Kodu Game Lab: a programming environment

Allan Fowler [1], Teale Fristce [2], Matthew MacLauren [3]

[1] Waiariki Institute of Technology, New Zealand
[2] University of California, Santa Cruz, United States
[3] Microsoft FUSE Labs / Microsoft Research (int.)

Article Information

Received: February 2012
Accepted: April 2012
Available: online May 2012

Key words: Kodu Game Lab, programming, computer games, children, 3D

Abstract

Kodu Game Lab is a tile-based visual programming tool that enables users to learn programming concepts through making and playing computer games. Kodu is a relatively new programming language designed specifically for young children to learn through independent exploration. It is integrated in a real-time isometric 3D gaming environment that is designed to compete with modern console games in terms of intuitive user interface and graphical production values.

1: Introduction

Kodu Game Lab (KGL) is a tile-based visual programming environment that enables users to create and play video games and animated stories that include feature rich multimedia and multisensory content. The software was developed to enable young children to create video games with graphical content that can compete with the types of games that most users would consider as standard. The visual (Figure 1), auditory and kinesthetic attributes of the software have the potential to make it attractive and distinct for the students and provide deep and engaging learning experiences.

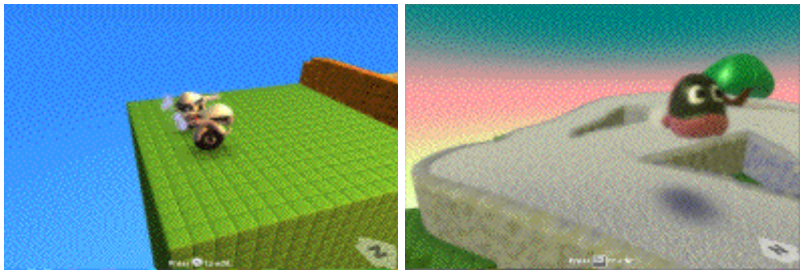

Figure 1: screenshots from Kodu Game Lab projects

The core of the software is its programming language, Kodu. This is a high-level visual language that can be represented as a context-free grammar (Stolee, 2010). Unlike other programming languages like Java or C++, Kodu is entirely event driven, whereby programming involves the placement of tiles in a meaningful sequence to form a condition and action on each rule (Figure 2).

Figure 2: programming interface in KGL

This is similar to the behavior based approach used in NXT-G (a tile-based programming language for Lego Mindstorms) which has been successfully incorporated into primary and secondary school classrooms (Apiola et al, 2010, Lund & Pagliarini, 2000) as well as tertiary education institutions (Klassner & Anderson, 2003).

KGL was initially developed for the Xbox 360 and has recently been ported to the Windows compatible Personal Computer (Windows 7; Windows Vista; Windows XP, a graphics card that supports DirectX 9.0c and Shader Model 2.0 or higher; .NET Framework 3.5 or higher; or a XNA Framework 3.1 Redistributable required).

KGL features navigation with the Xbox 360 controller, which most children are familiar with. However, the software still allows keyboard and mouse input to support legacy PC hardware, thus making the software more accessible to potential users and more affordable to implement in economically disadvantaged schools.

2: Modes of Use

KGL has two user modes – a play mode and an edit game mode. In the play mode users can play the prebuilt games and view supplied interactive tutorials. These provide some instructions and include basic game worlds and characters, as well as some initial code to help get the user started (Figure 3).

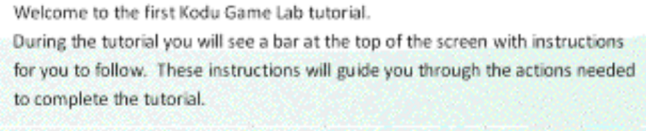

Figure 3: a KGL Tutorial Screen

To edit a game, the user can select the Edit World option in the home menu, or press the Escape key on the keyboard (or the back button on the Xbox 360 controller) while playing. This ease of transition between the two modes allows for rapid iteration during development.

KGL also includes a community facility for sharing and playing user created games. This service gives the opportunity to play with, edit and learn from other programmers and thus has the potential to provide a community of practice (Wenger, 1998).

3: Programming Environment

As noted, KGL includes finished playable games as well as tutorials that can assist the user in creating their own virtual worlds or modifying the existing games. The intention of these tutorials is to encourage self-directed learning. By providing editable working games, KGL encourages students to explore the inner workings of existing games and to implement into their own games.

3.1: The KGL Game World

A KGL game world can include objects, sounds, paths and an environment:

Objects

The objects available in KGL include both characters and environmental elements (trees, apples, rocks, coins and clouds) that are programmable. The extent of programmability of each object depends on the type of object and its inherent (and sometimes unique) attributes. The major difference between the characters and the environmental elements is that character can be programmed to move (either of its own accord or controlled by the user) whereas the majority of the environmental elements can not move unless assisted by another character. Within the game world each object can see and hear all other objects (unless set as invisible), and can detect when another object has collided with it.

Sounds

KGL includes both sound effects (explode, bong, pow, *etc*.) and background music. The sound effects can be triggered based on certain events and the music can be programmed to play through the duration of the game.
Paths

Paths are included in KGL to allow the user to program a character to move within a constraint or direction. Consider the following code (figure 4) where the character will follow the direction of the red path.

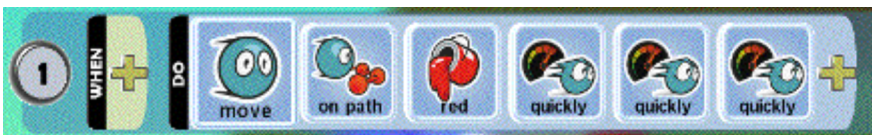

Figure 4: Programming a character to follow a path.

Paths can also be used for decorative purposes. For example a bridge, wall or a row of flora can be included in the world for aesthetics. Like objects, the path can be visible or invisible, the latter being useful for controlling the movement of objects in the game.

The environment

The environment is created using the terrain editing tools, discussed in depth in section 3.3. The World Tool also allows the programmer to establish world settings. This tool can be used to:

- Establish glass walls (to stop the object(s) falling off the world);
- Set camera mode (fixed, fixed offset, free);
- Set wave height and strength;
- Set sky color (21 options);
- Set day or night time (8 options);
- Set breeze strength;
- Turn on or off the debug tool (which is used to assist in seeing lines of sight and sound);
- Set sound effects volume;
- Set music volume; and,
- Set score visibility (some scores, used as variables by the programmer, may not need to be seen by the user).

3.2: Graphical User Interface

The graphical user interface facilitates game development through the use of a collection of tools in the Tool Menu (Figure 5). It includes: options to add, edit and program objects within the program (Object Tool); three terrain editing tools; a water editor tool; an erase tool; and a general world settings tool. There is also a play game option and a home icon, which enables access to the home menu.

Many options presented to the programmer, including the process of selecting an object to add to the world, are made simple through the provision of selector circle (Figure 6).

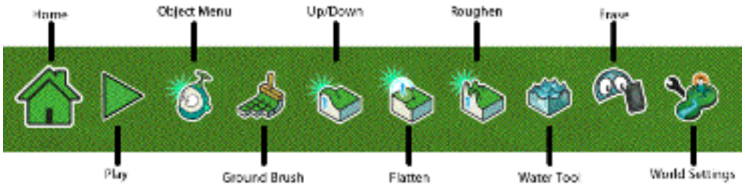

Figure 5: Tool Menu

Figure 6: Insert Object Selector Circle

3.3: Terrain Editors

To build the game world KGL includes three terrain-editing tools and a water tool, with which the user can create his or her own world and environment. The programmer can select from an extensive range of terrain textures (22 choices). The terrain editing tools include a ground brush for creating or editing ground, a flatten ground tool for making the ground smooth or level, and a roughen ground tool for creating spiked or hilly ground.

The water tool enables the programmer to include water in the game world, setting the water level and selecting the color (10 choices).

3.4: Tiles

Users program in KGL by selecting options from the selector wheel and the relevant tile is then inserted into the line of code. Depending on the statement being made, certain tiles can be included in the when or do statement. Some tiles can be included in either of the when or do statements.

Visual cues

Visual cues (Figure 7) are also provided with each tile to give the user a better idea of which functions it will afford. These cues use everyday language and are presented in a user-friendly pictorial format based on the characters and objects in the program.

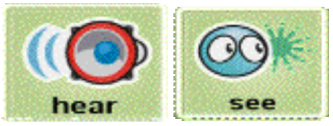

Figure 7: visual cues

3.5: Context sensitive help

To assist the novice programmer, context sensitive help is available. This provides useful and relevant examples of the tiles that could be used and the sequence they need to be placed in. It also includes a detailed description of the functionality these commands will afford (Figure 8). Moreover, KGL is configured to provide suggestions if the programmer appears to hesitate in choosing a tile or option (this facility can be switched off if needed).

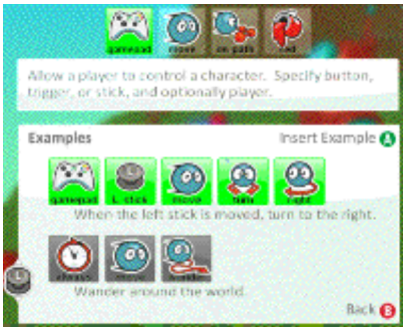

Figure 8: context sensitive help

4: the Kodu programming language

This section looks at how Kodu (the programming language) has been used to teach an eight week programming fundamentals course in a New Zealand middle school. The software has been used to teach students objects, variables, if statements, Boolean logic, inheritance and control flow.

Each Kodu rule has two clauses, a condition (when) and an action (do). The programmer is provided with appropriate tiles to consign to these clauses depending on the actions the game is intended to produce. In the case of the following code (figure 9), in line 1 when the object sees a red apple (condition), it will move toward quickly (action). In line 2, when the object bumps (condition) into a red apple, it will eat (action) the red apple.

Figure 9: example code

4.1: Objects

As noted, each character in KGL is an object. The programmer can allow an object (or instances of an object) to be created or deleted by another character within the game or created during runtime. Objects can also be cloned (or copied) by the programmer, such that each copy retains the code of the source object. Cloning creates a deep copy of an object and any subsequent modification of the code of the source or the copied objects results in different code.

4.2: Variables

The variables in Kodu are implemented as scores (Stolee and Fristoe, 2011). All scores are integers and are global. There are 37 different scores [*op cit*] that can be used by the program which are implemented through the letters of the alphabet (A to Z) or a range of colors. In addition to containing game scores, scores can be used to act as timers or triggers, among other applications.

4.3: Conditionals

Each rule in Kodu is a conditional, which appears to be easily formulated and expressed in the program. Recent trials of the software in a middle school in New Zealand [Fowler & Cusack, 2011] indicate that this was the case.

4.4: Boolean Logic

The Boolean constructs of negation, conjunction, and disjunction are present in Kodu (Stolee and Fristoe, 2011). Typically, Boolean logic is difficult for some students to grasp; however Kodu appears to make learning this easier, an observation borne out by successful use of the program to teach these concepts.

Negation - The not tile

Negation is relatively straightforward in Kodu as a specific tile is available to perform this function – the Not tile. In the case of the following code (figure 10), when the object does not see a puck, it will shoot a cruise missile. However, if the object sees a puck, the do clause will be ignored, so no missile will be fired.

Figure 10: Negation

Indentation

Kodu enables the user to indent lines of code (Figure 11), which creates a logical dependence where the rules of the indented code are evaluated only when the condition of the parent rule has been met.

Figure 11: indented code

In this example (Figure 11), lines two and three are indented under line one, which means they will not be evaluated unless the condition of line one is met (the object bumps into a saucer). Indentation allows for conjunction on the condition or action. In the previous example, the action of line two will only occur if the conditions of lines one and two are met, showing conjunction on the condition. Similarly, if the condition of line one is met, both the actions on lines one and three will occur, showing conjunction on the action.

Disjunction

Logical disjunction can be achieved in Kodu by including two (or more) rules with different conditions but resulting in the same action.

4.5: Classes

Although Kodu does not allow for multiple classes with a special relationship that allows them to reuse functionality between them, it is still possible to introduce students to the concept of inheritance. This can be introduced through establishing an initial character as creatable. The characters and environmental elements need to be set as such by the user and once a creatable object is included in the game world, any other character can create instances of it. These instances then inherit all the properties (including the code) of the original object. In the following figure (Figure 12) the character Sputnik 1 and the environmental elements Mine 1, Light 3, Heart 1 and Apple 1 have been defined by the user as being creatable and can now be created by any other object in the game world.

Figure 12: Creatables

4.6: Pages

Kodu uses the metaphor of pages to describe different states. An object can have up to 12 pages of code or 12 different states. Each character is on one and only one page at any given time and only follows code from its current page. In the following example (Figure 13: “page 1”) on line eight the program will switch to page three if the (red) score is above 50 points, so the character will subsequently only operate according to the code on page three. On line nine the program will switch to page two when the (red) score is below 20 points, so the character will subsequently only operate according to the code on “page two”.

From the code in Figure 13, we can see that when the score is above 52 points, the character will be automatically controlled by the computer to shoot cruise missiles, when he sees the cycle. However, when to score is above 20 points the character will be controllable by the user and when the user presses the A button, cruise missiles will be shot and the character will move forward very quickly when the L stick on the game pad is used.

(Page 1)

(Page 2)

(Page 3)

Figure 13: page references

5: Error handling

One of the major challenges that many beginner programmers face is programming code that is error free. KGL assists the novice programmer by eliminating syntax errors. This is achieved by making available relevant and legal tiles to the programmer. Moreover, should the programmer remove a particular tile from the line of code, KGL will also remove the associated tiles. For example, in the following figure (Figure 14) the programmer has programmed the character to move forward when the user moves the left stick on the gamepad. If the programmer chooses to remove the gamepad tile, KGL will also remove the left stick tile.

Figure 14: example code

6: Comparison

Kodu was developed to introduce programming to novices and there is similar application software that has been developed for the same purpose, including Alice (Dann, et al 2009), and Scratch (Resnick, et al, 2009). However, Kodu is unique in that it provides an integrated game development tool that focuses on engaging novice programmers through an entirely event driven game development system. Moreover, the Xbox 360 controller enables younger users to interact with the system by using a device that they are typically familiar with.

The language paradigm is also unique. The behavior-based approach appears to make learning the concepts of programming easier. Moreover, the 3D isometric graphics also provide an engaging environment that will generally appeal to younger users. Moreover, this software has been recently implemented in a tertiary environment in a vocational college in New Zealand and appears to also appeal to a more mature audience.

KGL is available on both the Xbox 360 and PC, and this makes the software more accessible to a wider audience. While the majority of schools will have Windows compatible PCs, many children will also have a game console at home, thus enabling in-home use.

Furthermore, the community feature of KGL enables users to share their games with each other. This can facilitate shared knowledge and other users can provide the programmer with direct feedback about their games and help with making them better.

7: Effectiveness

KGL is currently available as a technical release and there have been a limited number of studies of the effectiveness in the classroom. However, in a study by the Victorian Department Education and Early Childhood Development in Australia (2010, p6) it was suggested that:

- Kodu Game Lab provided a supportive and productive learning environment by catering for students with diverse needs and supporting collaborative learning;
- Kodu Game Lab promoted independence, interdependence and self-motivation through the use of cooperative learning groups to develop the games and rubrics to guide them in their learning;
- Students' diverse needs, backgrounds, perspectives and interests were reflected in the learning program as they used their knowledge, understanding and experience of computer games from home;
- deep thinking and application were supported and developed through the game making process and the creation, design and deconstruction of multimedia texts, and students engaged in problem solving and creative thinking;
- the learning connected strongly with the community and practice beyond the classroom, through the use of technologies that are also used beyond the classroom setting and through

involvement in the Planet Kodu wiki, which allowed students to communicate and collaborate with a global audience; and,
- assessment practices, including peer assessment, were an integral part of the teaching and learning.

Moreover, in a study in a New Zealand middle school Fowler and Cusack (2011, p77) found that:

- when compared to similar students who were taught more traditional programming languages (C++, Java) the levels of enjoyment and engagement were observed as being significantly higher;
- conversely, levels of boredom or frustration were observed as being significantly lower;
- students also did their learning with higher motivation levels; and,
- students had the embedded feedback of producing a working and functional game that built confidence and a sense of achievement in classroom learning.

Although these findings need further support and the software needs to be finalized, it is apparent that there is potential for Kodu Game Lab to be an effective and engaging educational tool.

8: Conclusions

KGL was developed to assist and enable novice programmers to develop their own 3D games and learn some programming principles. The graphical user interface facilitates ease of use, and the interactive tutorial system, prebuilt playable games and contextual help all serve to provide a progressive and active learning context. The programming language is close to a natural language, making it easy to learn.

Kodu Game Lab and the Kodu programming language provide an integrated system that is engaging, intuitive and easy to learn.

References

Apiola, M. Lattu, M. and Pasanen, T. A. 2010. Creativity and intrinsic motivation in computer science education: experimenting with robots. *Proceedings of the fifteenth annual conference on Innovation and technology in computer science education*, ser. ITiCSE '10. New York, NY, USA: ACM, pp199–203.

Cheung, C.Y., Ngai, G., Chan, C.F., & Lau, W. W. Y. 2009. Filling the gap in programming instruction: a text-enhanced graphical programming environment for junior high students. *Proceedings of the 40th ACM technical symposium on Computer science education* (SIGCSE '09). ACM, New York, NY, USA, 276-280. Available from: http://doi.acm.org/10.1145/1508865.1508968

Dann, W. Cooper, S. Pausch, R. *Learning to program with Alice* (2nd Ed.), Upper Saddle River, NJ. Copyright ©2009 Pearson Education

Fowler, A. and Cusack, B. 2011. Enhancing Introductory Programming with Kodu Game Lab: An Exploratory Study, *Proceedings of the 2nd Annual Conference of Computing and Information Technology Education and Research in New Zealand*, (Eds.) S, Mann. & M, Verhaart, (pp69-79), CITRENZ, Hamilton, New Zealand

Klassner, F., and Anderson, S.D. 2003. LEGO Mindstorms: Not just for K-12 anymore. *IEEE Robotics and Automation Magazine* (February 2010), pp12–18

Lund, H. H. and Pagliarini, L. 2000. RoboCup Jr. with LEGO Mindstorms. *Proceedings of ICRA2000*, New Jersey, IEEE Press

Resnick, M., Maloney, J., Monroy-Hernandez, Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., and Kafai, Y. 2009. Scratch: Programming for all. *Communications of the ACM*, ACM, 52, 11, pp60-67

Stolee, K. T. 2010. *Kodu language and grammar specification*.
Available from: http://research.microsoft.com/en-us/projects/kodu/kodugrammar.pdf

Stoolee, K. T. and Fristoe, T. 2011. Expressing computer science concepts through Kodu game lab. *Proceedings of the 42nd ACM technical symposium on Computer science education* (SIGCSE '11). ACM, New York, NY, USA, pp99-104

NB

MICROSOFT: **Kodu Game Lab** (download details):

http://www.microsoft.com/downloads/en/details.aspx?FamilyID=57a23884-9ecd-4c8a-9561-64bfd4fa2d3d&displaylang=en