

University of Alabama in Huntsville

LOUIS

Honors Capstone Projects and Theses

Honors College

5-7-2021

Novel Gamification of Coding Education

Owen Seidler

Follow this and additional works at: <https://louis.uah.edu/honors-capstones>

Recommended Citation

Seidler, Owen, "Novel Gamification of Coding Education" (2021). *Honors Capstone Projects and Theses*. 567.

<https://louis.uah.edu/honors-capstones/567>

This Thesis is brought to you for free and open access by the Honors College at LOUIS. It has been accepted for inclusion in Honors Capstone Projects and Theses by an authorized administrator of LOUIS.

Novel Gamification of Coding Education

by

Owen Seidler

An Honors Capstone

submitted in partial fulfillment of the requirements

for the Honors Diploma

to

The Honors College

of

The University of Alabama in Huntsville

4/6/2021

Honors Capstone Director: Mr. Nicholas Diliberti

Computer Science Lecturer

Owen Seidler _____ May 6th, 2021 _____
Student (signature) Date

Nick Diliberti May 7 2021
Director (signature) Date

Department Chair (signature) Date

Honors College Dean (signature) Date



Honors College
Frank Franz Hall
+1 (256) 824-6450 (voice)
+1 (256) 824-7339 (fax)
honors@uah.edu

Honors Thesis Copyright Permission

This form must be signed by the student and submitted as a bound part of the thesis.

In presenting this thesis in partial fulfillment of the requirements for Honors Diploma or Certificate from The University of Alabama in Huntsville, I agree that the Library of this University shall make it freely available for inspection. I further agree that permission for extensive copying for scholarly purposes may be granted by my advisor or, in his/her absence, by the Chair of the Department, Director of the Program, or the Dean of the Honors College. It is also understood that due recognition shall be given to me and to The University of Alabama in Huntsville in any scholarly use which may be made of any material in this thesis.

Owen Seidler _____

Student Name (printed)

Owen Seidler _____

Student Signature

5/6/2021 _____

Date

Table of Contents

Abstract	2
Introduction	3
Rationale	4
Design	5
Resources	7
Process	8
Conclusion	11
Reference List	12
Appendix - Selection of Challenges	13

Abstract

As well as their most well-known purpose of entertainment, video games also offer a unique and interesting way to both teach and reinforce concepts to their players, using enjoyable and engaging interactions to encourage players to learn about and study topics that they otherwise might not have the motivation to occupy themselves with. To further explore this idea, I have created a proof-of-concept of a game to reinforce knowledge of the Java programming language, using fast-paced gameplay, similar to how one might study with flashcards, to encourage players to quickly and reliably recall how to write in, understand, and review Java code.

Introduction

Many scholarly studies have been performed which demonstrate the educational potential of video games. From those specifically designed to provide clear educational benefits, to commercial games not specifically created for such a reason, video games have firmly and undeniably shown their ability to teach concepts and aid in learning. However, the field is still young and new, with a lot of room to explore and synthesize new ideas. As part of my own academic career in studying entertainment computing, I sought to conceive of a game that could serve to combine my interests and expertise, and came to the idea to create a game that could be used to study a topic I know well from my college education: programming. Programming is complicated and features a lot of rules to learn and internalize, and I believe that a video game would be a good way to reinforce programming concepts to those trying to learn them, by increasing engagement with the act of writing code in order to improve recall and encourage the forming of connections.

In this paper, I will be covering my work on this project, from a look into the scholarly explorations of games as a tool for education, to the process I undertook in creating the project, to an assessment of my performance, and what I was able to, and not able to, accomplish. My hope is that this project will serve to further demonstrate the educational potential of games, specifically for a coding education, and to show a potential way one might choose to implement a more complete version of such a game.

Rationale

Although once it would have been controversial to assert such a fact, at this point in time it is well-established that video games have a strong potential to be used to assist in education and to develop mental skills. There are many games out there which are designed explicitly to teach, provide practice, or reinforce concepts to assist in education. An example can be found at the St. Michael's Hospital in Toronto, where staff members have access to a video game "that allows them to practise the critical steps necessary to treat diabetic ketoacidosis (DKA)" [1]. However, it is not just these kinds of games that can provide clear educational benefits, but commercial ones too. For instance, in one study, participants who had played the popular puzzle game Portal 2 for 8 hours had shown "significant increases" in tests on spatial skills [2]. Certainly it seems undeniable that using video games for educational purposes has significant scholarly potential.

Given this fact, the academic merit of the project should be clear. I believe that a game such as what I envisioned could assist computer science students, or anyone else attempting to learn programming, by making it fun and enjoyable to reinforce and internalize the necessary concepts, rules, and syntax to understand.

Design

When designing the game, I chose to take inspiration from the popular game series WarioWare, which pits the player against a gauntlet of short, speedy “microgames,” each with a different gameplay objective to be completed within a span of just a few seconds [3]. Taking inspiration from speed-learning methods such as flashcards, I decided to pair this style of gameplay with coding education by devising a series of fast, varied coding challenges to reinforce the concepts that a new student of programming would be currently learning, by asking them to recall and apply knowledge of coding in a short timespan. Ultimately, I came up with three ideas for coding “microgames:”

- “Fill In The Blanks,” in which an incomplete block of code is shown to the player with fillable blank spaces in the missing fields, and the player must complete the code according to the provided instructions.
- “Spot The Errors,” in which the code presented to the player has several errors present which must all be identified by clicking on them.
- “Question Time,” where the player must correctly answer multiple choice questions.

```

public class ClassExample {
    public static void main(String[] args) {

        int var1 = 5;
        int var2 = 3;

        for(int i = 1; i < 10; [ ]) { //increment i!
            var1 = var1 * i;
            var2 = var1 + var2;
        }

        [ ].println(var2); //print to console!

        var1 = blackBoxFunction(var2);

        while(var1 >= 0) {
            var--;
            if(var1 == 5) {
                [ ]; //exit the loop!
            }
        }
    }
}

```

Figure 1 - An example of “Fill In The Blanks.”

For each microgame, the player is placed under a limited amount of time, represented by the green bar on the bottom of the screen, which shrinks as time runs out until it disappears and the game ends. In WarioWare, these games would only last around a few seconds, but for this game, I decided to use a longer timer of 18 seconds, as solving problems of these kinds would be far too difficult with such a short timespan.

Linking each game is a brief period of respite on the main screen of the game, where the player can see how many microgames are awaiting them before the end of the play session, each one represented as a file on a computer screen, as well as the game's computer mascot, who will react positively or negatively based on the player's performance in the previous microgame. Although not every microgame was completed on time, in a more completed version of the program, the game would have also featured "boss microgames" with harder, longer challenges to test the player's programming abilities more thoroughly, and represented by black-and-red files.

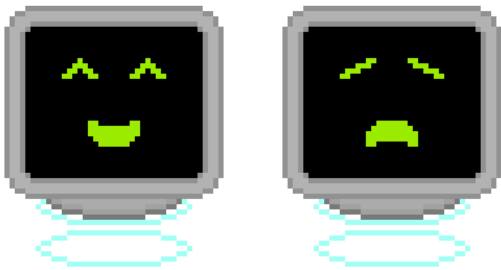


Figure 2 - The game's computer mascot.

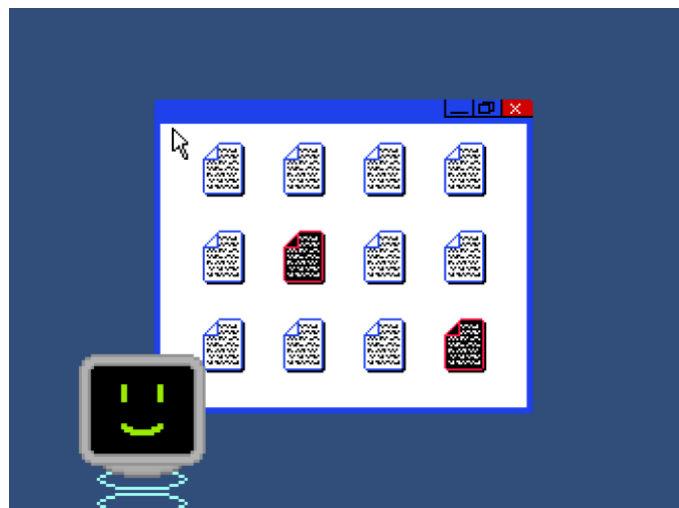


Figure 3 - The main game screen.

Resources

To create this game, I utilized several different programs, each for the creation of a different aspect or part of the program. The most important and central of these programs was Unity. Unity is a cross-platform game development engine that can be used to create both 3D and 2D games [4]. With Unity, one can easily create and edit a game world by creating and moving objects and attaching scripts or pre-created components in order to create behavior. Specifically, the version used was version 2019.4.19f1, which was used because it was the version I was already most familiar with.

For the art assets, I used Aseprite, a software used to create pixel art and animations [5]. This software was chosen for the creation of art for two reasons. First, I sought to create a computerized, digital aesthetic, and pixel art would suit this aesthetic well. Second, I am much more familiar with the creation of pixel art than any other visual art, making it the most obvious choice.

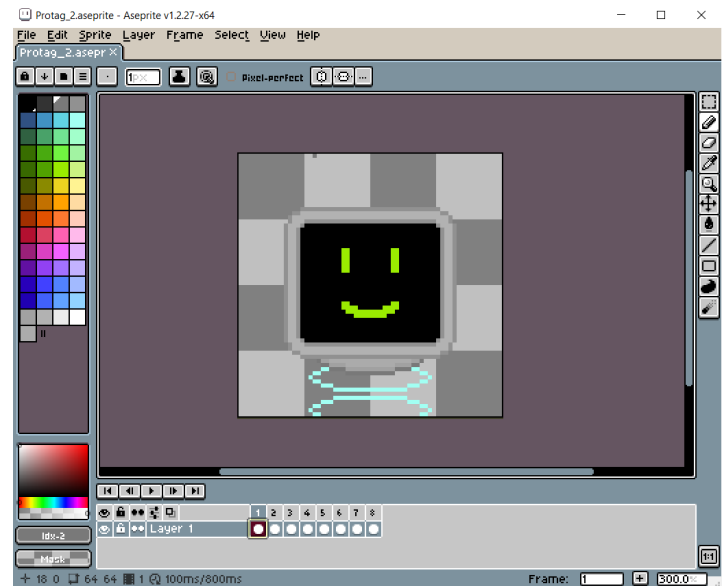


Figure 4 - A screenshot of Aseprite in use.

Finally, for music and sound, I used two different programs. For the music, I used a free chiptune music creation software known as FamiStudio [6], while for the sounds, I use a chiptune sound created called Bfxr [7]. Like the art, these programs were chosen for two reasons: because I felt that the chiptune sounds and music they produced would fit the digital aesthetic I was aiming for, and because of prior experience with these programs.

Process

Planning

To begin with, first I had to plan my path through the creation of this game, including deciding the aesthetic theme, the intended audience, and details of the gameplay. As mentioned in the Resources section, I decided to go with an aesthetic inspired by computers, including a file explorer window on a desktop as the main screen, and a computer character as the mascot, as it would fit well with programming being the concept that the game serves to reinforce, and my past experience with pixel art and 8-bit music would work well for this too. As for the intended audience, I initially struggled between deciding on focusing on experienced coders, like myself, or novice ones, as this decision would influence how I approached the details of the gameplay and the types of challenges I would create. Ultimately, I decided to focus on novice coders, as I knew that my general idea of using fast-paced challenges would work best with simpler challenges rather than the more complex ones that experienced coders would benefit from. This was also the point I chose to specifically focus on Java as the language the project would focus on, as it is an entry-level language for a lot of people, and also the language I am most familiar with. As for planning out the gameplay, I knew I had to come up with several ideas for different types of microgames that could each reinforce coding knowledge through a different mode of interaction. In the end, I came up with the three types of microgames mentioned earlier, each to serve a different purpose. “Fill in the Blanks,” for instance, serves to test whether the player can remember what to write to accomplish a specific task, and how to use correct syntax. “Spot the Errors,” on the other hand, asks the player to test their ability to parse an already completed block of code, and to read carefully in order to spot where typos and mistakes may lie, a skill that

is very important when actually coding. Finally, “Question Time” is an opportunity to directly ask the player questions to test more abstract concepts that direct interaction with code may not be able to address, such as whether they know all the data types, or how arrays are enumerated. With all my planning completed, it was time to begin setting up the project itself.

Setup

To begin, first I had to create a new Unity project, and create all the framework on which the actual game would be implemented, including the creation of a title screen, arranging placeholder objects in each scene to be replaced by actual functional objects later, and a couple basic scripts to allow the game to move from scene to scene.

This is also the phase during which asset creation began. To start out with, I prioritized the musical cue that would play before each microgame, and the music itself that would play during one. I knew that the flow between each microgame was a very important thing to get right, as it would help the player get in the right mindset and create the rhythmic feel of play that would help the player stay on track and in the moment while keeping the limited amount of time in mind. Then, I created a couple more musical cues, one for winning a microgame, and one for failing one, before moving on to sprite creation, where I created the file explorer window for the background, a file icon to serve as the indicator for each microgame, and the computer mascot, who I created because I wanted the game to have a personal “face” to it to give it some identity.

With the basic asset creation finished, the assets were then added to the Unity project itself. At this point, with the framework and assets, it was time to create the actual mechanics.

Implementing the Mechanics

With the framework established and assets added, mechanics had to be created for each microgame. Work began on “Fill In The Blanks” first: I created a placeholder block of code, and

created a few fillable blanks, each of which would register itself as “complete” when the user inputted the correct string. After filling all the blanks correctly, the game would exit and be marked as a success. This is also when I created the timer, which I would use for both other game types as well. At the bottom of the screen, a green bar will slowly shrink as time passes. After the bar fully disappears (around 18 seconds), the game will exit and be marked as a failure. By copying and pasting the bar into scenes for the other two microgames, this timer could easily be imported into those microgames as well, saving on resources.

Creating the mechanics for “Question Time” was fairly easy, as Unity has plenty of pre-existing button functionality, meaning all I had to do was set four up, have clicking on three of the four register a failure and exit the microgame, and have one register a success instead. Finally, I had to create the mechanics for “Spot The Errors.” For this one, I ended up creating invisible objects that could be placed over wherever the error in the code would be. When clicked on (as the user would be clicking where the error would be), the object would change appearance from being invisible to showing a green circle to indicate a success, and clicking every single one would register the microgame as a win.

Writing the Challenges

With the mechanics for the games created, it was time for me to write the actual challenges themselves that would use these microgames. To create these challenges, I referenced “The Java Tutorials,” a series of official tutorials put out by the Oracle Corporation to educate new programmers on the Java programming language [8]. Unfortunately, it was at this point that the end of the semester drew close, and I found myself unable to implement every challenge: however, I did create enough in the end to demonstrate each microgame with an actual educational challenge involved.

Conclusion

Video games have a lot of potential to be used for the education of numerous topics and subjects, and programming is no exception. Although in the end I was ultimately unable to complete all the challenges I planned to include, I still believe the project serves its intended purpose: a proof of concept to demonstrate the potential viability of using such a gameplay system for coding education. There is plenty of potential to develop the ideas and progress of this project into a more substantial and complete state, as well as the potential for research and playtesting to be performed in order to measure the effectiveness of such a game.

Reference List

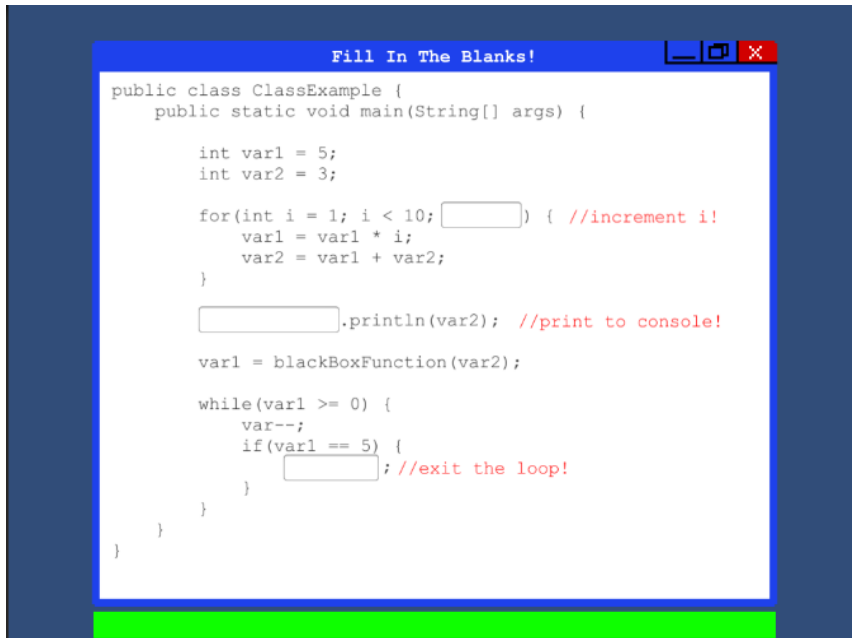
- [1] T. Lougheed, “Video games bring new aspects to medical education and training,” *Canadian Medical Association Journal*, vol. 191, no. 37, 2019.
- [2] V. J. Shute, M. Ventura, and F. Ke, “The power of play: The effects of Portal 2 and Lumosity on cognitive and noncognitive skills,” *Computers & Education*, vol. 80, pp. 58–67, 2015.
- [3] Super Mario Wiki, “WarioWare (series),” *Super Mario Wiki*, 04-May-2021. [Online]. Available: https://www.mariowiki.com/WarioWare_%28series%29. [Accessed: 05-May-2021].
- [4] U. Technologies, *Unity*. [Online]. Available: <https://unity.com/>. [Accessed: 05-May-2021].
- [5] D. Capello, *Aseprite*. [Online]. Available: <https://www.aseprite.org/>. [Accessed: 05-May-2021].
- [6] “FamiStudio NES Music Editor,” *FamiStudio*. [Online]. Available: <https://famistudio.org/>. [Accessed: 05-May-2021].
- [7] *Bfxr. Make sound effects for your games*. [Online]. Available: <https://www.bfxr.net/>. [Accessed: 05-May-2021].
- [8] *The Java™ Tutorials*. [Online]. Available: <https://docs.oracle.com/javase/tutorial/>. [Accessed: 06-May-2021].

Appendix - Selection of Challenges

This appendix contains a selection of one challenge of each microgame type. The full project including all completed challenges can be found at the following URL:

<https://drive.google.com/file/d/1O1CxSsa-ByUEcQRzVsAqNqCsL3gbFsBm/view?usp=sharing>.

Fill In The Blanks



```

public class ClassExample {
    public static void main(String[] args) {

        int var1 = 5;
        int var2 = 3;

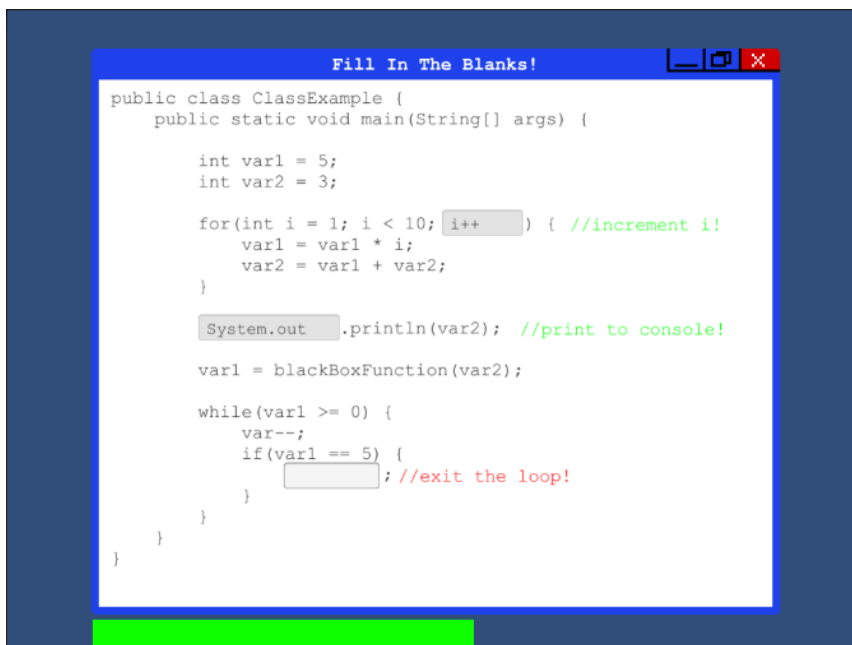
        for(int i = 1; i < 10; ) { //increment i!
            var1 = var1 * i;
            var2 = var1 + var2;
        }

        .println(var2); //print to console!

        var1 = blackBoxFunction(var2);

        while(var1 >= 0) {
            var--;
            if(var1 == 5) {
                ; //exit the loop!
            }
        }
    }
}

```



```

public class ClassExample {
    public static void main(String[] args) {

        int var1 = 5;
        int var2 = 3;

        for(int i = 1; i < 10; ) { //increment i!
            var1 = var1 * i;
            var2 = var1 + var2;
        }

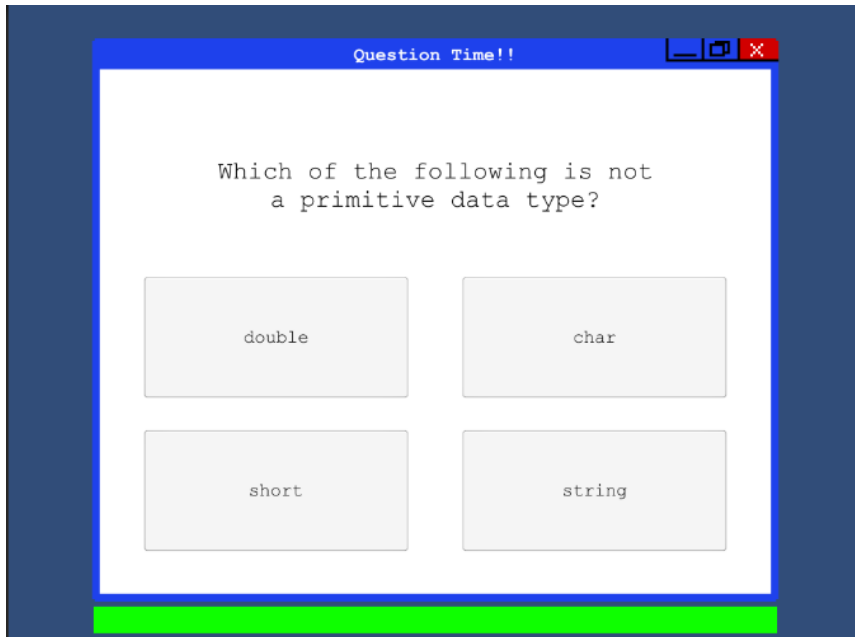
        .println(var2); //print to console!

        var1 = blackBoxFunction(var2);

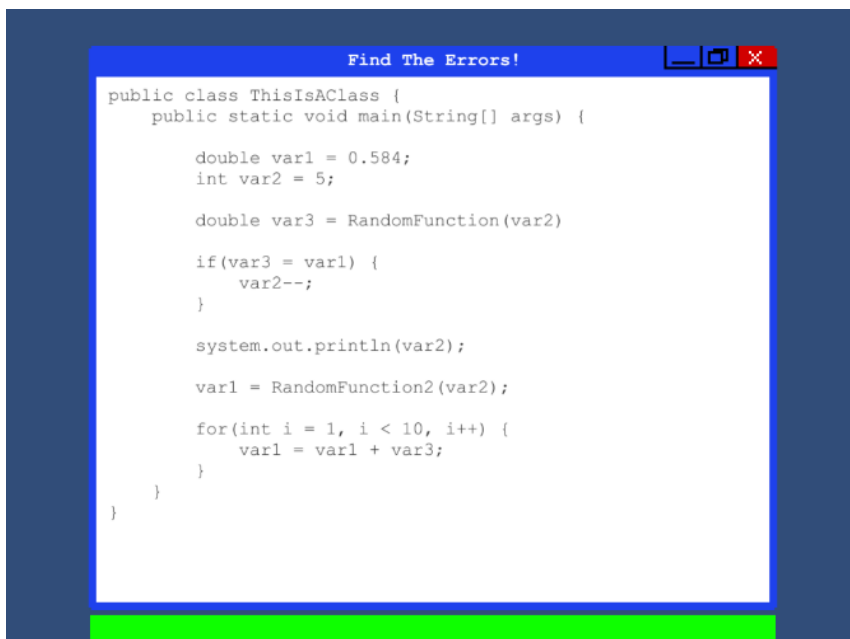
        while(var1 >= 0) {
            var--;
            if(var1 == 5) {
                ; //exit the loop!
            }
        }
    }
}

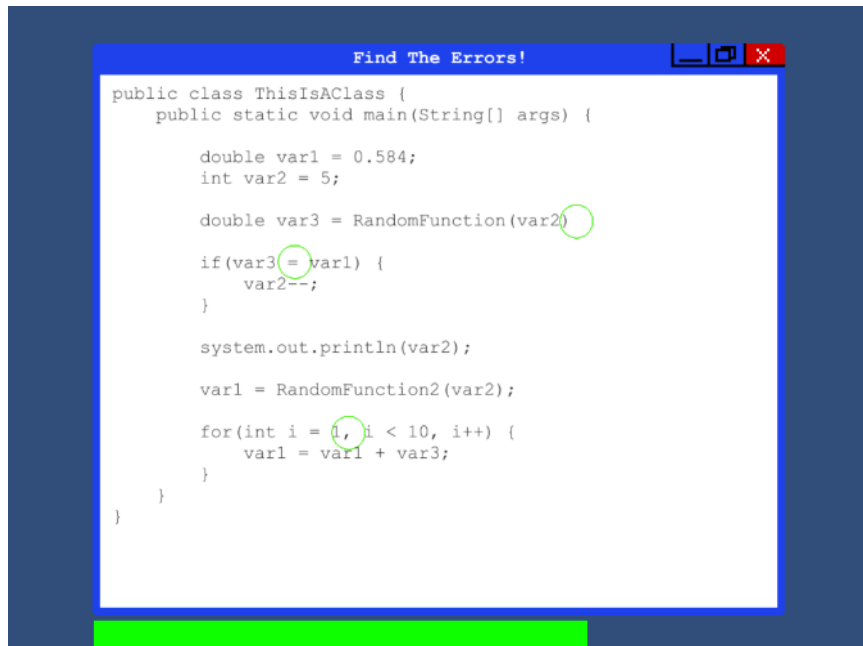
```


Question Time



Spot The Errors





```
public class ThisIsAClass {
    public static void main(String[] args) {

        double var1 = 0.584;
        int var2 = 5;

        double var3 = RandomFunction(var2)

        if(var3 = var1) {
            var2--;
        }

        system.out.println(var2);

        var1 = RandomFunction2(var2);

        for(int i = 1, i < 10, i++) {
            var1 = var1 + var3;
        }
    }
}
```

The screenshot shows a code editor window with a blue title bar that says "Find The Errors!". The code is in Java and contains several errors circled in green: the closing parenthesis of the first function call, the assignment operator in the if statement, the comma in the for loop, and the closing parenthesis of the second function call.