Robotics Rule-Based Formalism to Specify Behaviors in a Visual Programming Environment

Daniela Marghitu
Computer Science and Software Engineering
Auburn University
Auburn, USA
marghda@auburn.edu

Stephen Coy
FUSELABS
Microsoft
Redmond, USA
scoy@microsoft.com

***Abstract*—*Microsoft's Kodu is a visual programming environment for children inspired by robotics rule-based formalism to specify behaviors. The reactive nature of this approach means that systems just react to what they are currently sensing, they do not need to have a perfect model of the world, and they can easily respond to external changes in the world. This resilience makes behavior-based systems perfect for real world robotics use. When developers chose this approach for Kodu, the hope was that this resilience would help minimize the potential frustration of young users. This paper introduces the Kodu developers' strategy, results and future plans in developing a user-friendly programming environment that successfully introduces children in a fun and easy way to the fundamental concepts of computer science and computational thinking.*

Keywords*—*Kodu; rule-based formalism; computational thinking

I. Introduction

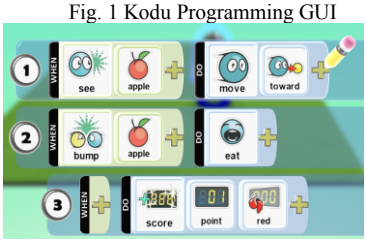
Several theoretical justifications are available for the successful use of manipulative resources that enable students' progress through a concrete-representational-abstract learning sequence [1]. In computer science (CS), physical manipulatives represent the foundation of CS Unplugged activities [2], and virtual manipulatives (e.g., shaped blocks corresponding to syntactic classes such as imperative statements, Boolean expressions, and loops), are used to develop the Graphical User Interfaces (GUI) for drag and drop based programming frameworks [3].

When developers decided to create the Kodu language [4] in late 2006, the goal was to address the challenge of introducing children to CS and computational thinking in a fun way while developing 3D simulated worlds. At that time Scratch was already well established and has since generated numerous imitators. Instead of following Scratch's example, Kodu developers took as inspiration the robotics rule-based formalism to specify behaviors [5] [6]. The reactive nature of behavior-based programming means that the systems just react to what they are currently sensing, they do not need to have a perfect model of the world, and they can easily respond to external changes in the world. This resilience makes behavior-based systems perfect for real world robotics and its numerous applications. When Kodu developers chose this approach, the hope was that this resilience would help minimize the frustration of introductory, young programmers.

II. The Past

The Kodu language consists of a series of rules that are evaluated for each frame (nominally 60 times per second). Each rule consists of a *WHEN* clause and a *DO* clause. The *WHEN* clause consists of a *Sensor and zero* or more filters. The *DO* clause of the rule contains an *Actuator* and zero or more *Modifiers*. For instance, in Fig. 1 the first rule shows the *See sensor* along with an *Apple filter*. This will sense all apples in the world. The list of apples in the world is internally reduced to a single instance using a *Selector* which chooses the closest instance. If an apple is seen then the *DO* clause of the rule is applied. In this case the *Move Actuator* is activated. The direction of the movement is controlled by the *Toward Modifier* which uses the location of the apple detected in the *WHEN* clause as the target destination. Similarly, in the second rule, if the *Bump Sensor* detects an apple that apple is passed to the *Eat Actuator* to be acted upon.

Fig. 1 Kodu Programming GUI



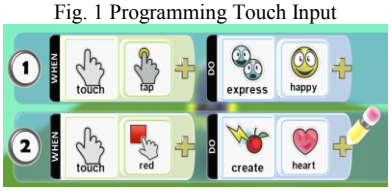
The third rule demonstrates another aspect of Kodu programming. Rules can be indented to indicate that they are a *child* of the preceding rule. Child rules are only executed when the *WHEN* clause of their parent evaluates to true, in this case when an apple is bumped. The empty *WHEN* clause in rule 3 implies that the *WHEN* always evaluates to true. One of the results of the reactive nature of the language is that the movement system in Kodu is both easier to work with and yet less accurate. Precise movement, as seen in Logo [7], is not possible. Instead of moving in precise increments Kodu characters move "toward" other characters. Since various characters have differing movement capabilities the speed and accuracy of the resulting undertaking may vary. If the target

character changes position during this process, the reactive nature of the system comes into play and the movement of the programmed character adjusts automatically. Another advantage of the language's robotics roots is that the tiles in the system have comprehensible, physical counterparts. Kids readily understand seeing an apple or eating an apple. The high-level nature of the tiles empowers users to create interesting behaviors with only a few lines of code. This model is quickly providing a positive feedback cycle for new programmers.

Another way that Kodu eases the transition for new programmers is that the language doesn't use word *variable*. Instead, users are presented with *scores* that can be added to, subtracted from and compared. From a CS perspective, there is no difference. However, for new users, the concept of a score and how it can be manipulated is already an established, and therefore, concept. By playing to the familiar for these young programmers, Kodu helps to eliminate one more potential stumbling block. Similarly the *when/do* terminology was also carefully considered and chosen over the more obvious *if/then*. The use of *when* implies that the Boolean value of the statement may change over time whereas *if* implies a more static value. For the actuator clause *do* is clearly more action oriented, than *then*. In some ways the difference may seem trivial but when trying to reach new programmers the developers wanted every advantage.

Kodu's origins on the Xbox dictated the GUI to be fully accessible via the Xbox Game Controller. This constraint has proven beneficial since it ensures a minimalist interface. With the move to the PC, full support for keyboard and mouse was added. More recently, touch support has been added, which makes Kodu more accessible for the user with hand motion control impairments {see Fig. 2 [8]}. In each case, the GUI responds dynamically as the user switches the input devices. The support for multiple input devices also extends to the Kodu programming language.

Fig. 1 Programming Touch Input

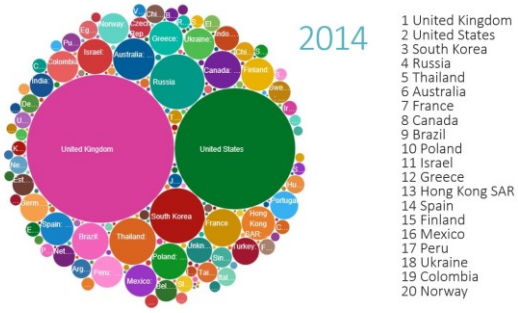


The Kodu language has been extended to support mode specific input within the games created by the users. As shown in Fig. 2, Kodu characters can be programmed to respond to touch input. Additionally, when the user programs a character to respond to a touch on a button, in this case the red button, the button will then appear on the screen and respond to both touch and mouse input.

III. The Present

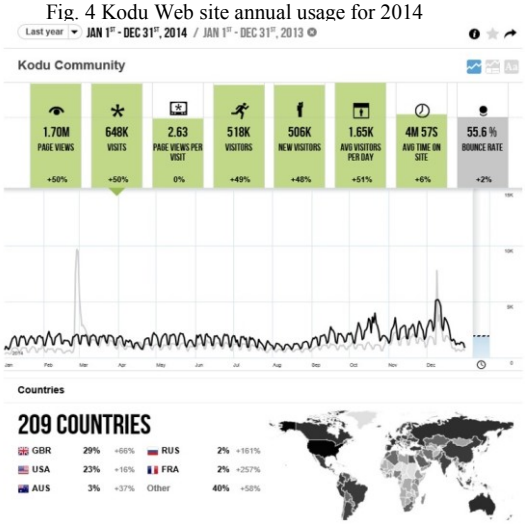
To date, Kodu has been translated into seventeen languages (by the Kodu community members), and has been successful around the world. Kodu is one of the platforms recommended for K-8 educators by Code.org, along with Alice and Scratch [9]. It is also one of three languages, along with Scratch and Logo, recommended by the Computing at School Working Group for the new United Kingdom (UK) primary school computing curriculum [10].

Fig. 3 Kodu top 20 downloads for 2014 by country



Kodu is currently downloaded in over two hundred countries. Fig. 3 shows the top twenty countries where it is used. Currently about 135k games have been uploaded to the KoduGameLab web site [4]. The UK surpasses the US, and this is mainly due to the UK laws requiring CS education at all K-12 grade levels. It is estimated that as much as 1/3 of school age children in the UK have been exposed to Kodu. It's also interesting to note that South Korea and Thailand both show up in the top 5 despite not having localizations available. There's clearly a demand in Asia for something like Kodu, and the developing team hopes to fill it. Fig. 4 shows the 2014 annual usage of the Kodu web site. The remarkable thing indicated by this report is that Kodu is clearly used more during the week than the weekend. For the developing team, this is another indicator that Kodu is being used largely in schools or after school programs.

Fig. 4 Kodu Web site annual usage for 2014



IV. The Future

In 2014, Kodu developers released its commercial successor Project Spark (projectspark.com). The language used in Project Spark showed one possible progression of the Kodu language where the priority focused more on functionality and flexibility rather that simplicity and ease of use.

For the next generation of Kodu, the developing team plans on continuing Kodu's focus on young, first-time programmers. One of the key improvements developers hope to make is to eliminate the possibility of incomplete statements. Whereas the Kodu language does prevent syntax errors, it still allows incomplete statements to be created. For instance, in Fig. 1 rule 3 shows incrementing the red score by one point. If the *Red tile* is not placed then it is ambiguous which score should be incremented. Similarly in the *WHEN* clause we often see kids programming *WHEN GamePad DO Move* to control their characters when they need *WHEN GamePad LeftStick DO Move*. The first option looks like it should be functional but isn't. The developing team plans to demote ambiguous tiles like *Score+* and *GamePad* and make them groups in the tile selection menu. This would then make *LeftStick* a sensor rather than a filter on *GamePad*. Likewise, instead of *Score+ Red* we would have *Red+,* which would eliminate the ambiguous case. The incomplete statement syntax was not seen as a problem (reasonable defaults were chosen for these cases) until the Kodu developing team started extending the language. Recently, the Kodu team has doubled and that will allow a bit more aggressive work on updates. However, this will still be a large task and will, unfortunately, break backwards compatibility for our existing users.

Closely related with these ambiguous tiles is a problem that occurs when users have filters than could possibly apply to multiple objects. The users' worst case for this is the *ShotHit* sensor that detects when a shot hits a character. With the program *WHEN ShotHit Red* it is not clear if the red filter should apply to the shooter, the target, or the shot itself.

V. CASE STUDY

Kodu, aimed at middle and elementary school students [10] [11] [12] [13], was integrated in the Auburn University (AU) Laboratory for Education and Assistive Technology Lab (LEAT) K12 computing outreach and research inclusive program since 2008. Kodu curriculums were developed for formal and informal K12 programs.

A three-step adaptive model and related curriculum for introduction to computing was developed for AU K-12 Robo Camp [14]. In this model of computing instruction, students start with the highly scaffolded programming environment, Kodu, and progress to more challenging frameworks (e.g., Alice or App Inventor and Lego EV3). While moving forward and sometime backwards, deepening their individual abilities and preferences, between the three steps of the model, students are encouraged by instructors to explore how concepts such as variables, conditionals, and looping are implemented toward building the foundation of their computational thinking. Fig. 5 shows a Kodu world developed by Robo Camp middle school students as an implementation of the CS Unplugged Binary code activity [2]. Another informal curriculum was developed in 2013 for the Computer Science for all Girls (c4allg.eng.auburn.edu) summer camp [8], funded by the 2012 National Center for Women in Information Technology Academic Alliance Seed Fund.

Fig. 5 Kodu world implementing the CS Unplugged Binary activity

A formal Kodu curriculum was developed for 7th grade, (cs4allb.eng.auburn.edu) with funds from Microsoft Research and AccessComputing, and has been taught since 2013 [14]. Feedback obtained from instructors, participants and participants' parents, reinforced by the results of the pre and post programs evaluations, clearly indicated all these programs as a success. Therefore, the LEAT team will continue to refine its formal and informal Kodu curriculum for middle school and explore moving the first programing instruction to elementary school.

REFERENCES

[1] ETA hand2mind; "Why Teach Mathematics with Manipulatives?" http://www.hand2mind.com/pdf/Benefits_of_Manipulatives.pdf. Accessed 2015 July 1.

[2] CS Unplugged, http://csunplugged.com/, Accessd 2015, July 1.

[3] Ward, B., Marghitu, D., Bell, T., Lambert, L.: Teaching computer science concepts in Scratch and Alice, Pages: 173-180, Journal of Computing Sciences in Colleges, Volume 26, Issue 2, December 2010

[4] Kodu, http://www.kodugamelab.com/. Accessed 2015 July 1.

[5] J. Jones, Robot Programming – A Practical Guide to Behavior-Based Robotics, New York, McGraw-Hill, 2004.

[6] Coy, S. "Kodu game lab, a few lessons learned", XRDS: Crossroads, The ACM Magazine for Students, Pages 44-47, Volume 19 Issue 4, Summer 2013, ACM New York, NY, USA

[7] S. Papert, Mindstorms: Children, Computers, and Powerful Ideas, New York, Basic Books, 1980.

[8] Marghitu, D., et-al.: "Attracting Girls, Special Needs, Minority and Underserved K-12 Students to Computing Majors and Careers", SITE 2014 Conference, Jacksonville, FL, March 17-19, 2014

[9] Code.org,"3rd Party Educator Resources;" http://code.org/educate/3rdparty. Accessed 2015 June 30.

[10] Berry, M. "Computing in the national curriculum: A guide for primary teachers." Computing At School, at http:// computingatschool.org.uk/. Accessed 2014 September 20.

[11] Touretzky, D. S., Marghitu, D., et-al.:"Accelerating computational thinking using scaffolding, staging, and abstraction", Proceedings of SIGCSE '13, Denver, CO. Association for Computing Machinery, pp. 609-614, 2013

[12] Touretzky, D. S.: Teaching Kodu with physical manipulatives. ACM Inroads, 5(4):44-51, 2014

[13] Fristoe, T., et-al.: Say it with systems: Expanding Kodu's expressive power through gender-inclusive mechanics. Proceedings of Foundations of Digital Games, France, June 2011

[14] Marghitu, D., et-al., "Kodu, Alice and Lego Robotics: A three-step model of Effective Introducing Middle School Students to Computer Programming and Robotics", SIGCSE 2013, Denver, CO, March 6-9, 2013