JavaStrike: A Java Programming Engine Embedded in Virtual Worlds

Dominic Kao
Purdue University
West Lafayette, IN, USA
kaod@purdue.edu

ABSTRACT
In this paper, we describe JavaStrike\textsuperscript{1}. JavaStrike is a Java development and execution environment that was developed from scratch inside Unity. The engine currently supports classes, functions, inheritance, polymorphism, interfaces, key-value stores, and much more. JavaStrike allows code to be displayed, executed, and debugged in the virtual world. We then create a third-person shooter game called CodeBreakers, which leverages the JavaStrike engine. CodeBreakers covers basic programming concepts such as variable types, intermediate programming concepts such as stacks, queues, and hashmaps, and advanced programming concepts such as inheritance, interfaces, and method overriding. JavaStrike is a first step towards general purpose programming engines embedded in virtual worlds.

CCS CONCEPTS
• Applied computing → Computer games; • Social and professional topics → Computer science education; Software engineering education; Computing education; Computational thinking;

KEYWORDS
JavaStrike; CodeBreakers; Programming Engine; Virtual Worlds; Java Programming; Games

ACM Reference format:

1 INTRODUCTION
Programming is more than coding, it is the foundational practice that underlies computational thinking [5, 16]. There is strong consensus that computational thinking is broadly important in virtually all subject areas: biology, astronomy, archaeology, chemistry, economics, journalism, law, medicine and healthcare, meteorology, neuroscience, sports, and more [41]. As such, there has been unprecedented interest in developing programming environments for the purpose of teaching programming [27]. However, one major limitation of these programming environments is that they are limited to specific implementations on specific platforms. Creating a new implementation requires extensive effort, the result of which is a limited subset of programming functionality designed for specific use cases. For this reason, we created JavaStrike.

JavaStrike is a Java programming engine we created from the ground up. Developed in Unity, JavaStrike can be utilized on any of the platforms supported by Unity: desktop devices, mobile devices, virtual reality devices, augmented reality devices, console devices, and web browsers. JavaStrike supports many of the functionalities in the Java programming language, including polymorphism, inheritance, interfaces, and data structures. In this paper, we provide an overview of the JavaStrike engine. We then discuss a Java game called CodeBreakers that covers the same breadth of material as the Java course on CodeAcademy, a popular course for learning programming. Finally, we performed a study with Java programming experts, of which the results indicate that CodeBreakers has promise as a game for learning Java. The JavaStrike engine makes it possible to incorporate a run-time Java programming environment into an arbitrary game on an arbitrary device.

2 RELATED WORK
General purpose programming environments for virtual worlds currently do not exist. One can argue that game-specific programming environments (e.g., Colobot [15]) might be generalizable, but these programming environments are often limited to specific applications. Closely related to JavaStrike are programming environments that allow construction of a wide range of types of programs (e.g., Scratch [32], Greenfoot [23]). However, integrating a full programming language (e.g., Java) into a new game takes significant effort.

Games and systems that incorporate programming and computer science include Logo [26], Alice [11] and Storytelling Alice [20], NetLogo [40], MIT App Inventor [42], Gidget [24], LightBot [1], CodeCombat [2], BOTS [17], RoboBuilder [38], AgentSheets and AgentCubes [31], Code.org [10], the Arduino [6], Kodu Game Lab [37], Game Maker [9, 30], Gogo Boards [36], the STELLA programming language [22], Bots & (Main)Frames [28], CMX [27], Mazzy & MazeStar [18, 19], Pyrus [35], and more [3, 7, 21, 25, 29].

3 JAVAStrike
Before detailing the low-level components that make up the JavaStrike engine, we start with a high-level example of how one can use JavaStrike in Unity. For an overview video, see the following: https://youtu.be/0z-qf6miLro.
Figure 1: An example scenario utilizing JavaStrike. Code snippets can be thrown at the blanks (\[\_\]).

3.1 Unity Example Using JavaStrike

To better illustrate how JavaStrike works, consider the following example scenario. (See Figure 1). In this example, there are several code snippets which can be picked up and thrown by the player. For example, the snippet \[1\] is represented internally in the JavaStrike engine by the class \texttt{Variable}. \texttt{Variable} contains a \texttt{VariableReference} object. Objects of type \texttt{VariableReference} specify a \texttt{ToString} method (for visual display) and an \texttt{Execute} method, which permits the run-time to resolve this variable. The \texttt{VariableReference} class inherits from the class \texttt{Value}, which then inherits from \texttt{Executable}, which then inherits from \texttt{Spell}. Both \texttt{Executable} and \texttt{Spell} are abstract base classes which other JavaStrike components inherit from. These abstract classes define basic code for a component to be displayed and executed. This design maximizes code re-use across the many components in JavaStrike.

In this example scenario, there are several code snippets on the ground. One snippet is the sum assignment statement: \[i = a + b\]. This is created using the following code:

```java
return new Assignment(null, new BinaryOperation(null,
    BinaryOperationType.Addition, null));
```

\texttt{Assignment} is a JavaStrike object that is constructed using a variable, and the value to assign that variable. In this specific example, we set the first argument to \texttt{null} to indicate that this variable can be changed by the player in the example (i.e., it will appear as \texttt{\_\_}). The second argument is an object of type \texttt{BinaryOperation}, with a type \texttt{Addition}, and two replaceable values on either side. This generates the \texttt{i = a + b} portion of the snippet. This type of syntax is used to create arbitrary snippets, functions, and classes.

3.2 Engine Overview

The engine contains 3 main sets of components: 1) Components for representing Java code; 2) Components that handle Java execution; and 3) Components for displaying code. See Tables 1, 2, and 3.

<table>
<thead>
<tr>
<th>ArgumentDeclaration.cs</th>
<th>ForeachLoop.cs</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArrayClass.cs</td>
<td>ForLoop.cs</td>
</tr>
<tr>
<td>ArrayConstructor.cs</td>
<td>Function.cs</td>
</tr>
<tr>
<td>Assignment.cs</td>
<td>FunctionCall.cs</td>
</tr>
<tr>
<td>AssignmentOperationType.cs</td>
<td>GenericClass.cs</td>
</tr>
<tr>
<td>BinaryOperation.cs</td>
<td>Increment.cs</td>
</tr>
<tr>
<td>BinaryOperationType.cs</td>
<td>Literal.cs</td>
</tr>
<tr>
<td>BlankStatement.cs</td>
<td>ObjectConstructor.cs</td>
</tr>
<tr>
<td>Block.cs</td>
<td>ObjectFunctionCall.cs</td>
</tr>
<tr>
<td>Class.cs</td>
<td>ObjectIndex.cs</td>
</tr>
<tr>
<td>ClassFunctionCall.cs</td>
<td>ObjectProcedureCall.cs</td>
</tr>
<tr>
<td>ClassProcedureCall.cs</td>
<td>ObjectVariableReference.cs</td>
</tr>
<tr>
<td>ClassReference.cs</td>
<td>OperationAssignment.cs</td>
</tr>
<tr>
<td>Condition.cs</td>
<td>ProcedureCall.cs</td>
</tr>
<tr>
<td>Constructor.cs</td>
<td>Return.cs</td>
</tr>
<tr>
<td>Declaration.cs</td>
<td>Spell.cs</td>
</tr>
<tr>
<td>DeclarationAssignment.cs</td>
<td>Statement.cs</td>
</tr>
<tr>
<td>Decrement.cs</td>
<td>StringConstructor.cs</td>
</tr>
<tr>
<td>Discard.cs</td>
<td>This.cs</td>
</tr>
<tr>
<td>Executable.cs</td>
<td>Type.cs</td>
</tr>
<tr>
<td>Executable.cs</td>
<td>UnaryOperation.cs</td>
</tr>
<tr>
<td>ExternalArrayConstructor.cs</td>
<td>UnaryOperationType.cs</td>
</tr>
<tr>
<td>ExternalFunction.cs</td>
<td>Value.cs</td>
</tr>
<tr>
<td>ExternalFunctionDeclaration.cs</td>
<td>VariableReference.cs</td>
</tr>
<tr>
<td>ExternalReturn.cs</td>
<td>WhileLoop.cs</td>
</tr>
</tbody>
</table>

Table 1: Components handling representation.

<table>
<thead>
<tr>
<th>Clock.cs</th>
<th>PointerView.cs</th>
</tr>
</thead>
<tbody>
<tr>
<td>CodeView.cs</td>
<td>Snippet.cs</td>
</tr>
<tr>
<td>Entity.cs</td>
<td>TextSelection.cs</td>
</tr>
<tr>
<td>GenericEntity.cs</td>
<td>Trigger.cs</td>
</tr>
<tr>
<td>PlaceholderView.cs</td>
<td>WatchView.cs</td>
</tr>
</tbody>
</table>

Table 2: Components handling execution.

Components handling representation. These components deal with how Java code is represented in the system, e.g., \texttt{Class} is an arbitrary Java class, \texttt{Block} is an arbitrary Java code block, and \texttt{VariableReference} is an arbitrary Java variable.

Components handling execution. These components handle Java code execution. \texttt{Scope} handles which variables, functions, and classes are within the current execution scope. \texttt{Thread} handles a single execution thread including call stack, access control (e.g., private, public), exception handling, and so on. \texttt{Magician} handles the overall execution of the program. \texttt{Magician} handles high-level management of threads, including adding new threads, terminating threads, and determining whether a thread is run at program speed, slowed down, or step-wise based on user input.

Components handling display. These components handle display of Java code. \texttt{CodeView} renders Java code to an arbitrary Unity game object. \texttt{CodeView} also facilitates automatic highlighting of
code during execution. 

Snippet handles 3D physics related to code snippets. WatchView renders the values of variables in scope.

Not included in the 3 main sets of components in the JavaStrike engine are 120 other C# classes (which excludes other libraries and classes that were imported for use). These additional classes include helper classes (12), interfaces (3), higher-level components (17), and classes to support CodeBreakers (88).

4 CODEBREAKERS

Next, we developed a Java programming game called CodeBreakers. See Figures 2, 3, 4, and 5. CodeBreakers utilizes the JavaStrike engine, and is based on the same breadth of material covered in the Java course on CodeAcademy. CodeBreakers covers data types, conditionals and control flow, classes and objects, interfaces and inheritance, loops and recursion, polymorphism, method overriding, and data structures.

4.1 Game Overview

CodeBreakers is a fantasy third-person shooter game. The player has just discovered the world of CodeBreakers, and is figuring out how the world works. The world of CodeBreakers, however, is under siege by powerful bugs. Everything in the CodeBreakers world, including the player him/herself, is represented as an object:

class Player extends Character {
    String name = “”;
}

This is a persistent class that evolves throughout the game with the addition of methods (e.g., attack) and variables (e.g., hit points). Programming in CodeBreakers is done by throwing code snippets. Levels progress from replacing single keywords and values, to building single lines of code, to constructing entire code blocks. There are six levels in CodeBreakers:

1. Variable Canyon (data types)
2. Conditional Crossing (conditionals, boolean operations)
3. Loop of Life (loops)
5. Stepping Up (recursion)
6. One by One, Please (data structures)

Each CodeBreakers level builds upon previous ones by introducing new concepts.

5 USER STUDY

We ran a study with Java programming experts to ascertain initial thoughts about the CodeBreakers game.

5.1 Methods

5.1.1 Quantitative Measures. We use a standardized programming experience questionnaire [14], and the Player Experience of Need Satisfaction (PENS) scale [34]. PENS is based on self-determination theory (SDT) [12]. PENS contends that the psychological “pull” of games are largely due to their ability to engender three needs—competence (seek to control outcomes and develop mastery [39]), relatedness (seek connections with others [4]), and
Figure 5: In Level 2, the first bridge requires the player to find a code snippet matching the number of characters in their name.

autonomy (seek to be causal agents [8] while maintaining congruence with the self) [34]. PENS is considered a robust framework for assessing player experience [13, 33].

5.1.2 Participants. 27 Java programming experts were recruited through Amazon Mechanical Turk (AMT) to assess CodeBreakers. The data set consisted of 21 male, and 6 female participants. Participants were between the ages of 19 and 48 (M = 29.4, SD = 7.9), and were all from the United States. In order to recruit participants with Java programming experience, we used the Employment Industry - Software & IT Services qualification on AMT. Participants were reimbursed $15.00 for their participation.

5.1.3 Protocol. Participants played the entire CodeBreakers (in browser using WebGL) game. In case they were unable to get past a particular level, they were provided with video walkthroughs. After completing each level, participants were asked to describe their thoughts on it. After completing the game, participants were asked to describe what they felt were some of the things that the game did well, and some areas for improvement. Participants then completed the programming experience questionnaire, the PENS, and demographics.

5.2 Results

5.2.1 Prior Java Programming Experience. On a scale of 1:Very Inexperienced to 10:Very Experienced, participants rated their own Java programming experience as M=8.2, SD=1.4. For the question How many additional [programming] languages do you know (medium experience or better)?: participants had an average of M=4.3, SD=2.9. Participants averaged M=8.1, SD=4.9 years of programming experience, and M=4.4, SD=4.8 years of programming experience on large software projects (e.g., in a company). 21 participants were involved in professional projects that involved programming. Of those participants, 9 said those projects typically involved <900 lines of code, 7 said 900-40000, and 5 said >40000.

5.2.2 Player Experience of Need Satisfaction. On a scale from 1:Do Not Agree to 7:Strongly Agree, participants’ average scores on the PENS were M=5.4, SD=1.4 (competence), M=5.2, SD=1.1 (autonomy), M=3.5, SD=1.5 (relatedness), M=4.1, SD=1.4 (presence/immersion), M=4.7, SD=1.7 (intuitive controls).

5.2.3 Level Feedback. Players felt that Level 1 was a good introduction to Java concepts. Participants also enjoyed the puzzles in Level 2 and liked how the concepts were introduced. Towards the end of Level 3, participants had to combine a previously acquired sword (deal damage) and staff (loop over enemies in current level) to create a new weapon (deal damage to all enemies). However, some participants got stuck and did not know they had to do this. Participants felt that Level 4 was an interesting application of inheritance and interfaces (one part of Level 4 requires players to use inheritance to access the health field of an enemy). Participants felt Level 5 was a creative method of teaching loops and recursion (players were required to re-build a damaged stairwell, first using loops, then using recursion). They liked that the recursive puzzle built upon the iterative version of the puzzle, making the original iterative solution inaccessible for the purposes of having the player consider the recursive solution for the same identical problem. Participants felt Level 6 (which involved enemies sieging the top of a castle in various “formations” which depended on the data structure they were stored in) was a climactic ending and a good introduction to data structures.

5.2.4 Overall Feedback. Qualitatively, participants spoke highly of the main ideas behind CodeBreakers (19x), and felt that the level design in CodeBreakers was a strong point (16x). However, participants felt that more instructions were needed for beginner Java programmers (12x).

5.3 Discussion

27 Java programming experts rated the game positively on the need satisfaction measures of competence, autonomy, immersion, and controls. Overall, participants commended the main concept of the game and the level designs. However, participants felt that more instructional scaffolding for beginners would be beneficial.

With respect to Java programming experts, the game was moderately effective at engendering need satisfaction. We feel that these scores can be significantly improved in future versions of CodeBreakers once the aesthetics and gameplay are more polished, music and sounds are added, and once we have added in sufficient tutorials so that players are less likely to get stuck on the more difficult portions of the game. Future studies will seek to study less experienced Java programmers and their learning of Java concepts.

6 CONCLUSION

In this paper, we have described the JavaStrike engine. The JavaStrike engine is a Java programming engine for virtual worlds. JavaStrike was developed from scratch in Unity, and is supported on over 25 platforms. The JavaStrike engine supports polymorphism, inheritance, interfaces, data structures, and more. We have also described a game that utilizes the JavaStrike engine called CodeBreakers. CodeBreakers is a third-person Java programming game based on the same breadth of material covered in the Java course on CodeAcademy. To the best of our knowledge, JavaStrike is the first attempt at building a general purpose programming engine for virtual worlds. The JavaStrike engine makes it possible for designers, developers, and researchers to integrate Java programming into arbitrary virtual worlds on arbitrary platforms.
6 ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their valuable feedback.

REFERENCES