddle class that they need to create is essentially the Ball class that they just created with one minor change. The change they will need to do is to create a rectangle instead of an oval. This lesson should take anywhere between 30-45 minutes to complete, especially if the students understand that the classes are fundamentally the same.

3.5.6 Lesson 5: Animating an Object (20 min.)

To make Mr. Bounce move, we simply need to add 3 lines of code. Although these lines of code are very short and simple, they contain a lot of information that the instructor must explain thoroughly. The instructor will run the new sample of code and then explain the lines accordingly. Students will write their own program and then answer the questions that follow. These questions will ask the students what parameters they would need to change in order to achieve a particular action. These questions are written with the hope that the students will stumble upon code that will not be introduced by the instructor or this workbook. The hope is that the students will discover new and interesting things they can do with their programs, such as figuring out how to make their ball move left and right. This lesson should only take about 20 minutes to explain and complete.

3.5.7 Lesson 6: Movement and Collision Detection (60 min.)

This is the lesson where the students will actually learn how to implement animation, collision detection, speed control, and determining movements. This lesson is highly critical to the body of the student's game. The lesson will take roughly 60 minutes to complete. The students will be given about half of the code for this lesson, but they will be asked to write their own pieces in question 2. Explanations given by the instructor should be precise and methodical. Instructors must answer as many questions as possible that pertain to collision detection. It is imperative that they understand that concept, especially because it will help them to progress in the next lesson.

3.5.8 Lesson 7: Adding Movement with Keyboard Inputs (60 min.)

In this lesson, the students will be learning how to make an object move through the use of events such as pressing a key on a keyboard. The students will be expected to program a significant portion of this section, but those lines of code that pertain to using events will be given to the students. By the time they have completed this lesson, which should take about 60 minutes, they will have a ball that bounces around the screen, and a paddle that moves from side to side. The following lesson is where we will test their skills and see if they can complete the game.

3.5.9 Lesson 8: On Your Own (60 min. x 3)

By the time the students get to this lesson, they should have the tools they need to make what they have so far into a real game. The three questions in this section are programming questions. All three questions build on the code that they have generated thus far; however, these questions are written to have them develop the game more with little instruction. Each question in this lesson should take at least 60 minutes to complete. Although the answers seem easy to implement, this is really the first time we are asking the students to think about and develop their own code without actually walking them through what they need to do.

3.6 The Paddle Battle! Instructor's Manual

The instructor's manual contains all the code needed as the students step through the workbook. Each lesson's code is provided, and sample answers are provided for the questions. For the On Your Own section, the code is provided with some descriptions as to what the students will need to know in order to complete these

questions; however, there really should be little help given for answering these questions. These questions were chosen specifically so that the students would have a somewhat complete game to work with, but still have pieces of it that they need to implement. Their game will not really be a game until they implement the first 2 questions of the lesson. The third question is more of an added bonus if they can get it. For a more visual and thorough reference, please refer to APPENDIX B**: The Paddle Battle! Instructor's Manual** for the instructors manual.

Chapter 4

Future Work and Conclusions

4.1 Future Work

This thesis is a proposal for AOLME to continue to introduce more complex, but interesting topics to teach their students. If the AOLME group accepts this approach discussed in this report, future work will comprise of revising and improving the activity workbook. Instructors will be taught how to use the workbook by implementing the activities themselves so that they may draw on their experiences, as well as provide ideas on how to make it better.

Other future work will be to create other workbooks for more complex topics. A pong-like game is just the start of game programming using Python. There are additional capabilities within Python that can create more complex and interactive games. Workbooks can be created in order to teach these complex topics, but never shy away from being written and explained as simply as possible.

Lastly, a possible introduction to 3D environments could be considered for teaching students how to program. Originally, AOLME thought that perhaps this was the way to go, especially because the students were asking for more interactive and visual programming using 3D characters. Unfortunately, in order to introduce 3D objects, the students need to have a thorough understanding of angles, geometry, and intense mathematical reasoning that goes beyond middle-school mathematics. AOLME students use algebra and topics such as coordinate systems to develop their

programs. In order to introduce 3D programming, AOLME will have to develop a set of lessons that explain the different mathematical topics that the students will need to know in order to create their 3D images. Future work for this potential topic would be the lessons on mathematics, as stated above, as well as very simple programs for the students to create after they have become comfortable with the new mathematical subjects.

4.2 Conclusion

There are many introductory programming environments that are GUI-based and interactive. To provide access to computing practices, there needs to be an understanding of how programs operate at the source code level. The way to introduce source-code level programming is by providing an environment that engages the student and allows them to learn actual material using a general-purpose programming language. By using Python, and introducing programming concepts through the creation of games, we are able to keep the interest of the students while teaching them how to program using a programming language that will support their growth on and understanding of computing practices.

APPENDIX A: The Paddle Battle! Workbook

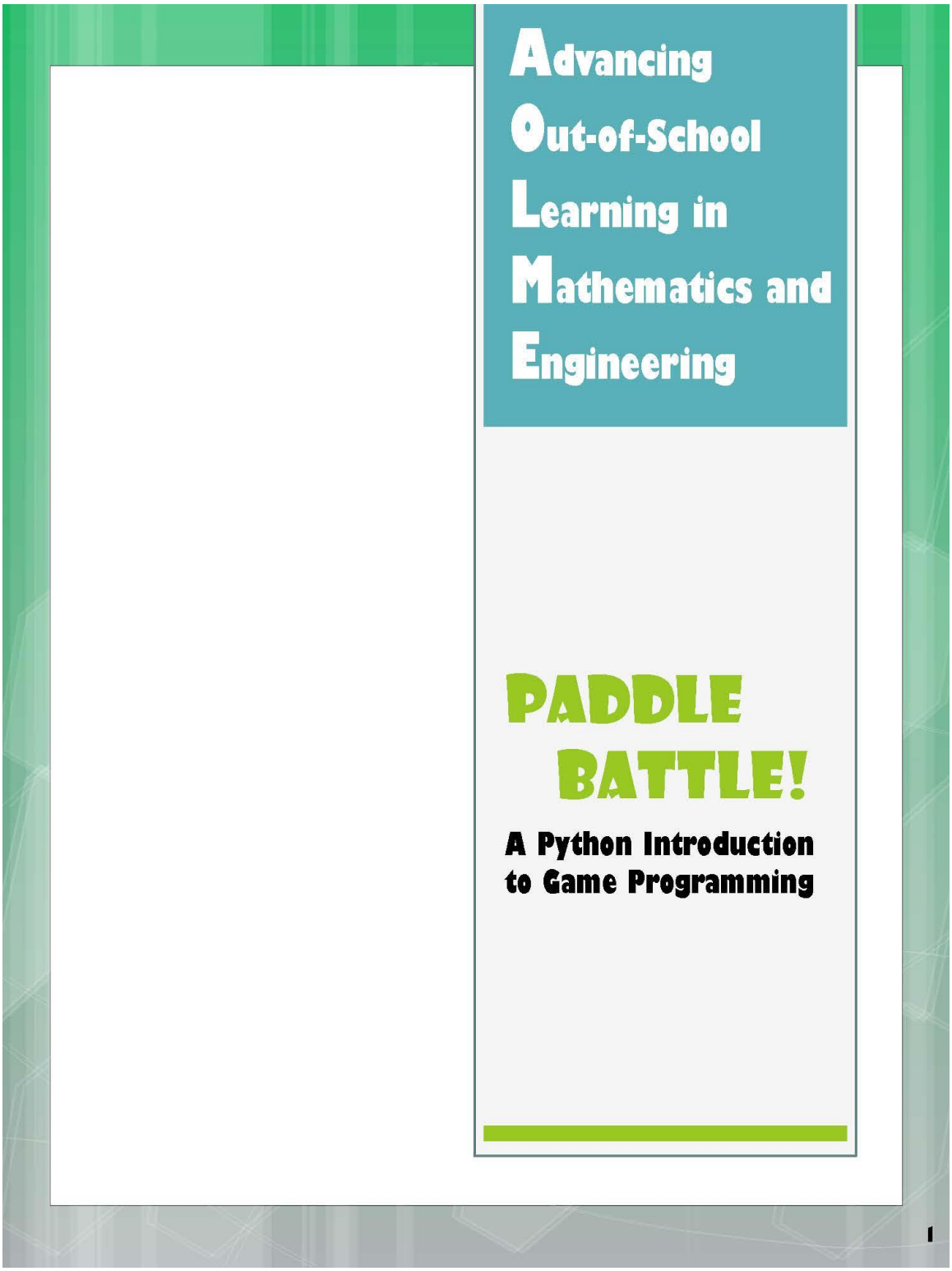


TABLE OF CONTENTS

AOLME

INTRODUCTION

Hi There!

Welcome to your first lesson in programming exciting games in Python! In this workbook we are going to learn how to create new and sensational things using Python, and in the end, we will write our very own simple game using all the things we learned, and of course, starring me! But first, let's start off with the basics!

~ Mr. Bounce

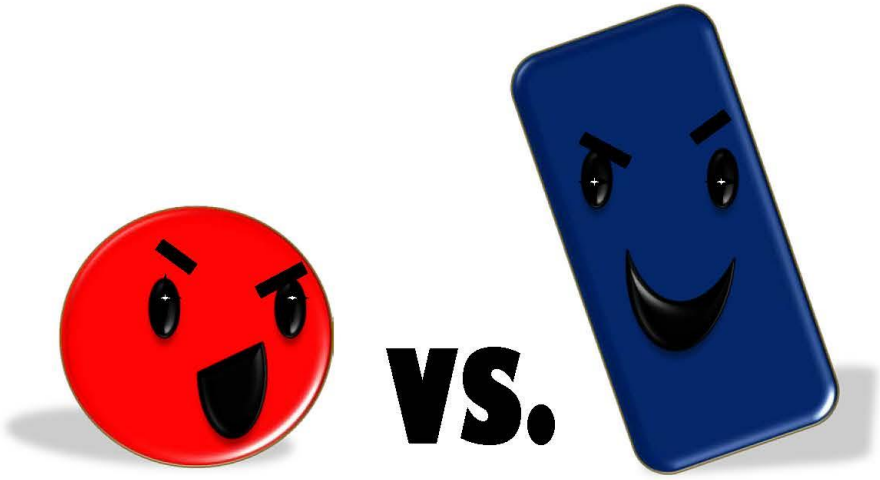
3

AOLME

LESSON 1:

PADDLE BATTLE IN ACTION!

In this exercise, let's take the Paddle Battle! code we gave you and run it to see what happens. Be prepared to answer the five questions on the following page.



4

AOLME

LESSON 1:

PADDLE BATTLE IN ACTION QUESTIONS!

- **Question 1: What does the ball do?**
- **Question 2: What does the paddle do?**
- **Question 3: What happens when the ball hits the sides or the top of the window?**
- **Question 4: What happens when the ball hits the paddle?**
- **Question 5: What happens if the ball hits the bottom of the window?**

5

AOLME

LESSON 2:

BUILDING A CANVAS

In order to start programming our own game, we must first understand the basics that make the program work. Our first lesson is on a package called `tkinter`. `tkinter` is a package in Python that allows you to make a drawing board called a `canvas`, in order to create all the wonderful characters and animations you dream up. Let's look at the example code from the Paddle Battle! program on the next page and run it to see what happens.

6

LESSON 2:

BUILDING A CANVAS

```
1. from tkinter import *
2. import random
3. import time
4.
5. tk = Tk()
6. tk.title("Paddle Battle!!")
7. tk.resizable(0, 0)
8. tk.wm_attributes("-topmost", 1)
9. canvas = Canvas(tk, width=500, height=500)
10.canvas.pack()
11.tk.update()
```

Before we build our own canvas, let's explore the code line-by-line.

Line #	Code	Explanation
1	`from tkinter import *`	This line is where we import the `tkinter` package. Now we will be able to use all of the tools form `tkinter` to make our Paddle Battle game.
2	`import random`	Here we are importing the `random` package. This package will be discussed more and used a little later in our workbook.
3	`import time`	Here we are importing the `time` package. This package, like `random`, will be discussed more and used a little later in our workbook.

AOLME

LESSON 2:

BUILDING A CANVAS

```
1. from tkinter import *
2. import random
3. import time
4.
5. tk = Tk()
6. tk.title("Paddle Battle!!")
7. tk.resizable(0, 0)
8. tk.wm_attributes("-topmost", 1)
9. canvas = Canvas(tk, width=500, height=500)
10.canvas.pack()
11.tk.update()
```

Before we build our own canvas, let's explore the code line-by-line.

Line #	Code	Explanation
5	`tk = Tk()`	Here we are building a `tk` object so that we can create our window.
6	`tk.title("Paddle Battle!!")`	Here we are using the `tk` object we created in the line above so that we can give our new window a name.
7	`tk.resizable(0, 0)`	This line sets how big our window can be resized. In this case, `(0, 0)` pixels means that the window is not `resizeable`, and cannot be changed horizontally or vertically.

8

AOLME

LESSON 2:

BUILDING A CANVAS

```
1. from tkinter import *
2. import random
3. import time
4.
5. tk = Tk()
6. tk.title("Paddle Battle!!")
7. tk.resizable(0, 0)
8. tk.wm_attributes("-topmost", 1)
9. canvas = Canvas(tk, width=500, height=500)
10.canvas.pack()
11.tk.update()
```

Before we build our own canvas, let's explore the code line-by-line.

Line #	Code	Explanation
8	`tk.wm_attributes("-topmost", 1)`	This line may seem confusing, but all this line is doing is making sure that when our game window loads, that it loads over all the other windows (`-topmost`).
9	`canvas = Canvas(tk, width=500, height=500)`	Here we are creating a `canvas` object and giving it some parameters. We give it the `tk` object, and then give it a height and width of `500` pixels as it's window size.
10	`canvas.pack()`	This line tells the window to follow the parameters we gave it in the line above.
11	`tk.update()`	This line tells the tkinter library to prepare itself for animations to come.

9

AOLME

LESSON 2:

BUILDING A CANVAS

```
1. from tkinter import *
2. import random
3. import time
4.
5. tk = Tk()
6. tk.title("Paddle Battle!!")
7. tk.resizable(0, 0)
8. tk.wm_attributes("-topmost", 1)
9. canvas = Canvas(tk, width=500, height=500)
10.canvas.pack()
11.tk.update()
```

Now that we have gone through each line of code, and understand what each line does, lets run this piece of code to again to see it in action.

Question 1: What happens when we run this code?

Now, let's write our own code using the above code as an example. Be sure to ask questions if you run into trouble, and remember, if you get an error, go back and compare your code to see where you may have went wrong. No error is a bad error. Once you have written your own program, run it and then try answering these questions.

Question 2: Let's change the width of the canvas to `30`, and our height to `300`, and then run our code. Describe what happens

Question 3: Remove the line that says "`canvas.pack()`", and then run your code. Describe what happens.

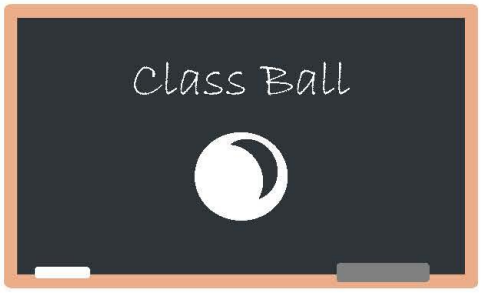
10

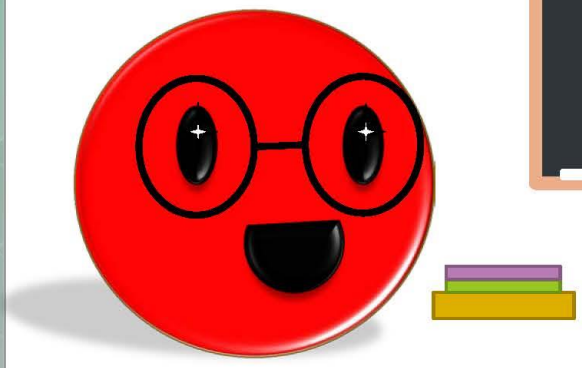
AOLME

LESSON 3:

DRAWING AN OBJECT, PART I

Now that we have made our window for our own Paddle Battle! game, we can now move on to more difficult topics such as creating the ball and making it move. In this lesson we will be writing our own `class` for our ball. A `class` is a part of the program that defines a type of object. A class performs tasks, calculations, and provides information that an object of the class will need once it is created. Let's run our sample code now to see what happens!





11

AOLME

LESSON 3:

DRAWING AN OBJECT, PART I

```
1.  class Ball:
2.      def __init__(self, canvas, color):
3.          self.canvas = canvas
4.          self.id = canvas.create_oval(10, 10, 25, 25, fill=color)
5.          self.canvas.move(self.id, 250, 250)
6.
7.      def draw(self):
8.          pass
```

Before we build our own ball, let's explore the code line-by-line. There are quite a few new concepts introduced in this small piece of code. Make sure and pay attention to the explanations given for each line.

Line #	Code	Explanation
1	`class Ball:`	In this line, we are simply naming our class. Make sure when you name a class or variable, that you name it something with meaning so you know what it's for later on.
2	`def __init__(self, canvas, color):`	When we have this line in our code, we are initializing our function, or `class Ball`. In the parentheses, we are sending it three key things it will need in order to initialize it. We send it itself, to create the function. We then send it `canvas`, which we made earlier and will need it in order to draw our ball, and lastly, we send it `color` so we are able to set a color for our ball.
3	`self.canvas = canvas`	In this line we are simply setting the object `canvas` we created earlier, to the value of the parameters given by canvas.

12

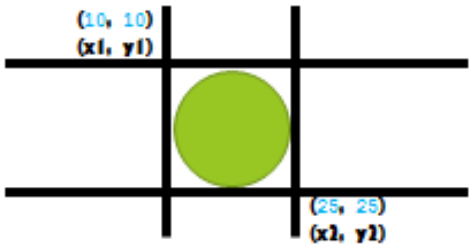
AOLME

LESSON 3:

DRAWING AN OBJECT, PART I

```
1.  class Ball:
2.      def __init__(self, canvas, color):
3.          self.canvas = canvas
4.          self.id = canvas.create_oval(10, 10, 25, 25, fill=color)
5.          self.canvas.move(self.id, 250, 250)
6.
7.      def draw(self):
8.          pass
```

Line 4 above is an interesting line. Here we call our `create_oval` function. You need to give `create_oval` 5 parameters in order for it to work correctly. We need to supply `x` and `y` coordinates for the top and bottom of the oval so we know how big our circle will be, and its color. In this case, the `x` and `y` coordinates for the top are `10`, and `25` for the bottom. This tells us that the circle we are creating will fit into a rectangle defined by these coordinates, like the example below. We will set the `color` for our ball a little later.



There are also many shapes you can make such as `create_rectangle` and `create_polygon`. `create_rectangle` will draw a rectangle, and `create_polygon` allows you to draw your own shape using parameters you give it.

13

AOLME

LESSON 3:

DRAWING AN OBJECT, PART I

```
1.  class Ball:
2.      def __init__(self, canvas, color):
3.          self.canvas = canvas
4.          self.id = canvas.create_oval(10, 10, 25, 25, fill=color)
5.          self.canvas.move(self.id, 250, 250)
6.
7.      def draw(self):
8.          pass
```

Line #	Code	Explanation
5	`self.canvas.move(self.id, 250, 250)`	This line is where we tell our program where we would like the ball to be drawn on our canvas. We give it our object `id`, and the pixel coordinates where we want `id` drawn `(250, 250)`.
7	`def draw(self):`	This line creates our `draw` function so that we can draw our ball
8	`pass`	This line is just to move the code on. More code will be added as we develop our game.

AOLME

LESSON 3:

DRAWING AN OBJECT, PART I

Now, if we ran this code as it is, we would see that our ball does not appear on the screen. That's because we are missing a key line like the one below:

```
Line # 10: ball = Ball(canvas, 'red')
```

This line is where the magic happens. We've set our `class Ball:` equal to it's object `ball` and then given it `canvas` as a parameter (so we can draw on the canvas) and the `color` we want our ball to be. There are many other colors you can choose such as green and blue, just explore to find more!

However, there is one more thing we need to program, and that is our main function. A main function or loop is what controls all the things we created in our program. Right now, all we need our main function to do is make sure the window stays open until we close it, and make our ball appear. So we introduce the following code at the end of our program:

Line #	Code	Explanation
11	`while 1:`	This is the start of a `while` loop. A loop is a function that repeats an action until the set parameter is met. This particular loop runs until we close the window.
12	`tk.update_idletasks()`	This line tells our program to update the window from time to time.
13	`tk.update`	This line is from our previous code we wrote earlier. It tells the `tkinter` library to prepare itself for animations to come.
14	`time.sleep(0.01)`	This line tells `tkinter` to redraw itself and then sleep for `0.01` seconds.

15

AOLME

LESSON 3:

DRAWING AN OBJECT, PART I

After all is said and done, we can now add the Ball class to the sample code from before. The sample code should now look like the code below. Lines added have a comment next to them in green.

```
from tkinter import *
import random
import time

tk = Tk()
tk.title("Paddle Battle!!")
tk.resizable(0, 0)
tk.wm_attributes("-topmost", 1)
canvas = Canvas(tk, width=500, height=500)
canvas.pack()
tk.update()

class Ball: # Lesson 3 code
    def __init__(self, canvas, color): # Lesson 3 code
        self.canvas = canvas # Lesson 3 code
        self.id = canvas.create_oval(10, 10, 25, 25, fill=color) # Lesson 3 code
        self.canvas.move(self.id, 250, 250) # Lesson 3 code

    def draw(self): # Lesson 3 code
        pass # Lesson 3 code

ball = Ball(canvas, 'red') # Lesson 3 code

while 1: # Lesson 3 code
    tk.update_idletasks() # Lesson 3 code
    tk.update() # Lesson 3 code
    time.sleep(0.01) # Lesson 3 code
```

Now that we have a working example, and have seen how the program works, let's add the code we learned about in this lesson to our existing code from Lesson 2. Remember, if you get any errors while you are trying to program your own class, use the sample code as a guide. Once you have completed your program, answer the questions on the following page.

16

AOLME

LESSON 3:

DRAWING AN OBJECT, PART I

Question 1: If you get errors when you compiled, please list some of them here and tell me why you think they happened.

Question 2: Mr. Bounce loves the color 'red', but today he wants to wear green. What line of code would you change to make him green? Change the parameter and run your code to describe what happens.

Question 3: What if Mr. Bounce wants to be bigger in the game? What line of code would you have to change in order to make your ball bigger? Change the parameter and run your code to describe what happens.

Question 4: What happens if we change the position of where our ball is drawn to 450, 200? Change the parameter and run your code to describe what happens.

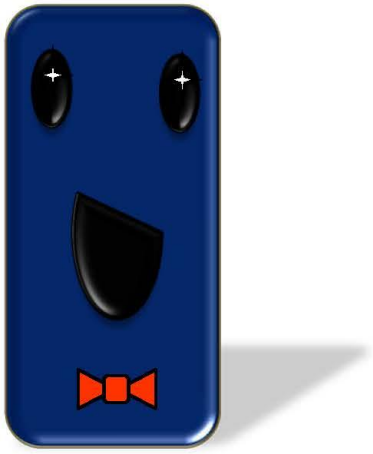
17

AOLME

LESSON 4:

DRAWING AN OBJECT, PART II

Now that we've created Mr. Bounce, it's time to introduce his adversary, the Paddle. We have already learned how to create objects on our canvas. In this lesson, you will create your own paddle by simply looking at the code we used when we made the ball. The only difference, Paddle is a rectangle, not an oval. Let's run our sample code and see what happens!



18

AOLME

LESSON 4:

DRAWING AN OBJECT, PART II

```
1.  class Ball:
2.      def __init__(self, canvas, color):
3.          self.canvas = canvas
4.          self.id = canvas.create_oval(10, 10, 25, 25, fill=color)
5.          self.canvas.move(self.id, 250, 250)
6.
7.      def draw(self):
8.          pass
```

The Ball and Paddle classes will be very similar. Let's start by creating and adding our own Paddle class after the Ball class in our code. Use the Ball class code above as a model for what your Paddle class will look like.

Question 1: If we want to create a rectangle instead of an oval, what line do you think we should change? Write the line and show the changes.

The line that should be changed is the `create_oval` line on line 4. The new line should look like the following:

```
self.id = canvas.create_rectangle(0, 0, 100, 10, fill=color)
```

Make sure and experiment with the size of your paddle by changing the `x` and `y` coordinates inside the `create_rectangle` function. Figure out which size paddle is best for your game!

Question 2: We should also position our new paddle in a different location to make sure we aren't covering up our Ball. What line should we change if we want to change the position of our Paddle? Write the line and show the changes.

19

AOLME

LESSON 4:

DRAWING AN OBJECT, PART II

```
1.  class Ball:
2.        def __init__(self, canvas, color):
3.              self.canvas = canvas
4.              self.id = canvas.create_oval(10, 10, 25, 25, fill=color)
5.              self.canvas.move(self.id, 250, 250)
6.
7.        def draw(self):
8.              pass
```

The line that should be changed for Question 2 is the `self.canvas.move` line on line 5. The new line should look like the following:

```
self.canvas.move(self.id, 200, 300)
```

As with the rectangle parameters before, experiment with the position of your paddle by changing the pixel coordinates inside the `canvas.move` function. Figure out which position is best for your paddle!

Question 3: What happens if we run the code as it is now? Hint: We ran the Ball class code once when we only had these lines. Do you remember what happened? Run your code and describe what happens.

Question 4: What line are we missing in order to make the paddle appear on our canvas? Hint: You might find an example of it in Lesson 3. Add the line to your code and then run your program to describe what happens.

20

AOLME

LESSON 4:

DRAWING AN OBJECT, PART II

As a guide, your code should now look something like the following:

```
from tkinter import *
import random
import time

tk = Tk()
tk.title("Paddle Battle!!")
tk.resizable(0, 0)
tk.wm_attributes("-topmost", 1)
canvas = Canvas(tk, width=500, height=500)
canvas.pack()
tk.update()

class Ball:
    def __init__(self, canvas, color):
        self.canvas = canvas
        self.id = canvas.create_oval(10, 10, 25, 25, fill=color)
        self.canvas.move(self.id, 250, 250)

    def draw(self):
        pass

class Paddle: # Lesson 4 code
    def __init__(self, canvas, color): # Lesson 4 code
        self.canvas = canvas # Lesson 4 code
        self.id = canvas.create_rectangle(0, 0, 100, 10, fill=color) # Lesson 4
code
        self.canvas.move(self.id, 200, 300) # Lesson 4 code

    def draw(self): # Lesson 4 code
        pass # Lesson 4 code

ball = Ball(canvas, 'red')
Paddle = Paddle(canvas, 'blue') # Lesson 4 code

while 1:
    tk.update_idletasks()
    tk.update()
    time.sleep(0.01)
```

Lines added have a comment next to them in green.

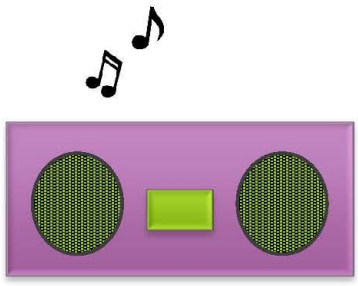
21

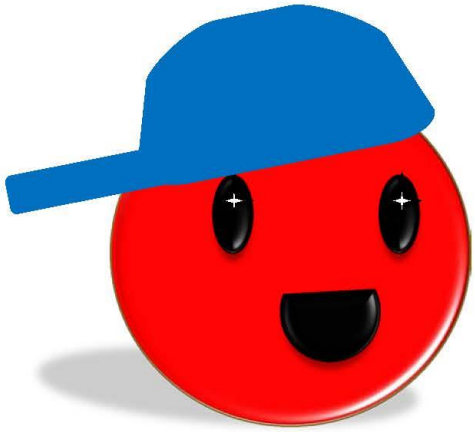
AOLME

LESSON 5:

ANIMATING AN OBJECT

We now have a working program where we have successfully drawn a ball and a paddle. However, Mr. Bounce loves to dance and needs to be able to move. In this lesson we will learn how to make our ball move on it's own using the code we've already written. Let's run our sample code now to see what happens!





22

AOLME

LESSON 5:

ANIMATING AN OBJECT

There are 2 simple additions that must be made in order to make our ball move up. First, we must change our original `draw` function. If you remember, our original draw function looked something like this:

```
def draw(self)
    pass
```

This piece of code really didn't do anything, but now we're going to add some action to it. Instead of `pass`, we'll add this one line:

```
def draw(self)
    pass self.canvas.move(self.id, 0, -1)
```

What this line does is call the `move` function on `canvas`, and then pass it 3 parameters. The first parameter is the circle we created before labeled `self.id`. The second parameter is where we tell the ball to move left or right, and finally, the third parameter is to tell the ball to move up or down. Our code is currently telling our ball to move up `1` pixel.

23

LESSON 5:

ANIMATING AN OBJECT

The second change we are going to add to our code is in our main function we created earlier. Our original main function looked like this:

```
while 1:
    tk.update_idletasks()
    tk.update()
    time.sleep(0.01)
```

Now we are going to add this line:

```
ball.draw()
```

This line tells the program to start performing all the actions you just added in your `draw` function. In other words, it will tell your program to start drawing your ball and then move it `1` pixel up.

Since we know that in order to make our ball move we need this line, do you think that maybe we will need to do this for our paddle as well? The answer is yes. Go ahead and add `paddle.draw()` to your code as well. Adding this line won't make your paddle move. We still need to add code to our Paddle class to make it move. We'll learn how to do that in Lesson *7*.

AOLME

LESSON 5:

ANIMATING AN OBJECT

We've already seen what happens when we run the code, so now, let's take these 2 lines we learned about and add them to our own program. Once you have compiled and run your code, answer the following questions.

Question 1: What parameter would you have to change to make the ball go down? Change the parameter and run your code to describe what happens.

Question 2: What parameter would you change to make the ball go left or right? Change the parameter and run your code to describe what happens.

Question 3: What happens if I change the speed at which the ball moves from `1` pixel to `25` pixels? Change the parameter and run your code to describe what happens.

Question 4: What happens if you forget to include the line '`ball.draw()`' in your main function? Change the parameter and run your code to describe what happens.

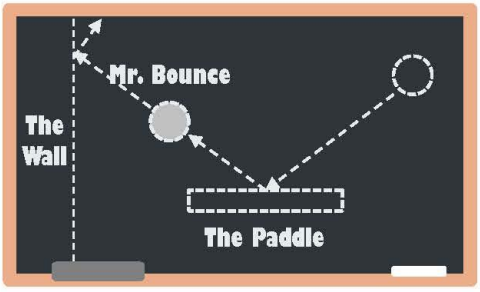
25

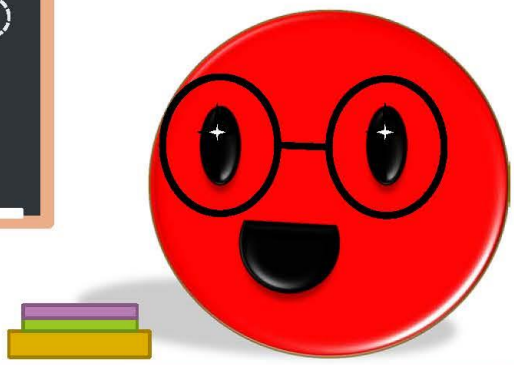
AOLME

LESSON 6:

MOVEMENT AND COLLISION DETECTION

Now that we've created Mr. Bounce and the Paddle, we're ready to introduce some movements. Right now in the sample, all that Mr. Bounce does is go up and out of the screen. This really isn't very fun or interesting. In this lesson we're going to discuss the topic of collision detection. Collision detection is where we figure out when our object crashes into or hits another object. In our case, when Mr. Bounce hits a wall or the Paddle, we want Mr. Bounce to bounce away from it. Let's run our sample code and see what happens!





26

AOLME

LESSON 6:

MOVEMENT AND COLLISION DETECTION

Before we began adding in the code to make our ball bounce, let's first go line by line through the new code. This set of code is a little more difficult to understand. We will first discuss how to make the ball bounce up and down without flying off the canvas. Then we will discuss how to make the ball bounce around the whole canvas. As a guide, your code should now look something like the code below.

```
from tkinter import *
import random
import time

tk = Tk()
tk.title("Paddle Battle!!")
tk.resizable(0, 0)
tk.wm_attributes("-topmost", 1)
canvas = Canvas(tk, width=500, height=500)
canvas.pack()
tk.update()

class Ball:
    def __init__(self, canvas, color):
        self.canvas = canvas
        self.id = canvas.create_oval(10, 10, 25, 25, fill=color)
        self.canvas.move(self.id, 250, 250)

    def draw(self):
        self.canvas.move(self.id, 0, -1)

class Paddle:
    def __init__(self, canvas, color):
        self.canvas = canvas
        self.id = canvas.create_rectangle(0, 0, 100, 10, fill=color)
        self.canvas.move(self.id, 200, 300)

    def draw(self):
        pass

ball = Ball(canvas, 'red')
Paddle = Paddle(canvas, 'blue')

while 1:
    ball.draw()
    paddle.draw()
    tk.update_idletasks()
    tk.update()
    time.sleep(0.01)
```

27

AOLME

LESSON 6:

MOVEMENT AND COLLISION DETECTION

The first thing we want to be able to do is make the ball move or bounce without having it fly off our canvas. We'll be dealing primarily with the Ball class for our code changes in this lesson.

```
1.  class Ball:
2.      def __init__(self, canvas, color):
3.          self.canvas = canvas
4.          self.id = canvas.create_oval(10, 10, 25, 25, fill=color)
5.          self.canvas.move(self.id, 250, 250)
6.
7.      def draw(self):
8.          pass
```

The first thing we need to do is create some new variables. These variables will live in the `def __init__` function of the Ball class, after line 5.

Line #	Code	Explanation
6	`self.x = 0`	This line is setting the variable `x` to the value of `0`.
7	`self.y = -1`	This line is setting the variable `y` to the value of `-1`.
8	`self.canvas_height = self.canvas.winfo_height()`	This line is setting the variable `canvas_height` to the value of `winfo_height`. `winfo_height` returns the value of the current height of your canvas. Our canvas height is `500` pixels.

Question 1: What is the current height of your canvas?

28

AOLME

LESSON 6:

MOVEMENT AND COLLISION DETECTION

Now that we have created our variables, we are going to add some code to the `draw` function of the Ball class. In order to make the ball bounce, and detect when it's about to hit a wall, we need to look at our canvas like it is a graph with coordinates. We've already written the coordinates for our oval variable `self.id`. Now we need to add some lines that will determine the current `x` and `y` coordinates of an object, as well as coordinates to determine when the ball has hit a wall.

Line #	Code	Explanation
11	`self.canvas.move(self.id, self.x, self.y)`	Here we have replaced `0` and `-1` with the new variables we created earlier, `self.x` and `self.y`.
12	`pos = self.canvas.coords(self.id)`	The `pos` variable created here will return the `x` and `y` coordinates of any object drawn on the canvas. In this line, the `x` and `y` coordinates of our oval variable, `self.id`, are returned. The `pos` variable will return a list of four coordinates in the format [`x1, y1, x2, y2`].
13/14	`if pos[1] <= 0:` `  self.y = 1`	`pos[1]` is the value of the `y1` coordinate. In this if statement to the left we are saying, "If the value of `pos[1]` is less than or equal to `0`, then `self.y` equals `1`." In other words, if we hit the top of the canvas, then stop moving up.
15/16	`if pos[3] >= self.canvas_height:` `  self.y = -1`	`pos[3]` is the value of the `y2` coordinate. In this if statement to the left we are saying, "If the value of `pos[3]` is greater than or equal to `canvas_height`, then `self.y = -1`."

29

AOLME

LESSON 6:

MOVEMENT AND COLLISION DETECTION

Our Ball class should now look like the following code below. Lines with a green comment are the lines added in this lesson so far.

```
1.  class Ball:
2.      def __init__(self, canvas, color):
3.          self.canvas = canvas
4.          self.id = canvas.create_oval(10, 10, 25, 25, fill=color)
5.          self.canvas.move(self.id, 250, 250)
6.          self.x = 0 # Lesson 6 Code
7.          self.y = -1 # Lesson 6 Code
8.          self.canvas_height = self.canvas.winfo_height() # Lesson 6 Code
9.
10.     def draw(self):
11.         self.canvas.move(self.id, self.x, self.y) # Lesson 6 Code
12.         pos = self.canvas.coords(self.id) # Lesson 6 Code
13.         if pos[1] <= 0: # Lesson 6 Code
14.             self.y = 1 # Lesson 6 Code
15.         if pos[3] >= self.canvas_height: # Lesson 6 Code
16.             self.y = -1 # Lesson 6 Code
```

Now, let's run your code so that you can answer the following questions.

Question 2: Right now we only have code that detects when the ball hits the top or bottom of the window. What code would we need to add to make sure that the left and right ends (x-axis) of the window are detected by the ball? Add the lines of code you think you need, run your code, and describe what happens. If you run into errors, describe those as well.

Question 3: What do you think we need to do in order to get the ball to move around the window, instead of just up and down? Make a guess.

AOLME

LESSON 6:

MOVEMENT AND COLLISION DETECTION

In order to answer Question 3, we need to introduce a few more lines of code to our program. These lines will essentially change the angle at which the ball starts at. In other words, instead of having our ball start it's movement in a vertical direction, we are now going to start its movement in a random direction.

Once again, the first thing we need to do is create some new variables and edit a few others.

Line #	Code	Explanation
6	`starts = [-3, -2, -1, 1, 2, 3]` **(add before the** `self.x` **variable)**	**Here we have created a variable called** `starts`**.** `starts` **will have a list of 6 values inside of it,** `-3`**,** `-2`**,** `-1`**,** `1`**,** `2`**, and** `3`**.**
7	`random.shuffle(starts)` **(add before the** `self.x` **variable)**	**The** `random.shuffle` **function is called on** `starts` **in this line so that we can change the order of the values in** `starts` **list. In other words, we choose a random direction for our ball.**
8	`self.x = starts[0]`	**The value of** `self.x` **is changed to** `starts[0]`**. This sets** `x` **equal to whatever value** `starts` **becomes after it is shuffled.**
9	`self.y = -3`	**The value for** `self.y` **is changed to** `-3` **to speed up the direction of** `y` **for the ball.**

Question 4: Why do we make the `starts` **variable shuffle it's values?**

Question 5: What happens if I change the speed of `y`**? Implement your answer in your code, run it, and discuss what happens.**

31

AOLME

LESSON 6:

MOVEMENT AND COLLISION DETECTION

Question 6: The following page shows what your code should look like at this point in time. Run your own code with the new inputs we just discussed, and describe in detail what is happening in the space below. If you run into any errors, describe them and discuss how you might fix them.

AOLME

LESSON 6:

MOVEMENT AND COLLISION DETECTION

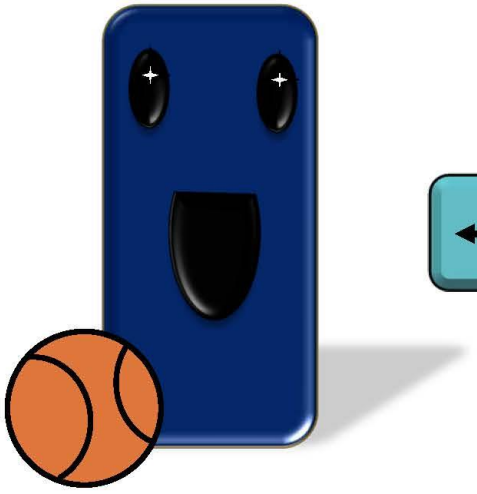
```
from tkinter import *
import random
import time

tk = Tk()
tk.title("Paddle Battle!!")
tk.resizable(0, 0)
tk.wm_attributes("-topmost", 1)
canvas = Canvas(tk, width=500, height=500)
canvas.pack()
tk.update()

class Ball:
    def __init__(self, canvas, color):
        self.canvas = canvas
        self.id = canvas.create_oval(10, 10, 25, 25, fill=color)
        self.canvas.move(self.id, 250, 250)
        starts = [-3, -2, -1, 1, 2, 3] # Lesson 6 code
        random.shuffle(starts) # Lesson 6 code
        self.x = starts[0] # Lesson 6 code
        self.y = -3 # Lesson 6 code
        self.canvas_height = self.canvas.winfo_height() # Lesson 6 code
        # Question 2 Code

    def draw(self):
        self.canvas.move(self.id, self.x, self.y) # Lesson 6 code
        pos = self.canvas.coords(self.id) # Lesson 6 code
        if pos[1] <= 0: # Lesson 6 code
            self.y = 1 # Lesson 6 code
        if pos[3] >= self.canvas_height: # Lesson 6 code
            self.y = -1 # Lesson 6 code
        # Question 2 Code
        # Question 2 Code
        # Question 2 Code
        # Question 2 Code

class Paddle:
    def __init__(self, canvas, color):
        self.canvas = canvas
        self.id = canvas.create_rectangle(0, 0, 100, 10, fill=color)
        self.canvas.move(self.id, 200, 300)

    def draw(self):
        pass

ball = Ball(canvas, 'red')
Paddle = Paddle(canvas, 'blue')

while 1:
    ball.draw()
    paddle.draw()
    tk.update_idletasks()
    tk.update()
    time.sleep(0.01)
```

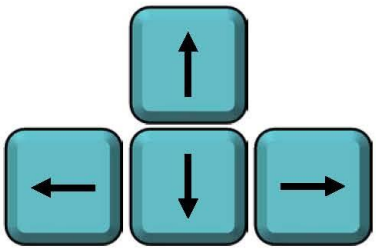
AOLME

LESSON 7:

ADDING MOVEMENT WITH KEYBOARD INPUTS

Now that we've created Mr. Bounce and the Paddle, we're ready to introduce some keyboard movements. We have Mr. Bounce moving along the screen, but the Paddle does not move. Right now, all the Paddle does is stay in one spot. In order to make this into the begins of a game, let's add some movement controls to the Paddle using the keyboard! Let's run our sample code and see what happens!





34

AOLME

LESSON 7:

ADDING MOVEMENT WITH KEYBOARD INPUTS

Before we began adding in the code to make our paddle move, let's first go line by line through the new code. We will be dealing mostly with the Paddle class code. Below is what your Paddle class should look like so far.

```
1.  class Paddle:
2.      def __init__(self, canvas, color):
3.          self.canvas = canvas
4.          self.id = canvas.create_rectangle(0, 0, 100, 10, fill=color)
5.          self.canvas.move(self.id, 200, 300)
6.
7.      def draw(self):
8.          pass
```

This first thing we have to do is create 2 new functions inside our Paddle class. These functions will be where we tell the paddle to move left or right. These new functions will be added after the draw function.

Line #	Code	Explanation
10	`def left(self, evt):`	This line is the start of our `left` function. The `evt` inside the parentheses is a parameter that indicates there will be some sort of event or action that will make this function work. In our case, we want this function to run when we hit the left arrow key.
11	`self.x = -2`	This line sets the variable `x` to `-2`, which means when we click the left arrow key, the paddle will move `2` pixels to the left.
13	`def right(self, evt):`	This line is the same as line 10, but in this case, this function will run when we hit the right arrow key.
14	`self.x = 2`	Like line 11 above, this line sets variable `x` to `2`, which moves the paddle `2` pixels to the right.

35

AOLME

LESSON 7:

ADDING MOVEMENT WITH KEYBOARD INPUTS

So, we've added our 2 new functions, but wait! We're using a variable that doesn't exist in our Paddle class!

Question 1: What variable are we using that we haven't created yet?

Question 2: How would you create your variable, and where would it go in your Paddle code? Write your code here, and tell me where you think it should go. Run your code when you have completed this question and describe what happens.

Question 3: We are also missing another variable that tells us the value of the width of the window. Create this variable and tell me where it should go. Hint: There is a similar variable that we created in the Ball class

AOLME

LESSON 7:

ADDING MOVEMENT WITH KEYBOARD INPUTS

Now that we have created our 2 new functions and variables, we are now ready to introduce movement through a keyboard event. We want the paddle to move from left to right without leaving the window. You already know how to determine whether your object has hit a wall in the window, so let's add the code you need to your Paddle class. Take a good look at the draw function in the Ball class below.

```
class Ball:
    def __init__(self, canvas, color):
        self.canvas = canvas
        self.id = canvas.create_oval(10, 10, 25, 25, fill=color)
        self.canvas.move(self.id, 250, 250)
        starts = [-3, -2, -1, 1, 2, 3]
        random.shuffle(starts)
        self.x = starts[0]
        self.y = -3
        self.canvas_height = self.canvas.winfo_height()
        # Question 2 Code, Lesson 6

    def draw(self):
        self.canvas.move(self.id, self.x, self.y)
        pos = self.canvas.coords(self.id)
        if pos[1] <= 0:
            self.y = 1
        if pos[3] >= self.canvas_height:
            self.y = -1
        # Question 2 Code, Lesson 6
        # Question 2 Code, Lesson 6
        # Question 2 Code, Lesson 6
        # Question 2 Code, Lesson 6
```

Question 4: We only need to verify whether the paddle has hit the left or right wall, so what code would you need to add to your Paddle draw function? Hint, the answer to Question 2 from Lesson 6 will be helpful here. Write your code, run your program, and describe what happens. If you run into errors, describe those as well.

37

AOLME

LESSON 7:

ADDING MOVEMENT WITH KEYBOARD INPUTS

Below is what your Paddle class should look like so far. Lines in blue are lines of code added so far.

```
1.  class Paddle:
2.      def __init__(self, canvas, color):
3.          self.canvas = canvas
4.          self.id = canvas.create_rectangle(0, 0, 100, 10, fill=color)
5.          self.canvas.move(self.id, 200, 300)
6.          # Question 2 Code, Lesson 7
7.          # Question 3 Code, Lesson 7
8.
9.      def draw(self):
10.         # Question 4 Code, Lesson 7
11.         # Question 4 Code, Lesson 7
12.         # Question 4 Code, Lesson 7
13.         # Question 4 Code, Lesson 7
14.
15.     def left(self, evt): # Lesson 7 Code
16.         self.x = -2 # Lesson 7 Code
17.
18.     def right(self, evt): # Lesson 7 Code
19.         self.x = 2 # Lesson 7 Code
```

The last thing we need to do is 'bind' the new functions we created to the left and right arrow keys. Binding in Python is where we make our program run a function when a particular key is pressed on the keyboard. You can even bind functions with mouse clicks. To bind our functions with a keyboard event we will use the `bind_all` function. These lines will go at the end of your Paddle `__init__` function.

Line #	Code	Explanation
8	`self.canvas.bind_all('<KeyPress-Left>', self.left)`	This line binds the `'<KeyPress-Left>'` event to our `left` function we created early in our Paddle class.
9	`self.canvas.bind_all('<KeyPress-Right>', self.right)`	This line binds the `'<KeyPress-Right>'` event to our `right` function we created early in our Paddle class.

38

AOLME

LESSON 7:

ADDING MOVEMENT WITH KEYBOARD INPUTS

Below, and on the next page is what your code should now look like.

```
from tkinter import *
import random
import time

tk = Tk()
tk.title("Paddle Battle!!")
tk.resizable(0, 0)
tk.wm_attributes("-topmost", 1)
canvas = Canvas(tk, width=500, height=500)
canvas.pack()
tk.update()

class Ball:
    def __init__(self, canvas, color):
        self.canvas = canvas
        self.id = canvas.create_oval(10, 10, 25, 25, fill=color)
        self.canvas.move(self.id, 250, 250)
        starts = [-3, -2, -1, 1, 2, 3]
        random.shuffle(starts)
        self.x = starts[0]
        self.y = -3
        self.canvas_height = self.canvas.winfo_height()
        # Question 2 Code, Lesson 6

    def draw(self):
        self.canvas.move(self.id, self.x, self.y)
        pos = self.canvas.coords(self.id)
        if pos[1] <= 0:
            self.y = 1
        if pos[3] >= self.canvas_height:
            self.y = -1
        # Question 2 Code, Lesson 6
        # Question 2 Code, Lesson 6
        # Question 2 Code, Lesson 6
        # Question 2 Code, Lesson 6
```

39

AOLME

LESSON 7:

ADDING MOVEMENT WITH KEYBOARD INPUTS

Below, and on the next page is what your code should now look like.

```
class Paddle:
     def __init__(self, canvas, color):
          self.canvas = canvas
          self.id = canvas.create_rectangle(0, 0, 100, 10, fill=color)
          self.canvas.move(self.id, 200, 300)
          # Question 2 Code, Lesson 7
          # Question 3 Code, Lesson 7

     def draw(self):
          # Question 4 Code, Lesson 7
          # Question 4 Code, Lesson 7
          # Question 4 Code, Lesson 7
          # Question 4 Code, Lesson 7
          # Question 4 Code, Lesson 7
          # Question 4 Code, Lesson 7

     def left(self, evt): # Lesson 7 Code
          self.x = -2 # Lesson 7 Code

     def right(self, evt): # Lesson 7 Code
          self.x = 2 # Lesson 7 Code

ball = Ball(canvas, 'red')
Paddle = Paddle(canvas, 'blue')

while 1:
    ball.draw()
    paddle.draw()
    tk.update_idletasks()
    tk.update()
    time.sleep(0.01)
```

Question 5: Run your code now and describe what happens when the ball hits the paddle.

40

AOLME

LESSON 8:

ON YOUR OWN

Programming Question 1:

In Question 5 from Lesson 7, you found out that the ball goes straight through the paddle. This is a problem. We can't have a Paddle Battle! game without a paddle that bounces the ball back. We've already gone through how to do collision detection with the window walls. Use what you have learned about detection to find out when the ball hits the paddle.

Programming Question 2:

Right now, all the ball does is bounce around when it hits a wall or the paddle. This isn't much of a game if you can't lose. What we need to do is make one of the walls a bad wall to hit. Use your knowledge of collision detection to make the bottom of your window the wall that ends the game.

Programming Question 3:

Now that you have figured out how to make the ball bounce away from the paddle, let's make this game interesting. What line would you need to change in order to make the balls speed go faster once it bounces away from hitting the paddle?

41

CONCLUSION

You've completed your first ever Python game. We hope you had fun! You now have the tools to create even more exciting games like Asteroids or even a nifty Pacman game! The sky is the limit! Remember to always experiment with your code and try new things. And most importantly, remember that no error is a bad error. You can always learn amazing things from mistakes made! Happy Programming

Sincerely,

Mr. Bounce and the Paddle

REFERENCES

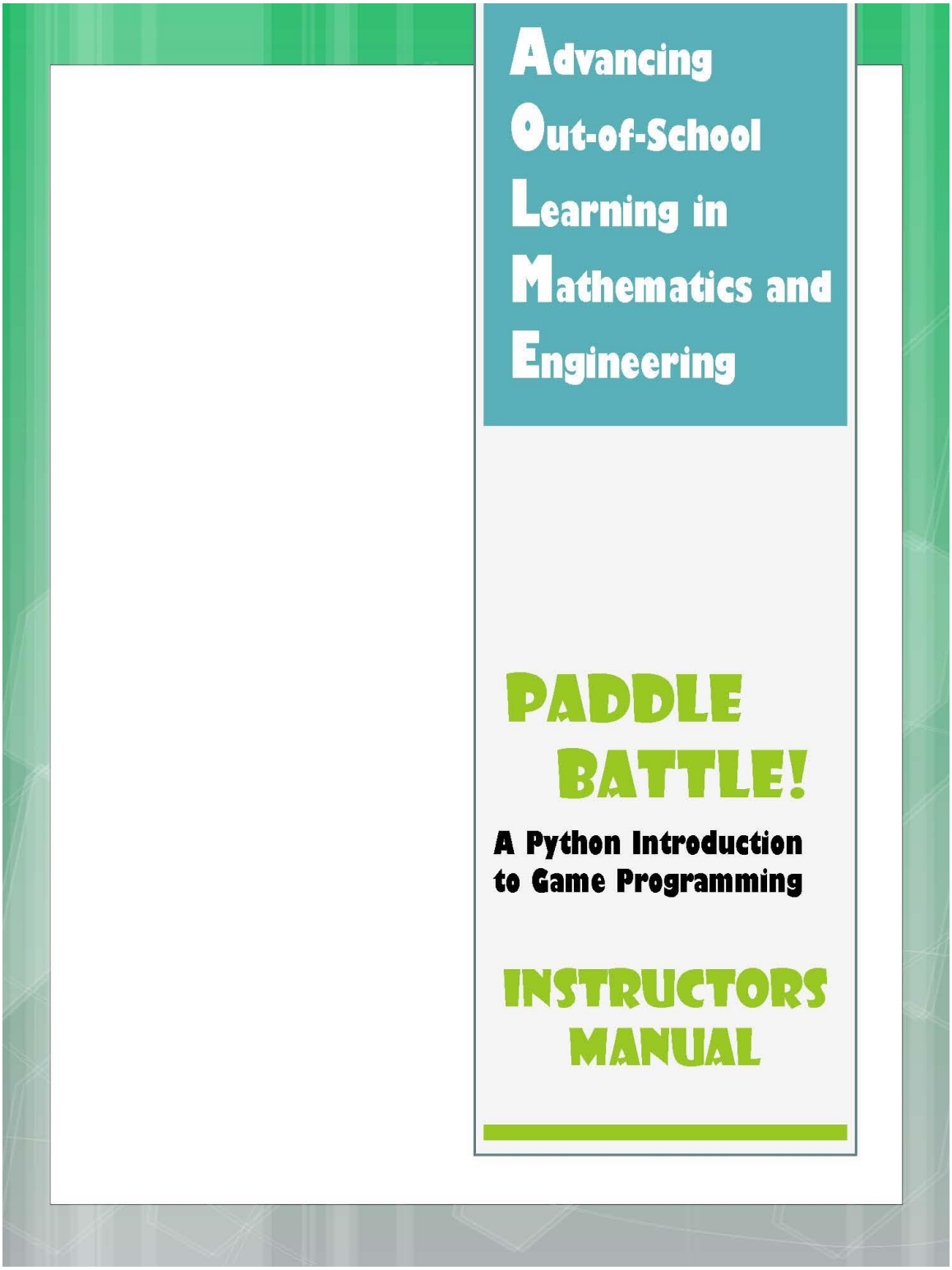
Code for the Paddle Battle! Game adapted from:

- **Briggs, J., *Python for Kids: A Playful Introduction to Programming*. No Starch Press, 2013.**

Code explanations and definitions can be found at:

- **Shipman, J., "Tkinter 8.5 reference: a GUI for Python" New Mexico Tech. [http://infohost.nmt.edu/tcc/help/pubs/tkinter/web/index.html].**

APPENDIX B: The Paddle Battle! Instructor's Manual



AOLME

TABLE OF CONTENTS

INTRODUCTION

This answer manual provides sample answers to the questions asked in the Paddle Battle! workbook. Each lesson starts off with a piece of code that the instructor must run and then explain line by line. Some lessons do not provide any code and rely solely on what the students have learned thus far. At the beginning of each lesson, the newly added code that will be provided in the lesson to come is run so that the students can see how their program progresses through each step.

LESSON 1 CODE:

This lesson should take less than 15 minutes to accomplish.

Show them the game and how it works, then let them answer the lesson questions accordingly.

```
from tkinter import *
import random
import time

tk = Tk()
tk.title("Paddle Battle!!")
tk.resizable(0, 0)
tk.wm_attributes("-topmost", 1)
canvas = Canvas(tk, width=500, height=500)
canvas.pack()
tk.update()

class Ball:
    def __init__(self, canvas, paddle, color):
        self.canvas = canvas
        self.canvas = paddle
        self.id = canvas.create_oval(10, 10, 25, 25, fill=color)
        self.canvas.move(self.id, 250, 250)
        starts = [-3, -2, -1, 1, 2, 3]
        random.shuffle(starts)
        self.x = starts[0]
        self.y = -3
        self.canvas_height = self.canvas.winfo_height()
        self.canvas_width = self.canvas.winfo_width()
        self.hit_bottom = False

    def some_function(self, pos):
        paddle_pos = self.canvas.coords(self.paddle.id)
        if pos[2] >= paddle_pos[0] and pos[0] <= paddle_pos[2]:
            if pos[3] >= paddle_pos[1] and pos[3] <= paddle_pos[3]:
                self.x += self.paddle.x
        return False

    def draw(self):
        self.canvas.move(self.id, self.x, self.y)
        pos = self.canvas.coords(self.id)
        if pos[1] <= 0:
            self.y = 1
        if pos[3] >= self.canvas_height:
            self.hit_bottom = True
        if self.some_function(pos) == True:
            self.y = -1
        if pos[0] <= 0:
            self.y = 1
        if pos[2] >= self.canvas_width:
            self.y = -1
```

3

LESSON 1 CODE (CONT.):

```
class Paddle:
    def __init__(self, canvas, color):
        self.canvas = canvas
        self.id = canvas.create_rectangle(0, 0, 100, 10, fill=color)
        self.canvas.move(self.id, 200, 300)
        self.x = 0
        self.canvas_width = self.canvas.winfo_width()
        self.canvas.bind_all('<KeyPress-Left>', self.left)
        self.canvas.bind_all('<KeyPress-Right>', self.right)

    def draw(self):
        self.canvas.move(self.id, self.x, 0)
        pos = self.canvas.coords(self.id)
        if pos[0] <= 0:
            self.y = 0
        if pos[2] >= self.canvas_width:
            self.y = 0

    def left(self, evt):
       self.x = -2

    def right(self, evt):
       self.x = 2

ball = Ball(canvas, paddle, 'red')
Paddle = Paddle(canvas, 'blue')

while 1:
    if ball.hit_bottom == False:
        ball.draw()
        paddle.draw()
    tk.update_idletasks()
    tk.update()
    time.sleep(0.01)
```

LESSON 1 QUESTIONS AND ANSWERS

- **Question 1: What does the ball do?**

  Sample Answer: The ball bounces around the screen, and also bounces off the paddle.

- **Question 2: What does the paddle do?**

  Sample Answer: The paddle moves left to right using the arrow keys. If the ball hits the paddle, the ball bounces away.

- **Question 3: What happens when the ball hits the sides or the top of the window?**

  Sample Answer: The ball bounces away from it.

- **Question 4: What happens when the ball hits the paddle?**

  Sample Answer: The ball bounces away from it.

- **Question 5: What happens if the ball hits the bottom of the window?**

  Sample Answer: If the ball hits the bottom of the window, then the game ends.

A0LME

LESSON 2 CODE:

This lesson should take less than 30 minutes to accomplish.

Show them the what happens when the code runs, explain the lines step by step in following with the workbook, and then have them answer the lesson questions accordingly.

```
from tkinter import *
import random
import time

tk = Tk()
tk.title("Paddle Battle!!")
tk.resizable(0, 0)
tk.wm_attributes("-topmost", 1)
canvas = Canvas(tk, width=500, height=500)
canvas.pack()
tk.update()
```

LESSON 2 QUESTIONS AND ANSWERS:

- **Question 1: What happens when we run this code?**

  Sample Answer: A window appears with the title Paddle Battle!!

- **Question 2: Let's change the width of the canvas to 30, and our height to 300, and then run our code. Describe what happens**

  In order to answer this question, the student needs to change the parameters in the following line:

```
canvas = Canvas(tk, width=500, height=500)
```

  Sample Answer: The window gets very small.

- **Question 3: Remove the line that says "canvas.pack()", and then run your code. Describe what happens.**

  Sample Answer: The window isn't the right size.

LESSON 3 CODE:

This lesson should take less than 45 minutes to accomplish. This lesson introduces a lot of material that is unfamiliar, so explanations may take longer.

Show them the what happens when the code runs, explain the lines step by step in following with the workbook, and them have them answer the lesson questions accordingly.

```
from tkinter import *
import random
import time

tk = Tk()
tk.title("Paddle Battle!!")
tk.resizable(0, 0)
tk.wm_attributes("-topmost", 1)
canvas = Canvas(tk, width=500, height=500)
canvas.pack()
tk.update()

#The code below is the new code being added in Lesson 3 to the end of
Lesson 2's code.
class Ball:
    def __init__(self, canvas, color):
        self.canvas = canvas
        self.id = canvas.create_oval(10, 10, 25, 25, fill=color)
        self.canvas.move(self.id, 250, 250)

    def draw(self):
        pass

ball = Ball(canvas, 'red')

while 1:
    tk.update_idletasks()
    tk.update()
    time.sleep(0.01)
```

LESSON 3 QUESTIONS AND ANSWERS:

- **Question 1: If you get errors when you compiled, please list some of them here and tell me why you think they happened.**

- **Question 2: Mr. Bounce loves the color 'red', but today he wants to wear green. What line of code would you change to make him green? Change the parameter and run your code to describe what happens.**

  In order to answer this question, the student needs to change the parameters in the following line from red to green:

  ```
  ball = Ball(canvas, 'red')
  ```

  Sample Answer: The ball turns green.

- **Question 3: What if Mr. Bounce wants to be bigger in the game? What line of code would you have to change in order to make your ball bigger? Change the parameter and run your code to describe what happens.**

  In order to answer this question, the student needs to change the parameters in the following line of code. Do not tell them which parameters they need to fix, let them try changing all of them until they find the right ones.

  ```
  self.id = canvas.create_oval(10, 10, 25, 25, fill=color)
  ```

  Sample Answer: The parameters that need to be changed are the x and y coordinates for the bottom of the oval (i.e, (25, 25)).

- **Question 4: What happens if we change the position of where our ball is drawn to 450, 200? Change the parameter and run your code to describe what happens.**

  In order to answer this question, the student needs to change the parameters in the following line:

  ```
  self.canvas.move(self.id, 250, 250)
  ```

  Sample Answer: The ball is drawn closer to the right side of the window.

LESSON 4 CODE:

This lesson should take less than 45 minutes to accomplish. This lesson does not introduce anything new, however, the students are expected to write a large portion of this code on their own using the knowledge they have gained from the previous lesson.

Show them the what happens when the code runs, explain the lines step by step in following with the workbook, and them have them answer the lesson questions accordingly.

```
from tkinter import *
import random
import time

tk = Tk()
tk.title("Paddle Battle!!")
tk.resizable(0, 0)
tk.wm_attributes("-topmost", 1)
canvas = Canvas(tk, width=500, height=500)
canvas.pack()
tk.update()

class Ball:
    def __init__(self, canvas, color):
        self.canvas = canvas
        self.id = canvas.create_oval(10, 10, 25, 25, fill=color)
        self.canvas.move(self.id, 250, 250)

    def draw(self):
        pass

class Paddle: # Lesson 4 code
    def __init__(self, canvas, color): # Lesson 4 code
        self.canvas = canvas # Lesson 4 code
        self.id = canvas.create_rectangle(0, 0, 100, 10, fill=color) # Lesson 4 code
        self.canvas.move(self.id, 200, 300) # Lesson 4 code

    def draw(self): # Lesson 4 code
        pass # Lesson 4 code

ball = Ball(canvas, 'red')
Paddle = Paddle(canvas, 'blue') # Lesson 4 code

while 1:
    tk.update_idletasks()
    tk.update()
    time.sleep(0.01)
```

LESSON 4 QUESTIONS AND ANSWERS:

- **Question 1: If we want to create a rectangle instead of an oval, what line do you think we should change? Write the line and show the changes.**

  Sample answer: The line that should change is the line below:

  ```
  self.id = canvas.create_oval(10, 10, 25, 25, fill=color),
  ```
  to,
  ```
  self.id = canvas.create_rectangle(0, 0, 100, 10, fill=color)
  ```
  (Let them decide on their own parameters.)

- **Question 2: We should also position our new paddle in a different location to make sure we aren't covering up our Ball. What line should we change if we want to change the position of our Paddle? Write the line and show the changes.**

  In order to answer this question, the student needs to change the parameters in the following line:

  ```
  self.canvas.move(self.id, 250, 250)
  ```
  for the ball, to
  ```
  self.canvas.move(self.id, 200, 300)
  ```
  for the paddle.

- **Question 3: What happens if we run the code as it is now? Hint: We ran the Ball class code once when we only had these lines. Do you remember what happened? Run your code and describe what happens.**

  Sample Answer: The paddle does not appear on the window.

- **Question 4: What line are we missing in order to make the paddle appear on our canvas? Hint: You might find an example of it in Lesson 3. Add the line to your code and then run your program to describe what happens.**

  In order to answer this question, the student needs to add the following line to their code:

  ```
  paddle = Paddle(canvas, 'blue')
  ```

LESSON 5 CODE:

This lesson should take less than 20 minutes to accomplish. This lesson introduces the first steps in making their object move. There are only 3 lines that need to be introduced and discussed here.

Show them the what happens when the code runs, explain the lines step by step in following with the workbook, and them have them answer the lesson questions accordingly.

```
from tkinter import *
import random
import time

tk = Tk()
tk.title("Paddle Battle!!")
tk.resizable(0, 0)
tk.wm_attributes("-topmost", 1)
canvas = Canvas(tk, width=500, height=500)
canvas.pack()
tk.update()

class Ball:
    def __init__(self, canvas, color):
        self.canvas = canvas
        self.id = canvas.create_oval(10, 10, 25, 25, fill=color)
        self.canvas.move(self.id, 250, 250)

    def draw(self):
        self.canvas.move(self.id, 0, -1) # Lesson 5 code

class Paddle:
    def __init__(self, canvas, color):
        self.canvas = canvas
        self.id = canvas.create_rectangle(0, 0, 100, 10, fill=color)
        self.canvas.move(self.id, 200, 300)

    def draw(self):
        pass

ball = Ball(canvas, 'red')
Paddle = Paddle(canvas, 'blue')

while 1:
    ball.draw() # Lesson 5 code
    paddle.draw() # Lesson 5 code
    tk.update_idletasks()
    tk.update()
    time.sleep(0.01)
```

LESSON 5 QUESTIONS AND ANSWERS:

- **Question 1: What parameter would you have to change to make the ball go down? Change the parameter and run your code to describe what happens.**

  In order to answer this question, the student needs to change the parameters in the following line:

  ```
  self.canvas.move(self.id, 0, -1), to,
  self.canvas.move(self.id, 0, 1)
  ```

- **Question 2: What parameter would you change to make the ball go left or right? Change the parameter and run your code to describe what happens.**

  In order to answer this question, the student needs to change the parameters in the following line:

  `self.canvas.move(self.id, -1, 0)` **if I want to make the ball go left, and** `self.canvas.move(self.id, 1, 0)` **if I want to make the ball go right. As an added exercise, have the students see what happens when they have a non-zero value in both parameters (ball goes diagonal).**

- **Question 3: What happens if I change the speed at which the ball moves from 1 pixel to 25 pixels? Change the parameter and run your code to describe what happens.**

  In order to answer this question, the student needs to change the parameters in the following line:

  ```
  self.canvas.move(self.id, 0, -1), to,
  self.canvas.move(self.id, 0, -25)
  ```

  Sample Answer: The ball moves up very quickly.

- **Question 4: What happens if you forget to include the line 'ball.draw()' in your main function? Change the parameter and run your code to describe what happens.**

  Sample Answer: The ball does not move at all.

13

LESSON 6 CODE:

This lesson should take less than 60 minutes to accomplish. This lesson introduces actual animation, collision detection, speed control, and movements with angles. The students are also expected program a portion of this code on their own.

Show them the what happens when the code runs, explain the lines step by step in following with the workbook, and them have them answer the lesson questions accordingly.

```
from tkinter import *
import random
import time

tk = Tk()
tk.title("Paddle Battle!!")
tk.resizable(0, 0)
tk.wm_attributes("-topmost", 1)
canvas = Canvas(tk, width=500, height=500)
canvas.pack()
tk.update()

class Ball:
    def __init__(self, canvas, color):
        self.canvas = canvas
        self.id = canvas.create_oval(10, 10, 25, 25, fill=color)
        self.canvas.move(self.id, 250, 250)
        starts = [-3, -2, -1, 1, 2, 3] # Lesson 6 code
        random.shuffle(starts) # Lesson 6 code
        self.x = starts[0] # Lesson 6 code
        self.y = -3 # Lesson 6 code
        self.canvas_height = self.canvas.winfo_height() # Lesson 6 code
        self.canvas_width = self.canvas.winfo_width() # Lesson 6 code, Question 2

    def draw(self):
        self.canvas.move(self.id, self.x, self.y) # Lesson 6 code
        pos = self.canvas.coords(self.id) # Lesson 6 code
        if pos[1] <= 0: # Lesson 6 code
            self.y = 1 # Lesson 6 code
        if pos[3] >= self.canvas_height: # Lesson 6 code
            self.y = -1 # Lesson 6 code
        if pos[0] <= 0: # Lesson 6 code, Question 2
            self.y = 1 # Lesson 6 code, Question 2
        if pos[2] >= self.canvas_width: # Lesson 6 code, Question 2
            self.y = -1 # Lesson 6 code, Question 2

class Paddle:
    def __init__(self, canvas, color):
        self.canvas = canvas
        self.id = canvas.create_rectangle(0, 0, 100, 10, fill=color)
        self.canvas.move(self.id, 200, 300)

    def draw(self):
        pass

ball = Ball(canvas, 'red')
Paddle = Paddle(canvas, 'blue')

while 1:
    ball.draw()
    paddle.draw()
    tk.update_idletasks()
    tk.update()
    time.sleep(0.01)
```

LESSON 6 QUESTIONS AND ANSWERS:

- **Question 1: What is the current height of your canvas?**

  In order to answer this question, the student needs to look at this line below:

  ```
  canvas = Canvas(tk, width=500, height=500)
  ```

  Sample Answer: The height of our canvas is 500 pixels.

- **Question 2: Right now we only have code that detects when the ball hits the top or bottom of the window. What code would we need to add to make sure that the left and right ends (x-axis) of the window are detected by the ball? Add the lines of code you think you need, run your code, and describe what happens. If you run into errors, describe those as well.**

  Sample Answer:

  ```
  if pos[0] <= 0:
       self.y = 1
  if pos[2] >= self.canvas_width:
       self.y = -1
  ```

- **Question 3: What do you think we need to do in order to get the ball to move around the window, instead of just up and down? Make a guess.**

  Sample Answer: Make the ball start moving in a different direction.

- **Question 4: Why do we make the `starts` variable shuffle it's values?**

  Sample Answer: So that the angle and direction of the ball is different every time we start the game.

LESSON 6 QUESTIONS AND ANSWERS (CONT.):

- **Question 5: What happens if we change the value of y? Implement your answer in your code, run it, and discuss what happens.**

  In order to answer this question, the student needs to change the parameters in the following line:

  ```
  self.y = -3
  ```

  Sample Answer: The ball speeds up.

- **Question 6: The following page shows what your code should look like at this point in time. Run your own code with the new inputs we just discussed, and describe in detail what is happening in the space below. If you run into any errors, describe them and discuss how you might fix them.**

LESSON 7 CODE:

This lesson should take less than 60 minutes to accomplish. This lesson introduces how to move an object using an event like pressing a key. Events are introduced here, as well as the students are expected to write a portion of the code using the knowledge they learned from lesson 6, question 2.

Show them the what happens when the code runs, explain the lines step by step in following with the workbook, and them have them answer the lesson questions accordingly.

```
from tkinter import *
import random
import time

tk = Tk()
tk.title("Paddle Battle!!")
tk.resizable(0, 0)
tk.wm_attributes("-topmost", 1)
canvas = Canvas(tk, width=500, height=500)
canvas.pack()
tk.update()

class Ball:
    def __init__(self, canvas, color):
        self.canvas = canvas
        self.id = canvas.create_oval(10, 10, 25, 25, fill=color)
        self.canvas.move(self.id, 250, 250)
        starts = [-3, -2, -1, 1, 2, 3]
        random.shuffle(starts)
        self.x = starts[0]
        self.y = -3
        self.canvas_height = self.canvas.winfo_height()
        self.canvas_width = self.canvas.winfo_width()

    def draw(self):
        self.canvas.move(self.id, self.x, self.y)
        pos = self.canvas.coords(self.id)
        if pos[1] <= 0:
            self.y = 1
        if pos[3] >= self.canvas_height:
            self.y = -1
        if pos[0] <= 0:
            self.y = 1
        if pos[2] >= self.canvas_width:
            self.y = -1
```

LESSON 7 CODE (CONT.):

```
class Paddle:
    def __init__(self, canvas, color):
        self.canvas = canvas
        self.id = canvas.create_rectangle(0, 0, 100, 10, fill=color)
        self.canvas.move(self.id, 200, 300)
        self.x = 0 # Lesson 7 code, Question 2
        self.canvas_width = self.canvas.winfo_width() # Lesson 7 code, Question 3
        self.canvas.bind_all('<KeyPress-Left>', self.left) # Lesson 7 code
        self.canvas.bind_all('<KeyPress-Right>', self.right) # Lesson 7 code

    def draw(self):
        self.canvas.move(self.id, self.x, 0) # Lesson 7 code, Question 4
        pos = self.canvas.coords(self.id) # Lesson 7 code, Question 4
        if pos[0] <= 0: # Lesson 7 code, Question 4
            self.y = 0 # Lesson 7 code, Question 4
        if pos[2] >= self.canvas_width: # Lesson 7 code, Question 4
            self.y = 0 # Lesson 7 code, Question 4

    def left(self, evt): # Lesson 7 code
       self.x = -2 # Lesson 7 code

    def right(self, evt): # Lesson 7 code
       self.x = 2 # Lesson 7 code

ball = Ball(canvas, 'red')
Paddle = Paddle(canvas, 'blue')

while 1:
    ball.draw()
    paddle.draw()
    tk.update_idletasks()
    tk.update()
    time.sleep(0.01)
```

AOLME

LESSON 7 QUESTIONS AND ANSWERS:

- **Question 1: What variable are we using that we haven't created yet?**

  Sample Answer: The variable that we are using that isn't defined yet for our Paddle class is the x variable.

- **Question 2: How would you create your variable, and where would it go in your Paddle code? Write your code here, and tell me where you think it should go. Run your code when you have completed this question and describe what happens.**

  Sample Answer: The variable should look like the following: `self.x = 0`**, and should be placed right after the following line in their code:**
  `self.canvas.move(self.id, 200, 300).`

- **Question 3: We are also missing another variable that tells us the value of the width of the window. Create this variable and tell me where it should go. Hint: There is a similar variable that we created in the Ball class**

  Sample Answer: The variable should look like the following: `self.canvas_width = self.canvas.winfo_width()`**, and should be placed right after the variable x that they just created in Question 2.**

Question 4: We only need to verify whether the paddle has hit the left or right wall, so what code would you need to add to your Paddle draw function? Hint, the answer to Question 2 from Lesson 6 will be helpful here. Write your code, run your program, and describe what happens. If you run into errors, describe those as well.

Sample Answer: The code that should be added to the draw function is the following set of lines using the variables they have created in the previous questions:

```
def draw(self):
        self.canvas.move(self.id, self.x, 0)
        pos = self.canvas.coords(self.id)
        if pos[0] <= 0:
             self.y = 0
        if pos[2] >= self.canvas_width:
             self.y = 0
```

19

LESSON 7 QUESTIONS AND ANSWERS (CONT.):

- **Question 5: Run your code now and describe what happens when the ball hits the paddle.**

  At this point, their ball should be bouncing around the screen, and they should be able to move their paddle from left to right using the keyboard. One key thing they should notice is that their ball passes through their paddle instead of bouncing away from it.

LESSON 8 CODE, QUESTIONS, AND ANSWERS:

- **Programming Question 1:**

  In Question 5 from Lesson 7, you found out that the ball goes straight through the paddle. This is a problem. We can't have a Paddle Battle! game without a paddle that bounces the ball back. We've already gone through how to do collision detection with the window walls. Use what you have learned about detection to find out when the ball hits the paddle.

  This question should take about 60 minutes to figure out. In order for the ball to acknowledge the paddle, the paddle must become a part of the Ball class, and change the line where they create the ball object:

```
class Ball:
    def __init__(self, canvas, paddle, color):
        self.canvas = canvas
        self.canvas = paddle
        self.id = canvas.create_oval(10, 10, 25, 25, fill=color)
        self.canvas.move(self.id, 250, 250)
        starts = [-3, -2, -1, 1, 2, 3]
        random.shuffle(starts)
        self.x = starts[0]
        self.y = -3
        self.canvas_height = self.canvas.winfo_height()
        self.canvas_width = self.canvas.winfo_width()

        ball = Ball(canvas, paddle, 'red')
```

They must create a function that figures out when the ball has hit the paddle:

```
def some_function(self, pos):
        paddle_pos = self.canvas.coords(self.paddle.id)
        if pos[2] >= paddle_pos[0] and pos[0] <= paddle_pos[2]:
            if pos[3] >= paddle_pos[1] and pos[3] <= paddle_pos[3]:
                return True
        return False
```

And they must add a line to their draw function in the Ball class that calls their new function so that when the ball hits the paddle, it moves away:

```
    def draw(self):
        self.canvas.move(self.id, self.x, self.y)
        pos = self.canvas.coords(self.id)
        if pos[1] <= 0:
            self.y = 1
        if pos[3] >= self.canvas_height:
            self.y = -1
        if self.some_function(pos) == True:
            self.y = -1
        if pos[0] <= 0:
            self.y = 1
        if pos[2] >= self.canvas_width:
            self.y = -1
```

LESSON 8 CODE, QUESTIONS, AND ANSWERS:

- **Programming Question 2:**

  Right now, all the ball does is bounce around when it hits a wall or the paddle. This isn't much of a game if you can't lose. What we need to do is make one of the walls a bad wall to hit. Use your knowledge of collision detection to make the bottom of your window the wall that ends the game.

  This question should take about 60 minutes to figure out. In order for the game to end, the student needs to add the `hit_bottom` **object variable to their Ball class:**

```
class Ball:
    def __init__(self, canvas, paddle, color):
        self.canvas = canvas
        self.canvas = paddle
        self.id = canvas.create_oval(10, 10, 25, 25, fill=color)
        self.canvas.move(self.id, 250, 250)
        starts = [-3, -2, -1, 1, 2, 3]
        random.shuffle(starts)
        self.x = starts[0]
        self.y = -3
        self.canvas_height = self.canvas.winfo_height()
        self.canvas_width = self.canvas.winfo_width()
        self.hit_bottom = False
```

They need to change the main loop so that the game is constantly checking to see if the ball has hit the bottom:

```
while 1:
    if ball.hit_bottom == False:
         ball.draw()
         paddle.draw()
    tk.update_idletasks()
    tk.update()
    time.sleep(0.01)
```

And they must edit a line in their draw function in the Ball so it can detect when the ball hits the bottom, so it can end the game:

```
    def draw(self):
        self.canvas.move(self.id, self.x, self.y)
        pos = self.canvas.coords(self.id)
        if pos[1] <= 0:
             self.y = 1
        if pos[3] >= self.canvas_height:
             self.hit_bottom = True
        if self.some_function(pos) == True:
             self.y = -1
        if pos[0] <= 0:
             self.y = 1
        if pos[2] >= self.canvas_width:
             self.y = -1
```

22

LESSON 8 CODE, QUESTIONS, AND ANSWERS:

- **Programming Question 3:**

  Now that you have figured out how to make the ball bounce away from the paddle, let's make this game interesting. What line would you need to change in order to make the balls speed go faster once it bounces away from hitting the paddle?

This question shouldn't take long to figure out, but figuring out where to put the line may take some time. This question should take about 30-60 minutes to figure out. In order to answer this question, the students must have first answered Programming Question 1.

In the following function they created, the student must change the line in red:

```
def some_function(self, pos):
        paddle_pos = self.canvas.coords(self.paddle.id)
        if pos[2] >= paddle_pos[0] and pos[0] <= paddle_pos[2]:
            if pos[3] >= paddle_pos[1] and pos[3] <= paddle_pos[3]:
                return True
        return False
```

To,

```
        self.x += self.paddle.x
```

This allows us to add the value of the `x` variable of the paddle, to the value of the `x` variable of the ball. So if the ball is travelling at a speed of `1`, and it hits the paddle, which is travelling at a speed of `3`, then the ball will then bounce away at a speed of `4`.

REFERENCES

[1] Advancing Out-of-School Learning in Mathematics and Engineering. [http://aolme.unm.edu/]. 2013.

[2] deHaan, J., "5 best iPad apps to teach programming." *Technology with Intention.* July 2012. [http://www.techwithintent.com/2012/07/5-best-ipad-apps-to-teach-programming/].

[3] "Daisy the Dinosaur." *Mind Leap: Education Apps for Kids.* [http://www.mindleaptech.com/apps/daisy-the-dinosaur/].

[4] "What is Alice?" *Alice: An Educational Software that teaches students computer programming in a 3D environment.* [http://www.alice.org/index.php?page=what_is_alice/what_is_alice].

[5] Alice. [http://www.alice.org/index.php].

[6] Cooper, S., Dann, W., Pausch, R., "Alice: A 3-D Tool For Introductory Programming Concepts." *Technical Report. Stanford University.* [http://www.stanford.edu/~coopers/alice/ccscne00.PDF].

[7] Resnick, M., Maloney, J., Monroy-Hernandez, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., and Kafai, J., "Scratch: Programming For All". *Journal Article. Communications of the ACM, Vol. 52, No. 11*. November 2009.

[8] "Computer Programming/Hello world." [http://en.wikibooks.org/wiki/Computer_Programming/Hello_world#Scratch].

[9] Franklin, D., et. al, "Assesment of Computer Science Learning in a Scratch-Based Outreach Program." *Technical Report*. [http://www.cs.ucsb.edu/~franklin/cv/pubs/sigcse13at.pdf].

[10] Briggs, J., *Python for Kids: A Playful Introduction to Programming*. No Starch Press, 2013.

[11] Cooper, S., Dann, W., and Pausch, R., "Teaching Objects-first In Introductory Computer Science". *Proceeding. 34th SIGCSE Technical Symposium on Computer Science Education.* January 2003. http://portal.acm.org/citation.cfm?id=611966.

[12] Dourish, P., "Seeking a Foundation for Context-Aware Computing". *Journal Article. Human Computer Interaction. 2001.* http://www.dourish.com/embodied/essay.pdf.

[13] Fernaeus, Y. and Tholander, J., "Collaborative Computation On The Floor". *Technical Report. DSV, Stockton University, Sweden.* 2003. http://www.lkl.ac.uk/kscope/weblabs/papers/FloorProgramming_Fernaeus_Tholander.pdf.

[14] Mor, Y., and, Tholander, J. and Holmberg, J., "Designing For Cross-Cultural Web-Based Knowledge Building". *Technical Report. The 10th Computer Supported Collaborative Learning Conference.* June 2005. http://www.lkl.ac.uk/kscope/weblabs/papers/cscl-2005.pdf.

[15] Moskal, B. Lurie, D., and Cooper, S., "Evaluating the Effectiveness of a New Instructional Approach". *Proceeding. 35th SIGCSE Technical Symposium on Computer Science Education.* March 2004. http://portal.acm.org/citation.cfm?id=971328.

[16] Celedon-Pattichis, S., Lopez Leiva, C., Pattichis, M., Llamocca, D., "An interdisciplinary collaboration between computer engineering and mathematics/bilingual education to develop a curriculum for underrepresented middle school students". *Cultural Studies of Science Education Issue 3, Vol. 8.* September 2013.

[17] National Research Council. (2011). "A framework for K-12 science education: Practices crosscutting concepts, and core ideas". Washington, DC: National Academy Press.