Journal of Software Engineering and Applications, 2014, 7, 177-186
Published Online March 2014 in SciRes. http://www.scirp.org/journal/jsea
http://dx.doi.org/10.4236/jsea.2014.73019

Scientific Research

Islay3D—A Programming Environment for Authoring Interactive 3D Animations in Terms of State-Transition Diagram

Dandy Ling Kwong[1], Michitoshi Niibori[1], Shusuke Okamoto[2], Masaru Kamada[3], Tatsuhiro Yonekura[3]

[1]Graduate School of Science and Engineering, Ibaraki University, Hitachi, Japan
[2]Graduate School of Science and Technology, Seikei University, Musashino, Japan
[3]Department of Computer and Information Sciences, Ibaraki University, Hitachi, Japan
Email: dandyling@gmail.com, niibori@gmail.com, okam@st.seikei.ac.jp, m.kamada@mx.ibaraki.ac.jp, yone@mx.ibaraki.ac.jp

Received 17 February 2014; revised 11 March 2014; accepted 18 March 2014

Abstract

An educational programming language is a programming language that is designed primarily as a learning instrument and not so much as a tool for writing programs for production. Three-dimensional (3D) interactive animations provide an effective means to engage the attention of the audience to learn programming language. Traditionally, creating 3D games had been difficult as it requires specialized programming skills. However, it had been proven that the state-transition diagram, which is the most fundamental principle for automata, is intuitively so comprehensive that even children can create programs for interactive animations and video games in the two-dimensional world. Islay3D is a programming environment for authoring interactive 3D animations based on this concept. In this paper, the Islay3D animation language is introduced, where a character is modeled as an object, and its behavior is defined in term of a set of state-transition diagrams. The interpretation of the state-transition diagrams to JavaScript is also presented. Finally, the web-based programming environment is introduced. With the web-based platform, the public will be able express their creativity in creating interactive 3D animations and video games easily from within their browser.

Keywords

3D; Visual Programming Language; Computer Games; State Transition Diagram; Web Service

How to cite this paper: Kwong, D.L., *et al*. (2014) Islay3D—A Programming Environment for Authoring Interactive 3D Animations in Terms of State-Transition Diagram. *Journal of Software Engineering and Applications*, **7**, 177-186.
http://dx.doi.org/10.4236/jsea.2014.73019

1. Introduction

An educational programming language is a programming language that is designed primarily as a learning instrument and not so much as a tool for writing programs for production. An example is Scratch [1], which allows users to create interactive animations by arranging programming statements made up of reed-shape building blocks, without the worry of making syntax error.

Three-dimensional (3D) interactive animations provide an effective means to engage the attention of the audience to learn programming language, as the vast availability of gaming consoles such as Xbox, PlayStation, and Nintendo's 3DS allows children to get exposure to 3D games from an early age. Educational programming language based on 3D interactive animations is an attractive feature for children who want to learn how to create and play their own computer games.

Traditionally, creating 3D games had been difficult as it requires specialized programming skills. However, it had been proven that the state-transition diagram, which is the most fundamental principle for automata, is intuitively so comprehensive that even children can create programs for interactive animations and video games in the two-dimensional world [2]. Islay3D is a programming environment for authoring interactive 3D animations based on this concept.

Islay3D was first developed on the MFC framework and run on Microsoft Windows. To reach a wider user base, it was rewritten for use on the web using JavaScript and WebGL. In this paper, the Islay3D animation language is introduced, where a character is modeled as an object, and its behavior is defined in term of a set of state-transition diagrams. The interpretation of the state-transition diagrams to JavaScript is also presented. Finally, the web-based programming environment is introduced.

2. Islay3D Programming Language

The visual programming language of Islay3D is one where the behavior of a character is described in terms of state-transition diagram. **Figure 1** shows a robot with a set of state-transition diagrams. A state is represented by a circle, showing the action for the character to take in that state. A transition represented by an arrow means that the character shifts its state to the pointed one on condition written along the arrow.

In **Figure 1**, the state-transition diagram is defined so that the robot turns left or right when arrow keys are pressed. It is also possible to attach other diagrams to the same character. Attaching another diagram for moving forward or backward, for example, we can make the robot moving around a circular path.

A character can be recursively composed of lower-level characters as illustrated in **Figure 2**. Each character encapsulated in an orange oval has its shape in the blue oval and state-transition diagrams marked in yellow. The robot character on the top level has its appearance composed of other characters representing its head, body,

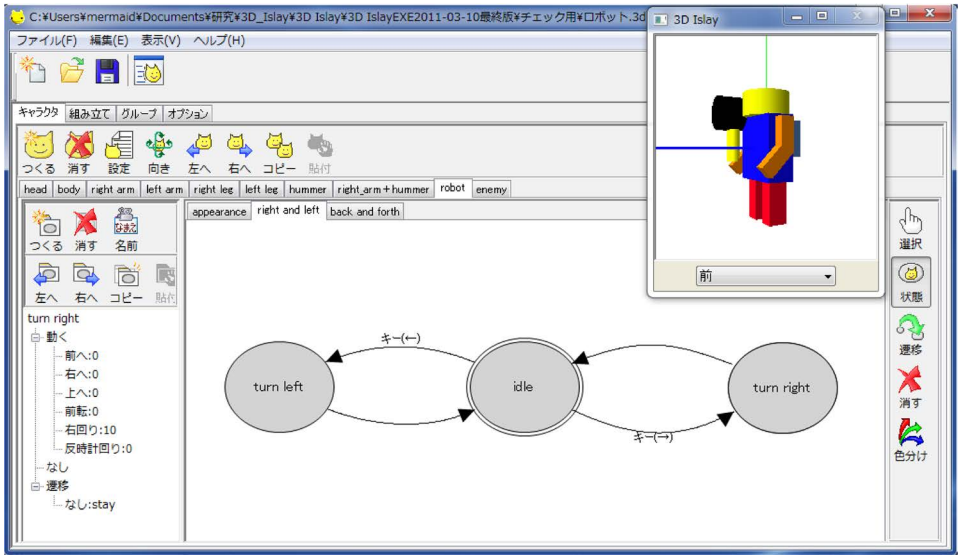

Figure 1. The character will turn left when the left keyboard key is pressed, and vice versa.

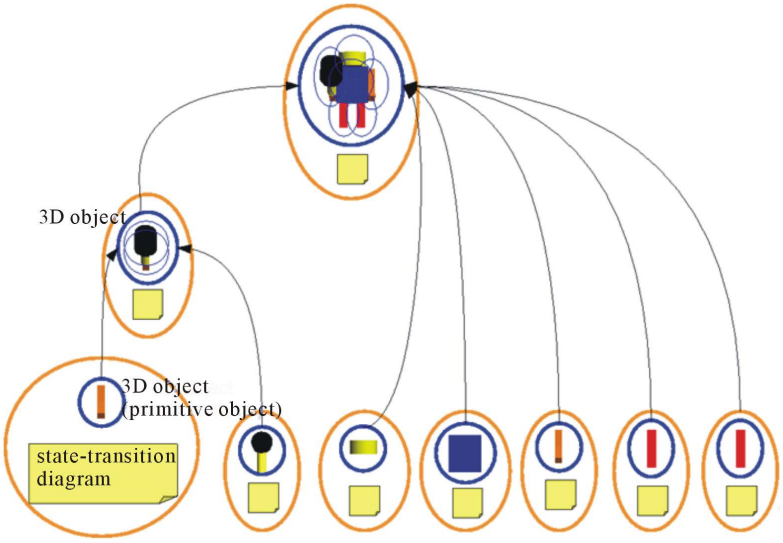

Figure 2. Hierarchical structure of characters.

right and left arms, and legs. This robot actually has a hammer in the right arm so that the right arm is defined as a higher-level character composed of an arm character and a hammer character. The 3D objects representing the bottom-level characters can be called the atomic 3D objects.

This hierarchical structure is nothing special but a standard model for composing 3D objects. The lower-level characters behave relative to the local coordinate system of the upper-level character. The top-level characters behave relative to the global coordinate system [3].

2.1. Actions

In a particular state, the character executes one or more actions. The actions are divided into two categories—1) geometrical translation actions, and 2) logical actions.

Geometrical translation actions allow the character to perform in-game physical movements. The actions are such as translation and rotational movements. It also includes the positioning of the character at a specified or random position. It is also possible to reset the action of the character to its default condition at the start of the game. **Table 1** shows a list of the geometrical translation actions.

Logical actions allow the character to perform game actions. For example, a character may be able to fork into other character. A character may also be able to send messages to other characters, such that other characters may respond to the message. **Table 2** shows a list of the logical actions.

2.2. Transition Conditions

After executing the action in a state, the transition condition is evaluated. If the condition is true, the current state of the character will shift to the next state in the direction indicated by the arrow.

The transition conditions range from user input such as mouse clicks and keyboard, to game conditions such as collision with other character. There are also transition conditions based on functions such as probability and state repetition. The list of transition conditions is shown in **Table 3**.

The combination of the execution of the state, and the evaluation of the transition condition, forms a control flow which defines the animation behavior of the 3D character in Islay3D. This control flow which is represented as circles and arrows in the state-transition diagram forms the core of the visual programming language in Islay3D.

3. Interpretation to JavaScript

The web-based execution model consists of two modules, the graphical editor to draw the state-transition diagrams, and the player which runs the output 3D interactive animations. The graphical editor produces an XML

Table 1. Geometrical translation actions [4].

Geometrical Action	Description
Idle	Not moving
Move	Move or rotate according to the parent character coordinate system
Navigate	Move or rotate according to the world coordinate system
Position	Specify position for the character to appear at
Random Jump	Position the character at random coordinates
Reset	Reset the movement and/or rotation of the character

Table 2. Logical actions [4].

Logical Action	Description
Broadcast	Send message to all characters
Upcast	Send message upwards in the character hierachical structure
Downcast	Send message downwards in the character hierachical structure
Group Fork	Fork a specified group of characters
Transform	Transform to a specified character
Hide	Make the character invisible
Show	Make the character visible
Disappear	Remove the character from the game
The End	End the game

Table 3. Transition conditions [4].

Transition Condition	Description
Keyboard Key	Transition when a particular keyboard key is pressed
Click	Transition when the character is clicked
Bump	Transition when the character collide with other characters
Message	Transition when the character receive a message from broadcast, upcast, or downcast action
Probablity	Transition when a specified probability occured
Timeout	Transition after the current state is repeated a specified times
Default	When other transition conditions are not fulfilled

file as shown in **Figure 3**. This XML file is an internal representation of the state transition diagram drawn by the user, defined as per the animation definition of Islay3D elaborated in [3]. It contains a list of states, which has actions to be taken in those states. It also contains a list of transitions, which is a triple of the transition condition, source and destination states. The player takes these as input and produces the 3D animation output.

The player is developed with enchant.js [5], an open source framework for developing simple games and applications in HTML5 + JavaScript, with support for WebGL using a plugin [6]. The player is an interpreter, which takes the input XML file and calls the corresponding JavaScript functions during run time.

Figure 4 shows a snippet of the state execution module in the interpreter. The green block shows the parsing of the state information from the XML statement. The red block then shows the corresponding enchant.js function calls are called. **Figure 5** shows a code snippet for the transition module. The transition module contains a *current state* pointer which points to the current state. On every frame change, the state execution module executes the *current state*, then transition module updates the *current state* to point to the next state. This is then repeated to produce a 3D animation.

```
<statediagram name="Diagram 1">
  <statelist>
    <state name="Right" action="move" front="0" right="1" up="0"
           pitch="0" yaw="0" roll="0" pos_x="774" pos_y="391"/>
    <state name="Idle" action="stay" pos_x="627" pos_y="390"/>
    <state name="Left" action="move" front="0" right="-1" up="0"
           pitch="0" yaw="0" roll="0" pos_x="485" pos_y="396"/>
  </statelist>
  <translist>
    <trans guard="default" from="Right" to="Idle"/>
    <trans guard="key" key="RIGHT" from="Idle" to="Right"/>
    <trans guard="key" key="LEFT" from="Idle" to="Left"/>
    <trans guard="default" from="Left" to="Idle"/>
  </translist>
</statediagram>
```

Figure 3. State transition diagram represented as XML.

```
executeActionType1 = function(character, state) {
    if(state.attributes["action"].value == "move") {
        var tZ = parseFloat(state.attributes["front"].value);
        var tX = parseFloat(state.attributes["right"].value);
        var tY = parseFloat(state.attributes["up"].value);
        var rZ = parseFloat(state.attributes["pitch"].value);
        var rY = parseFloat(state.attributes["yaw"].value);
        var rX = parseFloat(state.attributes["roll"].value);

        character.sidestep(tX);
        character.altitude(tY);
        character.forward(tZ);
        character.rotateRoll(rX*(Math.PI / 180));
        character.rotateYaw(rY*-(Math.PI / 180));
        character.rotatePitch(rZ*(Math.PI / 180));
    }
```

State information

Execute action

Figure 4. Code snippet of state execution module.

```
case 'prob':
  var prob = tran.attributes["prob"].value;
  if (Math.random() < parseFloat(prob) / 100) {
    character.XML.STATEDIAGRAMS[a].current =
        character.XML.STATEDIAGRAMS[a].STATES[to];
  }
  break;

case 'click':
  if (character.touched) {
    character.XML.STATEDIAGRAMS[a].current =
        character.XML.STATEDIAGRAMS[a].STATES[to];
    character.touched = false;
  }
  break;
```

Figure 5. Code snippet of transition module.

4. Programming Environment

In this section, the user interface of the web version of Islay3D is introduced.

4.1. 3D Characters

At the start of Islay3D, the user chooses a 3D model from the character selection panel. This is shown in **Figure 6**. Hovering over the model will show a tooltip that shows a description of the model. All the characters are programmable by using the state-transition diagram based visual language described in Section II. It is possible to set the character to be invisible when first inserted into the game.

The user also has the ability to import their own model files (COLLADA dae [7]) into their game.

4.2. Diagram Editor

Figure 7 shows the Diagram Editor where the user draws their state transition diagram. The user draws the diagram by using four tools: 1) the *pointer tool*, 2) the *circle tool*, 3) the *arrow tool*, and 4) the *erase tool*.

By selecting the *circle tool*, clicking anywhere in the canvas area of the Diagram Editor will create a circle. Two circles maybe connected together with an arrow by dragging from one circle to another using the *arrow tool*. The *erase tool* can be used to erase any circles or arrows to make corrections. The *pointer tool* is used to adjust the positions of the circles in the diagrams.

The user can create more than one diagram for a character by using the tab interface, where each tab corresponds to a state-transition diagram.

4.3. Toolbox

To define the action for the circles, an action toolbox will pop up whenever a circle is clicked. **Figure 8** shows the toolbox, with the geometrical actions grouped into *move*, *jump*, and logical actions grouped into *message*, *fork*, *change*, *transform*. Further clicking on the toolbox brings up a dialog box, allowing the user to define the parameter for the action. **Figure 8** shows the *move* setting dialog box, where the user will be able to enter the movement and rotation value for the x, y, and z axis.

A corresponding transition toolbox shows up when an arrow is clicked, with the list of transition options as listed in **Table 3**.

4.4. Previewer

While setting geometrical translation actions as listed in **Table 1**, the user will be able to preview immediately the effect to the character. This allows the user to perform on-the-fly manipulation, without having to run the game to confirm the movement of the character. **Figure 7** shows the previewer in the top right corner of the diagram editor.

4.5. Step-by-Step Tutorial

Figure 9 shows a step-by-step tutorial is provided when the program first started up. This tutorial will guide the users to use the features in the program, as well as teaching them the concept of state-transition diagram in Islay3D. This is done by walking through the users through creating a simple interactive animation using one of the characters in the game.

4.6. Player

When the user has created their game, they can run it anytime by using the *Run* button. This creates a new window as shown in **Figure 10** where the user will be able to play their game. When finished playing the game,

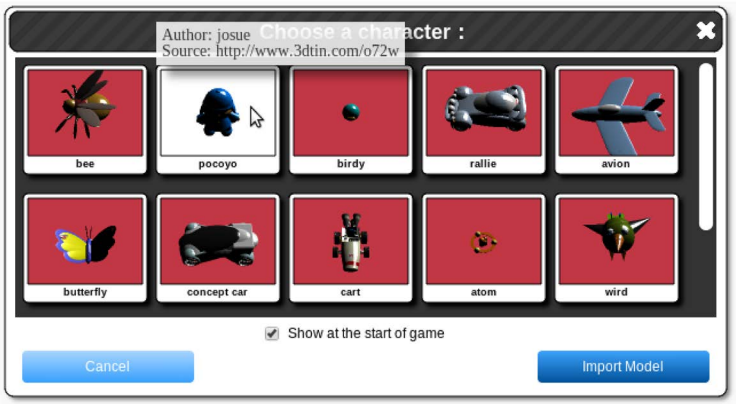

Figure 6. 3D model files in the character selection panel*. *3D model courtesy of user Josue, as well as other users for models from 3DTin.com.

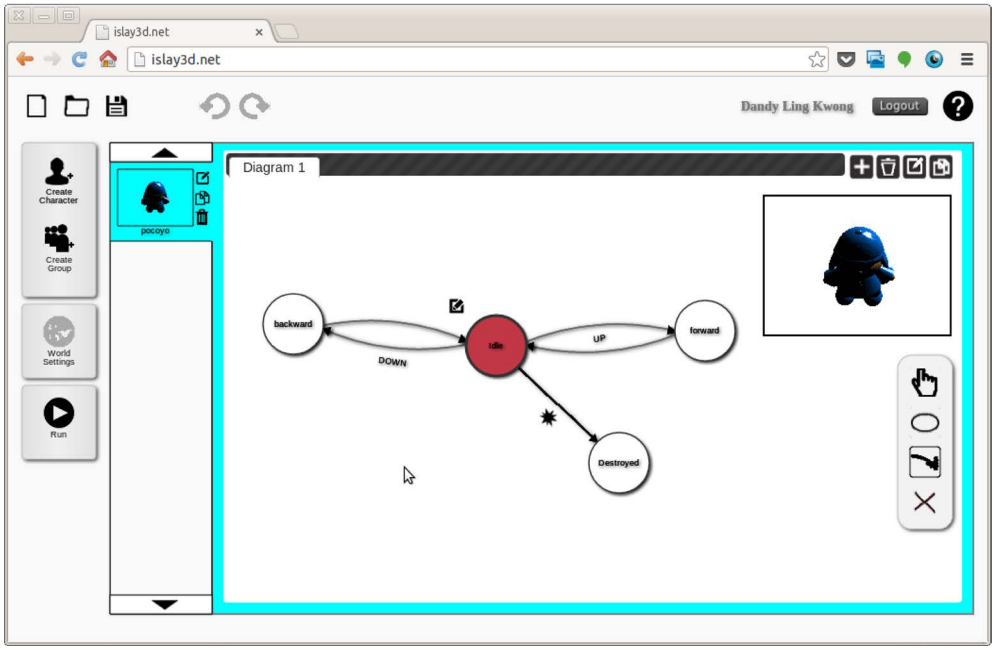

Figure 7. Diagram editor where the user draws the state-transition diagram.

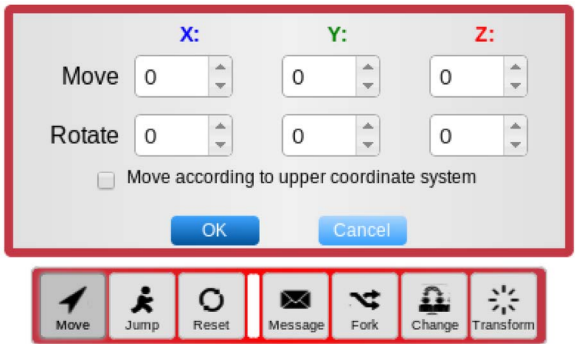

Figure 8. The action toolbox with the *move* setting dialog box.

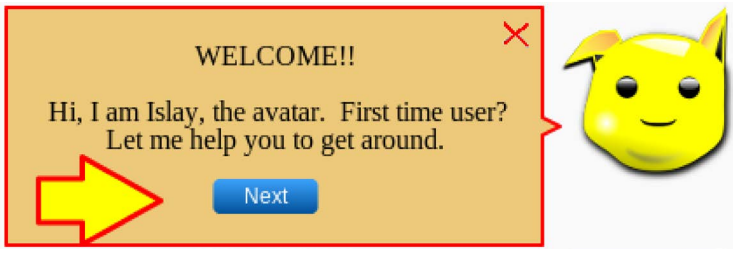

Figure 9. A tutorial which guides the users step-by-step.

they can close the window, and get back to the graphical editor to continue editing their game with ease.

5. Related Works

Figure 11 shows Alice [8], a programming environment designed to introduce student to the concept of object-oriented programming by creating 3D animated movies and simple video games. In Alice, students create a virtual world by using with a 3D scene editor, and the 3D objects in the scene are animated by creating a program

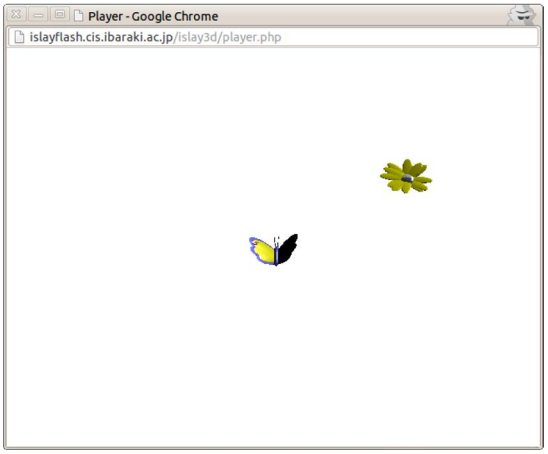

Figure 10. Player which runs the game in a new window.

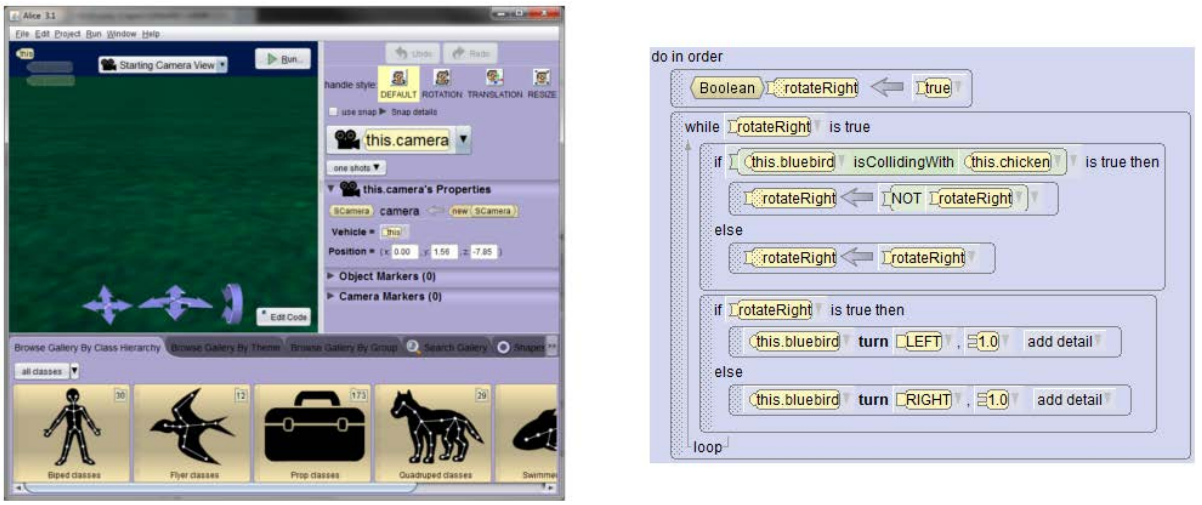

Figure 11. Scene editor and scripting language in Alice.

through selecting statements corresponding to production-oriented programming language such as Java, C++, and C#. Learning a scripting language is required because it is the very purpose of Alice to educate non-programmers to be programmers. On the other hand, Islay3D does not requires the user to write text-based programs, and they only need to draw a set of state-transition diagrams.

Kodu Game Lab [9] is a tile-based visual programming tool that enables users to learn programming concepts through making and playing computer games in real-time isometric 3D gaming environment. It is designed specifically for young children, with 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.

Kodu's programming language is a high-level visual language that can be represented as a context-free grammar. 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 as shown in **Figure 12**. Each Kodu rule has two clauses, a condition and an action, which is similar to Islay3D's transition and state.

However, the language paradigm of Islay3D has the advantage in that the user is able to represent the control flow of the program visually and comprehensively. This is possible through the usage of states, where each character can only be at one state at any given time in a state-transition diagram. Contrasting this to the rule-based event-driven programming environment such as Kodu, this allows user to learn easily the fundamental concept of serial control flow which forms the building block of programming language [10].

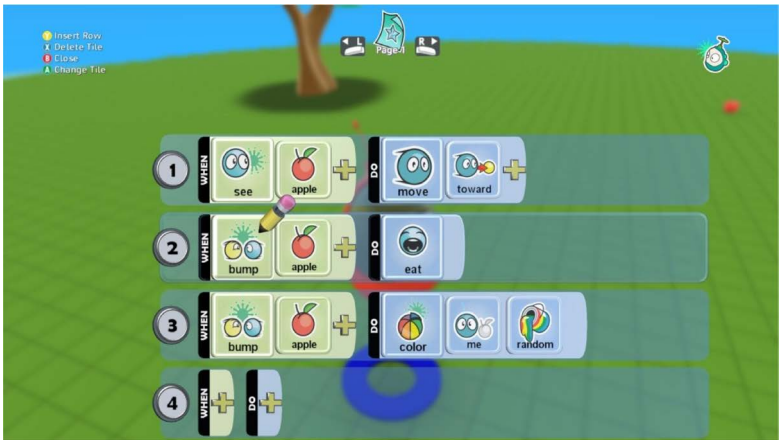

Figure 12. Programming interface in Kodu.

6. Evaluation

The web version of Islay3D is still under development and there have been limited number of studies on the effectiveness of the tool. In an evaluation of the basic function of the web version Islay3D which were carried out with a number of elementary and secondary school students in the local community, it was suggested that:

1) The concept of using state-transition diagram for animating character is easy to learn for the students after being given a tutorial.

2) The students enjoyed expressing their creativity in the manipulation of 3D characters, and would like to recommend and play Islay3D with their friends.

7. Conclusions

Islay3D was developed to allow novice programmer to create 3D interaction animations without the need to learn a text-based scripting language. The Islay3D programming language allows user to learn about automation through the serial control flow which is visually comprehensive in a state-transition diagram, the learning of which forms the building block for learning programming languages.

The implementation and the graphical interface of the web-based tool is presented. The web-based programming environment facilitates ease of use by providing easy accessibility within a browser, and step-by-step tutorial to guide first time users. It is available now at islay3d.net.

User-generated content is an important feature for users to express their creativity in artistic endeavor. As of the time of writing, Islay3D still doesn't support a 3D world editor for the users to edit the game world, and a 3D model editor for the users to create their own 3D models. Integration of the later with the behavior definition of complex character is a potential future work.

As Islay3D was traditionally developed on the Windows platform, more usability studies is recommended to investigate it's usage on the web. The comprehensiveness of the state-transition diagram in describing 3D animations compared to its 2D counterpart is also recommended.

Acknowledgements

The authors would like to thank Satomi Sugai, NaoyukiSone, Hideki Kotani, KatsuhisaKanno, MakotoRokujo for their pioneering work in the 3D version of Islay.

References

[1] Brennan, K., Monroy-Hernndez, A. and Resnick, M. (2009) Scratch: Creating and Sharing Interactive Media. *Proceedings of the* 9*th International Conference on Computer Supported Collaborative Learning* (*CSCL'*09), Rhodes, 8-13 June 2009, 217. http://dx.doi.org/10.3115/1599503.1599576

[2] Okamoto, S., Kamada, M. and Nakao, T. (2005) Proposal of an Interactive Animation Authoring Tool Based on State-Transition Diagram. *IPSJ Transactions on Programming*, **46**, 19-27.

[3] Rokujo, M., Niibori, M., Okamoto, S., Kamada, M. and Yonekura, T. (2012) Authoring Tool for Flash 3D Animations in Terms of State-Transition Diagrams. *Proceedings of the* 15*th International Conference on Network-Based Information* (*NBIS* 2012), Melbourne, 26-28 September 2012, 889-892.

[4] Kanno, K. (2010) Prototyping Tool for Three-Dimensional Video Game Characters in Terms of State Transition Diagrams. Master Dissertation, Ibaraki University, Ibaraki, Japan.

[5] Enchant.js (2014) A Simple JavaScript Framework for Creating Games and apps. http://enchantjs.com

[6] Enchant.js. Plugins (2014) Enchant.js' Plugins. http://wise9.github.io/enchant.js/doc/plugins/en/index.html

[7] COLLADA (2014) Digital Asset and FX Exchange Schema. https://collada.org/

[8] Cooper, S., Dann, W. and Paush, R. (2003) Teaching Objects-First in Introductory Computer Science. *ACM Proceedings of the* 34*th SIGCSE Technical Symposium on Computer Science Education*, Reno, 19-23 February 2003, 191-195.

[9] Fowler, A., Fristace, T. and MacLauren, M. (2012) Kodu Game Lab: A Programming Environment. *The Computer Games Journal*. http://tcjg.weebly.com/fowler-et-al.html

[10] Kwong, D.L., Dandy K., Niibori, M., Okamoto, S., Kamada, M. and Yonekura, T. (2013) Web-Based Tool for Programming Interactive 3D Animations in Terms of State-Transition Diagrams. 16*th International Conference on Network-Based Information Systems* (*NBiS*), Gwangju, 4-6 September 2013, 453-458.