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*[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:
Dominic Kao. 2019. JavaStrike: A Java Programming Engine Embedded in Virtual Worlds. In *Proceedings of The Fourteenth International Conference on the Foundations of Digital Games, San Luis Obispo, CA, USA, August 26–30, 2019 (FDG '19),* 5 pages.
https://doi.org/10.1145/3337722.3341828

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.

[1]JavaStrike Video: https://youtu.be/0z-qf6miLro.

FDG '19, August 26–30, 2019, San Luis Obispo, CA, USA

ACM ISBN 978-1-4503-7217-6/19/08…$15.00
https://doi.org/10.1145/3337722.3341828

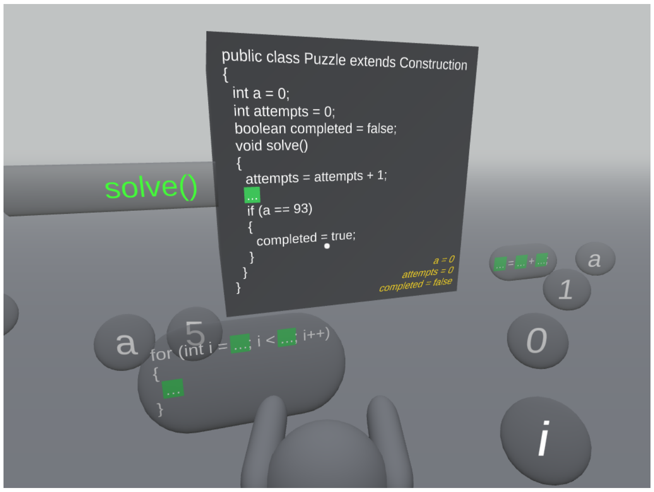

Figure 1: An example scenario utilizing *JavaStrike*. Code snippets can be thrown at the blanks (...).

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 `i` is represented internally in the *JavaStrike* engine by the class `Variable`. `Variable` contains a `VariableReference`. Objects of type `VariableReference` specify a `ToString` method (for visual display) and an `Execute` method, which permits the run-time to resolve this variable. The `VariableReference` class inherits from the class `Value`, which then inherits from `Executable`, which then inherits from `Spell`. Both `Executable` and `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: ... = ... + ... This is created using the following code:

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

`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 `null` to indicate that this variable can be changed by the player in the example (i.e., it will appear as ...). The second argument is an object of type `BinaryOperation`, with a type `Addition`, and two replaceable values on either side. This generates the ... + ... 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.

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

Table 1: Components handling representation.

Magician.cs	Scope.cs
Object.cs	Thread.cs
PayloadObject.cs	Variable.cs

Table 2: Components handling execution.

Clock.cs	PointerView.cs
CodeView.cs	Snippet.cs
Entity.cs	TextSelection.cs
GenericEntity.cs	Trigger.cs
PlaceholderView.cs	WatchView.cs

Table 3: Components handling display.

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

Components handling execution. These components handle Java code execution. `Scope` handles which variables, functions, and classes are within the current execution scope. `Thread` handles a single execution thread including call stack, access control (e.g., `private`, `public`), exception handling, and so on. `Magician` handles the overall execution of the program. `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. `CodeView` renders Java code to an arbitrary Unity game object. `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)
4. Encapsulation. Inheritance. Polymorphism. (inheritance)
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 2: In Level 1, the player must throw the correct data types matching the bug variables to neutralize them.

Figure 3: In the second half of Level 1, the player must find a code snippet to cure a wounded knight.

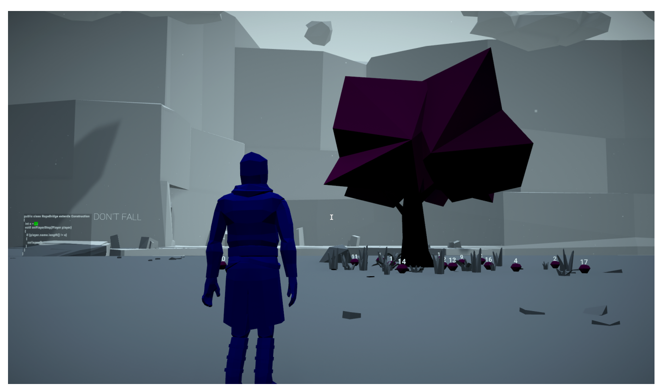

Figure 4: In Level 2, the player must cross a chasm by traversing a series of bridges.

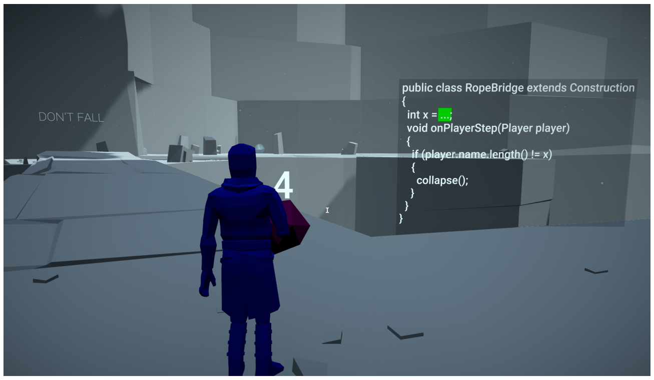

Figure 5: In Level 2, the first bridge requires the player to find a code snippet matching the number of characters in their name.

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.

7 ACKNOWLEDGMENTS

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

REFERENCES

[1] 2008. LightBot (http://lightbot.com/). (2008).
[2] 2016. CodeCombat. (2016). https://codecombat.com/
[3] Tiffany Barnes, Eve Powell, Amanda Chaffin, and Heather Lipford. 2008. Game2Learn. *Proceedings of the 3rd international conference on Game development in computer science education - GDCSE '08* January (2008), 1–5. https://doi.org/10.1145/1463673.1463674
[4] Roy F Baumeister and Mark R Leary. 1995. The need to belong: desire for interpersonal attachments as a fundamental human motivation. *Psychological Bulletin* 117, 3 (1995), 497.
[5] Karen Brennan and Mitchel Resnick. 2012. New frameworks for studying and assessing the development of computational thinking. *Annual American Educational Research Association meeting, Vancouver, BC, Canada* (2012), 1–25.
[6] Leah Buechley, Mike Eisenberg, and Jaime Catchen. 2008. The LilyPad Arduino: Using computational textiles to investigate engagement, aesthetics, and diversity in computer science education. *CHI '08 Proceedings of the twenty-sixth annual SIGCHI conference on Human factors in computing systems* (2008), 423–432.
[7] Amanda Chaffin, Katelyn Doran, Drew Hicks, and Tiffany Barnes. 2009. Experimental evaluation of teaching recursion in a video game. *Proceedings of the 2009 ACM SIGGRAPH Symposium on Video Games - Sandbox '09* 1, 212 (2009), 79. https://doi.org/10.1145/1581073.1581086
[8] Valery Chirkov, Richard M Ryan, Youngmee Kim, and Ulas Kaplan. 2003. Differentiating autonomy from individualism and independence: A self-determination theory perspective on internalization of cultural orientations and well-being. *Journal of Personality and Social Psychology* 84, 1 (2003), 97.
[9] Kajal Claypool and Mark Claypool. 2005. Teaching software engineering through game design. In *ACM SIGCSE Bulletin*, Vol. 37. ACM, 123–127.
[10] Code.org. 2014. Code.org. (2014). http://code.org
[11] Stephen Cooper, Wanda Dann, and Randy Pausch. 2000. Alice : a 3-D Tool for Introductory Programming Concepts. *Journal of Computing Sciences in Colleges* 15, 5 (2000), 107–116.
[12] Edward L Deci and Richard M Ryan. 2012. Motivation, personality, and development within embedded social contexts: An overview of self-determination theory. *The Oxford Handbook of Human Motivation* (2012), 85–107.
[13] Sebastian Deterding. 2015. The Lens of Intrinsic Skill Atoms: A Method for Gameful Design. *Human-Computer Interaction* 30, 3-4 (2015), 294–335. https://doi.org/10.1080/07370024.2014.993471
[14] Janet Feigenspan, Christian Kastner, Jorg Liebig, Sven Apel, and Stefan Hanenberg. 2012. Measuring programming experience. *2012 20th IEEE International Conference on Program Comprehension (ICPC)* 2005 (2012), 73–82. https://doi.org/10.1109/ICPC.2012.6240511
[15] Epsitec Games. 2017. CoLoBoT. (2017).
[16] S. Grover and R. Pea. 2013. Computational Thinking in K-12: A Review of the State of the Field. *Educational Researcher* 42, 1 (2013), 38–43. https://doi.org/10.3102/0013189X12463051
[17] Andrew Hicks, Barry Peddycord III, and Tiffany Barnes. 2014. Building games to learn from their players: Generating hints in a serious game. In *International Conference on Intelligent Tutoring Systems*. Springer, 312–317.
[18] Dominic Kao and D. Fox Harrell. 2015. Mazzy: A STEM Learning Game. *Foundations of Digital Games* (2015).
[19] Dominic Kao and D. Fox Harrell. 2018. The Effects of Badges and Avatar Identification on Play and Making in Educational Games. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems - CHI'18*.
[20] Caitlin Kelleher, Randy Pausch, and Sara Kiesler. 2007. Storytelling alice motivates middle school girls to learn computer programming. *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '07* (2007), 1455.
[21] Jackie O Kelly, N U I Maynooth, and J Paul Gibson. 2006. A non-prescriptive approach to teaching programming. *ACM SIGCSE Bulletin* 38, 3 (2006), 217–221.
[22] Kyu Han Koh, Ashok Basawapatna, Vicki Bennett, and Alexander Repenning. 2010. Towards the Automatic Recognition of Computational Thinking for Adaptive Visual Language Learning. *Visual Languages ...* (2010), 59–66. https://doi.org/10.1109/VLHCC.2010.17
[23] Michael Kölling. 2010. The greenfoot programming environment. *ACM Transactions on Computing Education (TOCE)* 10, 4 (2010), 14.
[24] Michael J. Lee, Andrew J. Ko, and Irwin Kwan. 2013. In-game assessments increase novice programmers' engagement and level completion speed. *Proceedings of the ninth annual international ACM conference on International computing education research - ICER '13* (2013), 153. https://doi.org/10.1145/2493394.2493410
[25] Renny S.N. Lindberg, Teemu H. Laine, and Lassi Haaranen. 2018. Gamifying programming education in K-12: A review of programming curricula in seven countries and programming games. *British Journal of Educational Technology* (2018). https://doi.org/10.1111/bjet.12685
[26] Logo Foundation. 2017. Logo (http://el.media.mit.edu/logo-foundation/). (2017).
[27] Christos Malliarakis, Maya Satratzemi, and Stelios Xinogalos. 2017. CMX: The Effects of an Educational MMORPG on Learning and Teaching Computer Programming. *IEEE Transactions on Learning Technologies* 10, 2 (2017), 219–235. https://doi.org/10.1109/TLT.2016.2556666
[28] Edward F Melcer and Katherine Isbister. 2018. Bots & (Main)Frames: Exploring the Impact of Tangible Blocks and Collaborative Play in an Educational Programming Game. *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (2018), 1–14. https://doi.org/10.1145/3173574.3173840
[29] Mathieu Muratet, Patrice Torguet, Fabienne Viallet, Jean-pierre Jessel, Mathieu Muratet, Patrice Torguet, Fabienne Viallet, and Jean-pierre Jessel Experimental Feed. 2016. Experimental Feedback on Prog & Play , a Serious Game for Programming Practice To cite this version : Experimental feedback on Prog & Play , a serious game for programming practice. (2016).
[30] Mark Overmars. 2004. Teaching computer science through game design. *Computer* 37, 4 (2004), 81–83.
[31] Alex Repenning. 1993. Agentsheets: a tool for building domain-oriented visual programming environments. In *Proceedings of the INTERACT'93 and CHI'93 conference on Human factors in computing systems*. ACM, 142–143.
[32] M Resnick and John Maloney. 2009. Scratch: programming for all. *Communications of the ...* (2009). http://dl.acm.org/citation.cfm?id=1592779
[33] Scott Rigby and Richard M Ryan. 2011. *Glued to games: How video games draw us in and hold us spellbound: How video games draw us in and hold us spellbound*. ABC-CLIO.
[34] Richard M. Ryan, C. Scott Rigby, and Andrew Przybylski. 2006. The Motivational Pull of Video Games: A Self-Determination Theory Approach. *Motivation and Emotion* 30, 4 (2006), 344–360. https://doi.org/10.1007/s11031-006-9051-8
[35] Joshua Shi, Armaan Sha, Garrett Hedman, and Eleanor O Rourke. 2019. Pyrus: Designing A Collaborative Programming Game to Support Problem-Solving Behaviors. *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (2019), 1–12.
[36] Arnan Sipitakiat, Paulo Blikstein, and David P Cavallo. 2004. GoGo board: augmenting programmable bricks for economically challenged audiences. In *Proceedings of the 6th international conference on Learning sciences*. International Society of the Learning Sciences, 481–488.
[37] Kathryn T Stolee and Teale Fristoe. 2011. Expressing computer science concepts through Kodu game lab. In *Proceedings of the 42nd ACM technical symposium on Computer science education*. ACM, 99–104.
[38] David Weintrop and Uri Wilensky. 2014. Program-to-play video games: Developing computational literacy through gameplay . (2014), 1–7. http://ccl.northwestern.edu/papers/2014/GLS-2014final.pdf
[39] Robert W White. 1959. Motivation reconsidered: The concept of competence. *Psychological Review* 66, 5 (1959), 297.
[40] Uri Wilensky. 1999. NetLogo. (1999).
[41] Jeanette M. Wing. 2006. Computational Thinking. (2006), 33–35. https://doi.org/10.1007/s11277-016-3679-9
[42] David Wolber. 2011. App inventor and real-world motivation. In *Proceedings of the 42nd ACM technical symposium on Computer science education*. ACM, 601–606.