Journal of Software Engineering and Applications, 2016, 9, 547-560
http://www.scirp.org/journal/jsea
ISSN Online: 1945-3124
ISSN Print: 1945-3116

Translator of Islay 3D Animations into Flash Platform

Michitoshi Niibori[1], Makoto Rokujo[1], Shusuke Okamoto[2], Masaru Kamada[3], Tatsuhiro Yonekura[3]

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

How to cite this paper: Niibori, M., Rokujo, M., Okamoto, S., Kamada, M. and Yonekura, T. (2016) Translator of Islay 3D Animations into Flash Platform. *Journal of Software Engineering and Applications*, **9**, 547-560.
http://dx.doi.org/10.4236/jsea.2016.911037

Received: August 24, 2016
Accepted: November 6, 2016
Published: November 9, 2016

Abstract

The three-dimensional (3D) interactive animations and video games are so attractive that successful educational programming environments like Alice and Kodu Game Lab deal with 3D characters. Islay 3D is another educational programming environment of which feature is an intuitively comprehensive user interface in terms of state transition diagrams. Unfortunately its animation definition was too memory-hungry when played by the built-in interpreter. In this paper, we present a translator of the animation definitions of Islay 3D into ActionScript3 (AS3). Compiling the AS3 codes by way of Papervision3D, we obtain the 3D Flash animation file playable on the Flash platform. It will be shown that the memory and CPU usages will be much saved, quartered and halved, respectively, by virtue of the translator.

Keywords

Interactive Animations, State Transition Diagrams, Flash

1. Introduction

The three-dimensional (3D) modeling is an effective means to enhance the impression of interactive animations and video games at less cost. Adobe Flash Player [1] allows for 3D contents by way of libraries like Papervision3D [2] and is a major platform for movies, interactive contents and video games on the web.

Adobe Flash Professional [3] is a genuine tool for authoring any kinds of Flash contents. But users have to write programs in the ActionScript language in order to make the movies as interactive as video games. There have also been educational programming environments that allow even children to work on their own dynamic 3D contents. Alice [4] lets children create a virtual 3D world by its scene editor and give the 3D objects in the virtual world to behave according to their program written in its script programming language. Kodu Game Lab [5] is a visual programming environment that lets children describe the behavior of the game characters in terms of event-driven sequences of actions represented by tiles.

It has 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 (2D) world [6]. This programming environment has been distributed with its name *Islay*. Its 3D version named *Islay* 3D [7] is an easy-to-use programming environment for authoring interactive 3D animations in terms of state-transition diagrams. It is something between Alice and Kodu Game Lab. Islay 3D is easier to use than Alice because of the state-based programming interface although Alice's script language is, of course, more powerful. Islay 3D allows for description of behavior explicitly bundled to the states while Kudo's characters are basically stateless although it is possible for Kudo to represent (at most 12) states by the concept of pages where the tiles belong to.

Islay 3D has been developed first as a Microsoft Foundation Class (MFC) application for Microsoft Windows and then rewritten in JavaScript to work on web browsers. The web version of Islay 3D plays the animation by an interpreter written in JavaScript to interpret the animation definition. It works fine on fast computers but is too memory-hungry to run on smaller computers.

In the 2D world, employing Islay as the front-end user interface, tools for authoring interactive Flash animations have been developed for ActionScript2 [8] and also for ActionScript3 (AS3) [9].

In this paper, we present a translator of Islay 3D animations into AS3. Compiling the AS3 codes by way of Papervision3D [2], we obtain the 3D Flash animation file playable on the Flash platform. Its effectiveness will be evaluated in the aspect of memory and CPU usages saved by virtue of the translator.

2. Islay 3D

Islay3D [7] is a graphical programming environment where the behavior of animation characters is described in terms of state-transition diagrams. A character can be recursively composed of lower-level characters that behave in the local coordinate system of the upper-level character. It is also possible to attach several state transition diagrams to a character.

2.1. Definition of Behavior in Terms of State-Transition Diagrams

Behavior of animation characters is defined by describing state-transition diagrams on the editor. **Figure 1** is a screenshot of the editor. A state represented by an oval has actions for the character to take in the state. Actions are selected out of geometric actions such as parallel translations and rotations or logical ones such as forking other characters. 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 diagram is attached

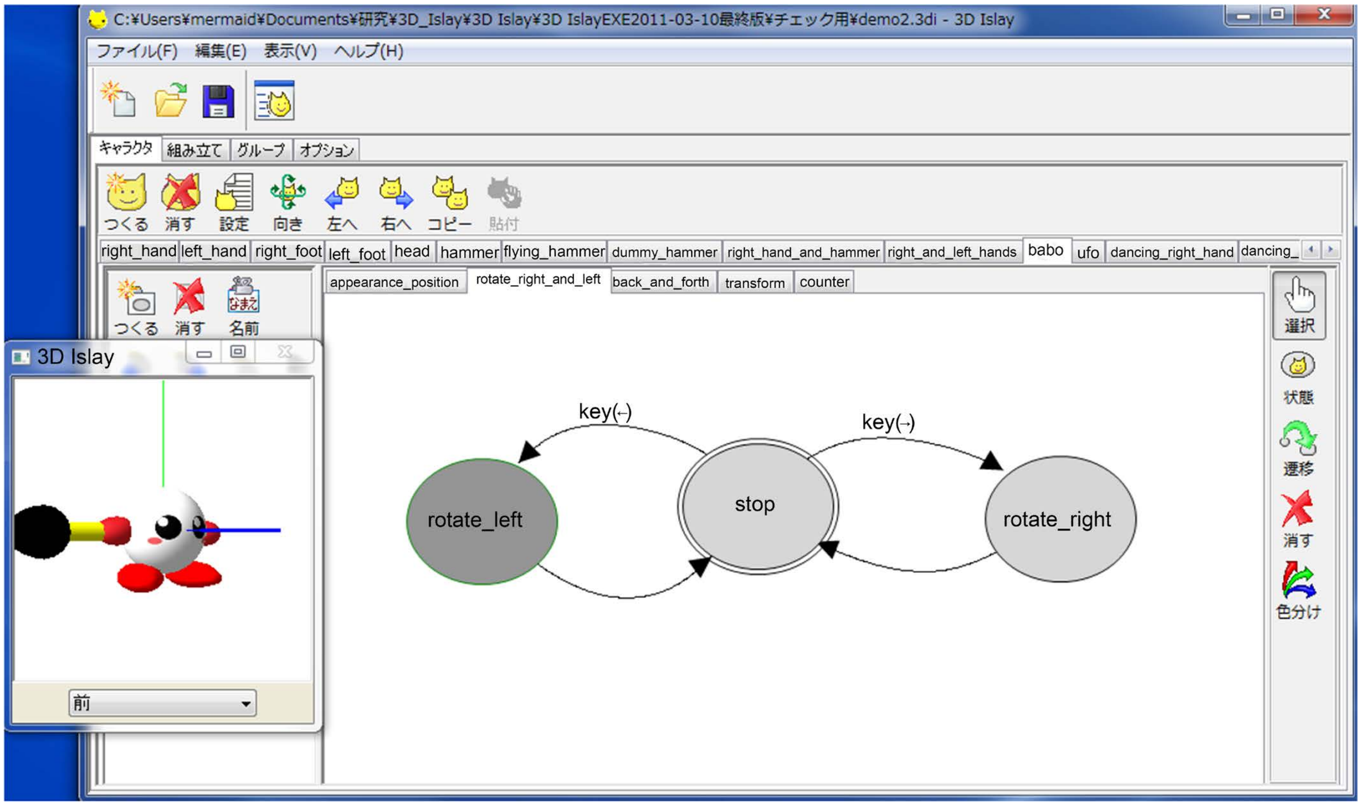

C:¥Users¥mermaid¥Documents¥研究¥3D_Islay¥3D Islay¥3D IslayEXE2011-03-10最終版¥チェック用¥demo2.3di - 3D Islay
ファイル(F) 編集(E) 表示(V) ヘルプ(H)
キャラクタ 組み立て グループ オプション
つくる 消す 設定 向き 左へ 右へ コピー 貼付
right_hand left_hand right_foot left_foot head hammer flying_hammer dummy_hammer right_hand_and_hammer right_and_left_hands babo ufo dancing_right_hand dancing_
つくる 消す 名前
appearance_position rotate_right_and_left back_and_forth transform counter
3D Islay
前
key(←)
key(→)
rotate_left
stop
rotate_right
選択
状態
遷移
消す
色分け

Figure 1. Screenshot of the editor.

to the character so that it turns left or right when an arrow key is pressed.

2.2. Hierarchical Structure of Characters

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 definition of behavior represented by the yellow box being a set of state-transition diagrams.

The original character on the top level has its appearance composed of other characters representing its head, body, right and left hands, and feet. This character has a hammer in the right hand so that the right arm is defined as an upper-level character composed of an arm character and a hammer character. The 3D objects representing the bottom-level characters are called atom 3D objects hereafter.

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.

2.3. Actions and Events

Actions are selected out of geometric actions such as parallel translations and rotations or logical ones such as forking other characters and sending messages. A state can have

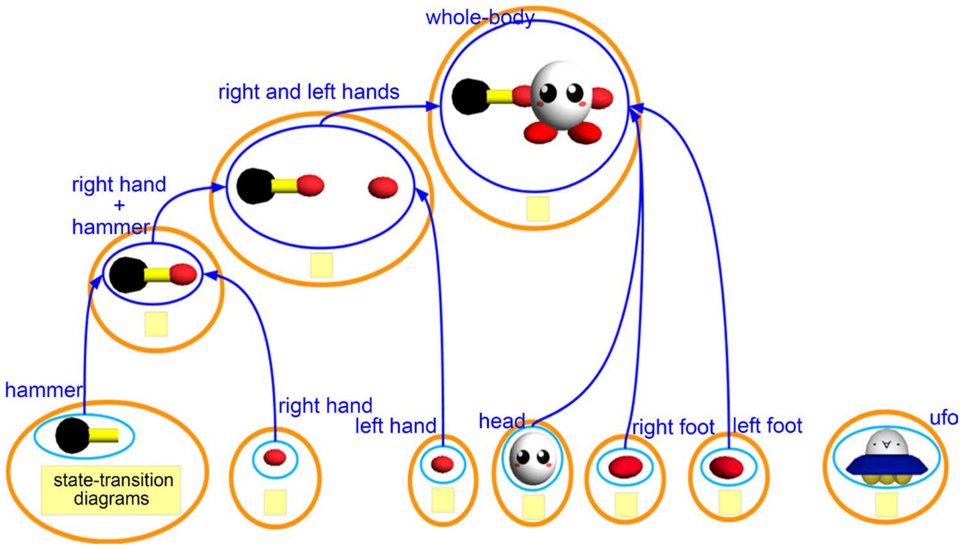


Figure 2. Hierarchal structure of characters.

both geometric action and logical action.

Events include key press, messages, and collisions with other characters or walls that condition transition of the states.

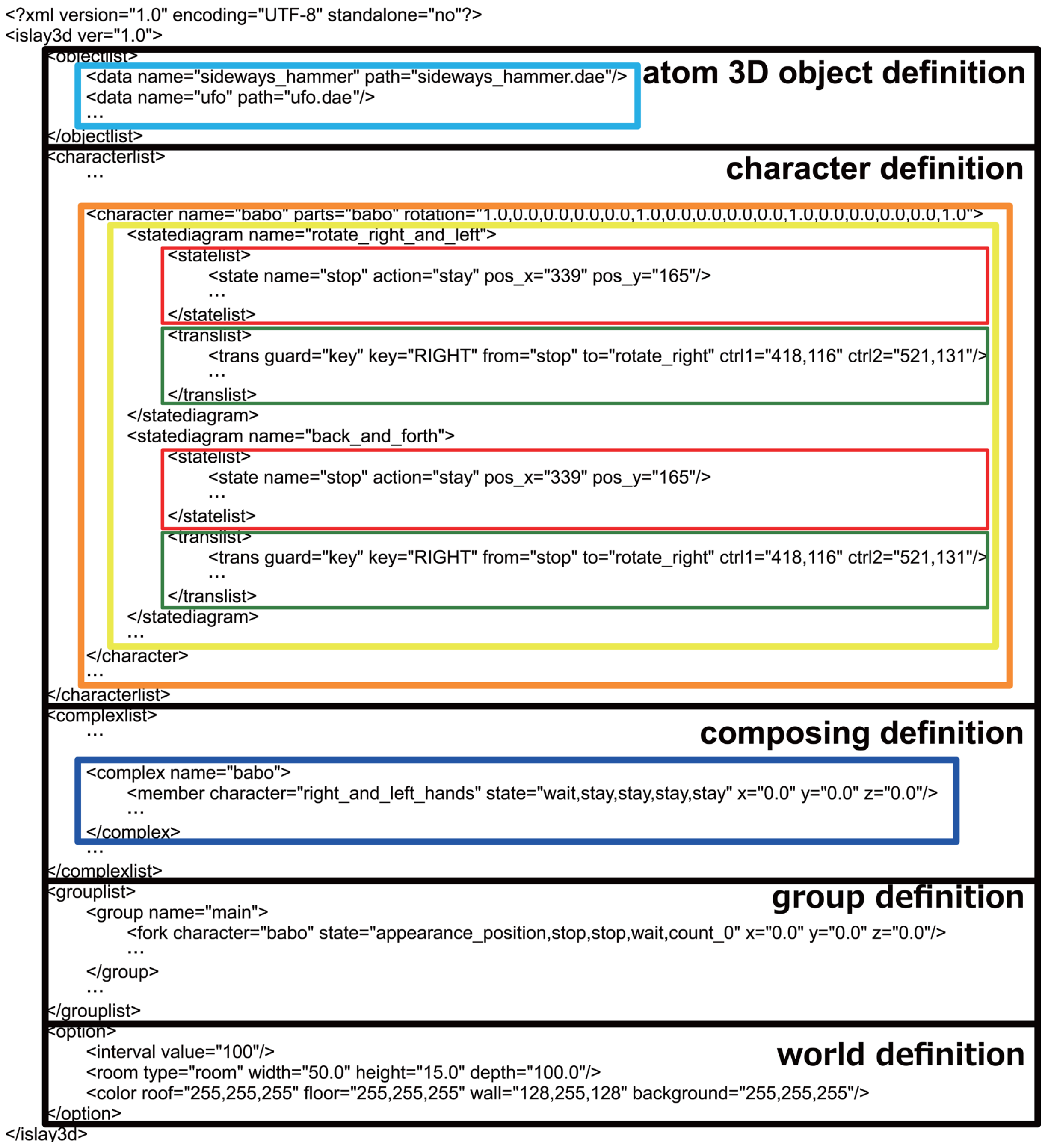
2.4. Data Structure of Animation Definition

The definition of animation includes the shapes of atom 3D objects, character definitions composed of state-transition diagrams and references to lower-level characters with their initial relative positions, and groups of characters to be forked at the same time. Those data are stored in an XML-based format as their example in Figure 3.

An atom 3D object's shape is expressed in a digital asset exchange (DAE) file in the Collada format [10] to be referred to by its file name. Its example is the light blue block in Figure 3.

A character definition in the orange block of Figure 3 has a name, a set of state-transition diagrams as in the yellow block of Figure 3. A bottom-level character has its appearance specified by a DAE file name in the light blue block in Figure 3. An upper-level character has its appearance specified by a list of lower-level character names and their relative initial positions in the blue block of Figure 3. Figure 4 is an expanded form of the yellow block in Figure 3. Diagram 1 is the expression of the state-transition diagram in Figure 1 that specifies horizontal rotation of the character according to the left and right arrow keys. Diagram 2 is the one for moving forward or backward according to the up and down arrow keys. A state-transition diagram in the yellow block has the orange block where names of states are listed together with the actions to be taken in the state. In the green block, transitions are listed each of which is a triple of the condition, the source and destination states.

An upper-level character has its appearance specified in the blue block of composing definition in Figure 3. There the lower-level characters are listed with their initial relative positions to form the upper-level character and also with their initial states.

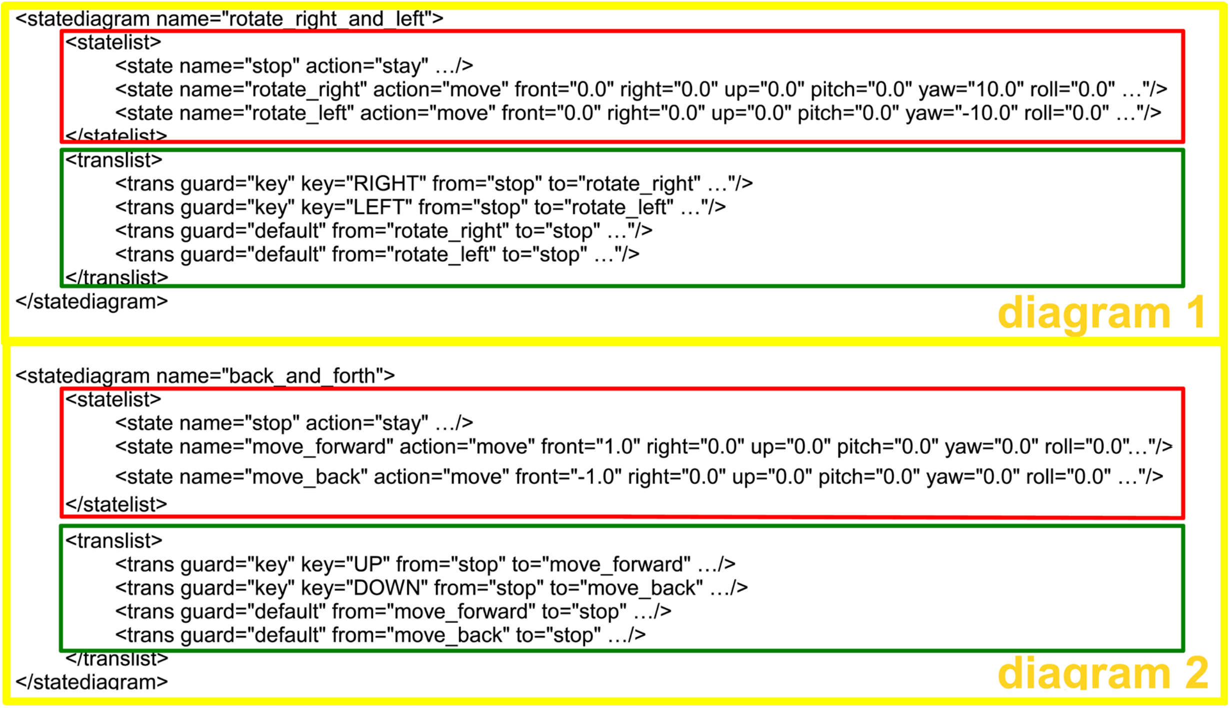

atom 3D object definition
character definition
composing definition
group definition
world definition

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<islay3d ver="1.0">
<objectlist>
    <data name="sideways_hammer" path="sideways_hammer.dae"/>
    <data name="ufo" path="ufo.dae"/>
    …
</objectlist>
<characterlist>
    …
    <character name="babo" parts="babo" rotation="1.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,1.0">
        <statediagram name="rotate_right_and_left">
            <statelist>
                <state name="stop" action="stay" pos_x="339" pos_y="165"/>
                …
            </statelist>
            <translist>
                <trans guard="key" key="RIGHT" from="stop" to="rotate_right" ctrl1="418,116" ctrl2="521,131"/>
                …
            </translist>
        </statediagram>
        <statediagram name="back_and_forth">
            <statelist>
                <state name="stop" action="stay" pos_x="339" pos_y="165"/>
                …
            </statelist>
            <translist>
                <trans guard="key" key="RIGHT" from="stop" to="rotate_right" ctrl1="418,116" ctrl2="521,131"/>
                …
            </translist>
        </statediagram>
        …
    </character>
    …
</characterlist>
<complexlist>
    …
    <complex name="babo">
        <member character="right_and_left_hands" state="wait,stay,stay,stay,stay" x="0.0" y="0.0" z="0.0"/>
        …
    </complex>
    …
</complexlist>
<grouplist>
    <group name="main">
        <fork character="babo" state="appearance_position,stop,stop,wait,count_0" x="0.0" y="0.0" z="0.0"/>
        …
    </group>
    …
</grouplist>
<option>
    <interval value="100"/>
    <room type="room" width="50.0" height="15.0" depth="100.0"/>
    <color roof="255,255,255" floor="255,255,255" wall="128,255,128" background="255,255,255"/>
</option>
</islay3d>
```

Figure 3. Example of Animation Definition by Islay 3D.

A group comprises pairs of a character name and its initial state. By the "fork" action, we can create their instances at the same time in the specified initial states. There is at least one group named *main group*, which is the set of characters to start up the animation as in the part of group definition in Figure 3.

The space where characters play inside is defined with its dimensions, type, background color, and the transition time interval as the part of the world definition in Figure 3. The type can be chosen out of a space having no boundaries, a ground with a



```
<statediagram name="rotate_right_and_left">
        <statelist>
                <state name="stop" action="stay" .../>
                <state name="rotate_right" action="move" front="0.0" right="0.0" up="0.0" pitch="0.0" yaw="10.0" roll="0.0" ..."/>
                <state name="rotate_left" action="move" front="0.0" right="0.0" up="0.0" pitch="0.0" yaw="-10.0" roll="0.0" ..."/>
        </statelist>
        <translist>
                <trans guard="key" key="RIGHT" from="stop" to="rotate_right" ..."/>
                <trans guard="key" key="LEFT" from="stop" to="rotate_left" ..."/>
                <trans guard="default" from="rotate_right" to="stop" ..."/>
                <trans guard="default" from="rotate_left" to="stop" ..."/>
        </translist>
</statediagram>
```

diagram 1

```
<statediagram name="back_and_forth">
        <statelist>
                <state name="stop" action="stay" .../>
                <state name="move_forward" action="move" front="1.0" right="0.0" up="0.0" pitch="0.0" yaw="0.0" roll="0.0"..."/>
                <state name="move_back" action="move" front="-1.0" right="0.0" up="0.0" pitch="0.0" yaw="0.0" roll="0.0" ..."/>
        </statelist>
        <translist>
                <trans guard="key" key="UP" from="stop" to="move_forward" .../>
                <trans guard="key" key="DOWN" from="stop" to="move_back" .../>
                <trans guard="default" from="move_forward" to="stop" .../>
                <trans guard="default" from="move_back" to="stop" .../>
        </translist>
</statediagram>
```

diagram 2

Figure 4. Internal expression of state-transition diagrams.

boundary on the floor, and a room that has six surrounding boundaries.

3. Execution Model in ActionScript 3

The animation definition should be converted by a translator into an ActionScript3 (AS3) code for its implementation as a Flash animation. Compiling the animation definition in AS3, we will get Flash 3D animation that works the same as the original Islay 3D animation. In that way, we can play Islay 3D animation on the Flash platform.

3.1. Implementation of State-Transition Diagrams

In a translation into AS3 code, actions and state transitions take place in the following steps.

1. A class for mediating the whole animation watches the timer until the next timing for actions and state transitions.
2. The mediator class calls the function for taking action and shifting states implemented in each of the top-level characters. The function call is accompanied by events conditioning the state transition.
3. Each character takes actions and shifts states on events.
4. If the character retains its children (lower-level characters), it calls the function of its children for taking action and shifting states with the events.
5. Repeat 3 and 4 until all the bottom-level characters take actions and shift states.

In that way, all the characters keep taking their actions shifting their states by a regular interval.

3.2. Implementing Hierarchal Structure of Characters

In order to implement the hierarchical structure as illustrated in Figure 2, we introduce a class named Performer. One instance of this class encapsulates that of any character. As indicated in the top line of Figure 5, this class inherits the DisplayObject3D class of the Papervision3D library that is the fundamental class for any kinds of 3D objects. Among the properties of the DisplayObject3D class, the Performer class especially utilizes the following: Multiple instances of the DisplayObject3D class can be retained in an instance of the DisplayObject3D class as its children. The children instances behave relative to the local coordinate system of their parent instance to constitute the appearance of the parent. The DisplayObject3D retains all the parameters for three-dimensional manipulation of objects such as vertex coordinates, local and global coordinate systems, and the rotation angles. It has also the function for transforming pose and position of 3D objects by manipulating those parameters.

In the light blue ovals, the Performer class retains a reference to an instance of the class for an atom 3D object in the case of bottom-level character or to a list of instances of the Performer class in the case the character is made up of lower-level characters. In that way, the hierarchical structure of the characters can naturally be implemented. It is even capable of communicating with the children, which provides a natural means to control the children.

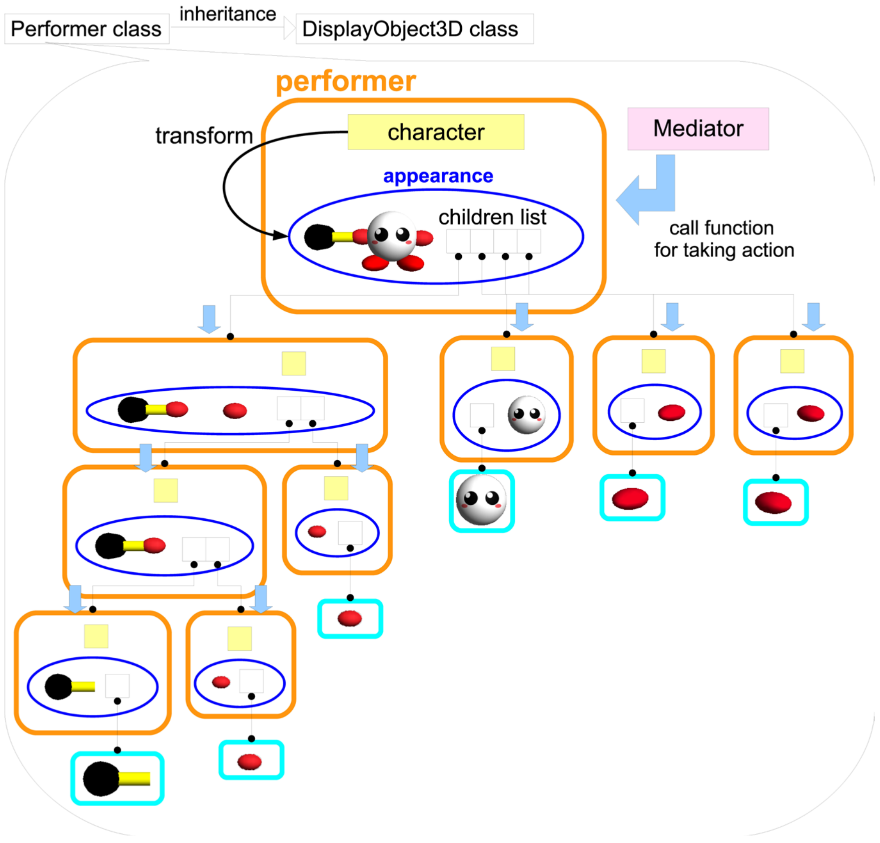


Figure 5. Hierarchical structure of characters by means of the Performer class.

The character represented by the yellow box in Figure 5 is an instance of the Character class of which detail will be described later. The Character class describes how a character should behave in accordance with the state-transition diagrams. Encapsulated in the Performer class, this class implements functions for state transitions and geometrically manipulating the Performer class instance as a 3D object.

3.3. Implementing Actions and State Transitions on Events

In order to cause actions and state transitions according to events, we introduce a class named Mediator that is marked in pink in Figure 5. It uses the ObjetCollision and WallCollision classes as schematized in Figure 6 for detecting collisions.

The mediator class controls the progress of whole animation by watching the timer and calling the function for taking actions implemented in the top level character by a constant time interval. When calling the function, events such as key press, messages, and collisions are sent to the function, which recursively calls that of the lower-level characters as indicated by the light blue arrows in Figure 5.

3.4. Conversion of Data Structure of Animation

We describe how to convert the Islay 3D animation definition in Figure 3 into the form of object class definitions in AS3.

In order to implement the atom 3D object definition such as the one in the light blue block of Figure 3, we introduce the Object class. This Object class inherits from the Objects class of the Papervision3D library its functions such as creating an instance of an atom 3D object and creating an instance of bounding box object for collision detection.

In Figure 7, the BoundingVolume class retains data fields of the oriented bounding box for collision detection. The ExtendedDAE class creates an atom 3D object as an instance of itself. An atom 3D object being a DAE file is parsed by the ExtendedDAE class extending the DAE class of Papervision3D to create an instance of the atom 3D object and embed it to be available in the animation.

In order to implement the character definition including actions and state transitions, we introduce the Character class that inherits the Characters class as illustrated in Figure 8.

The Characters class includes common functions for applying actions to the per-

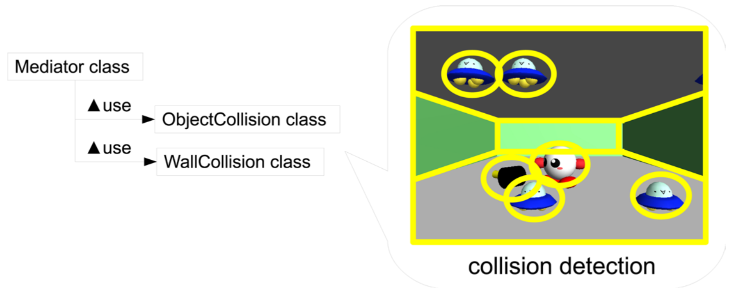


Figure 6. Mediator class.

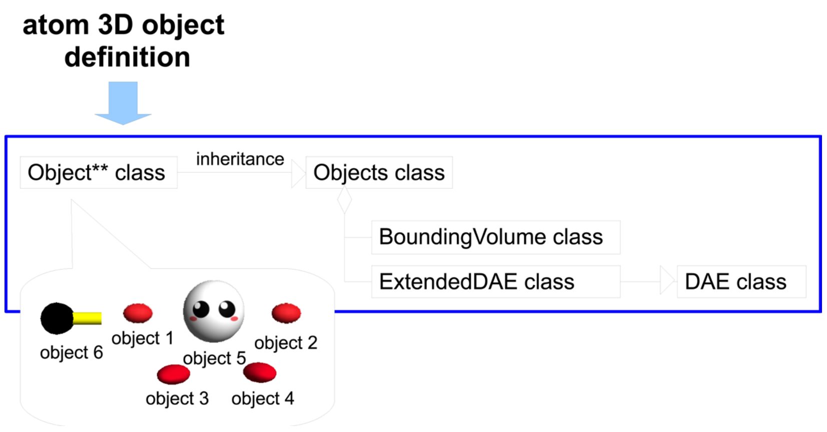


Figure 7. Object Class.

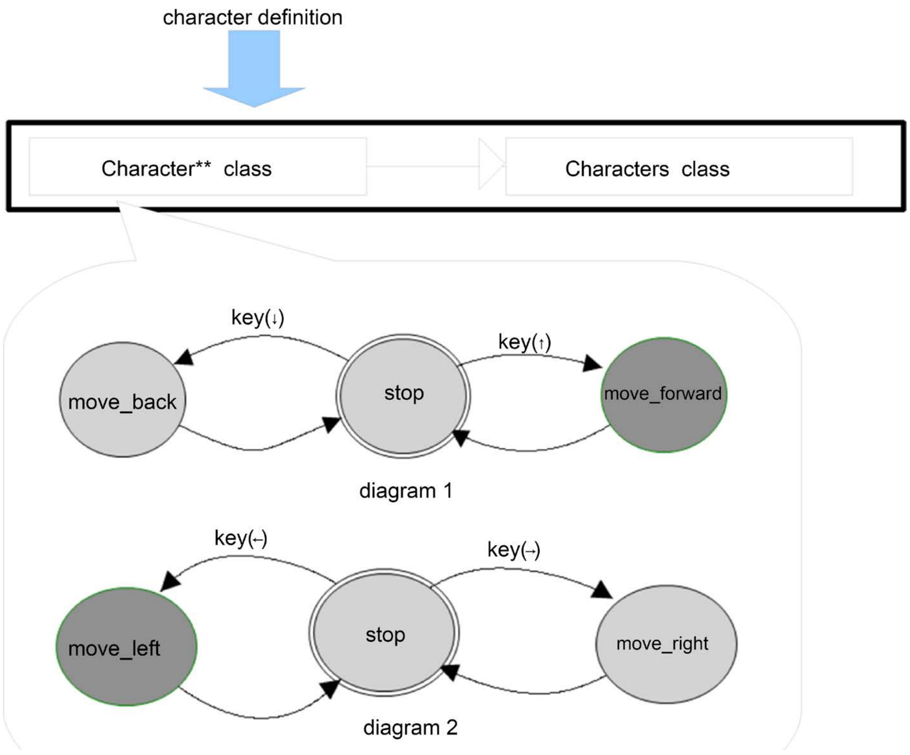
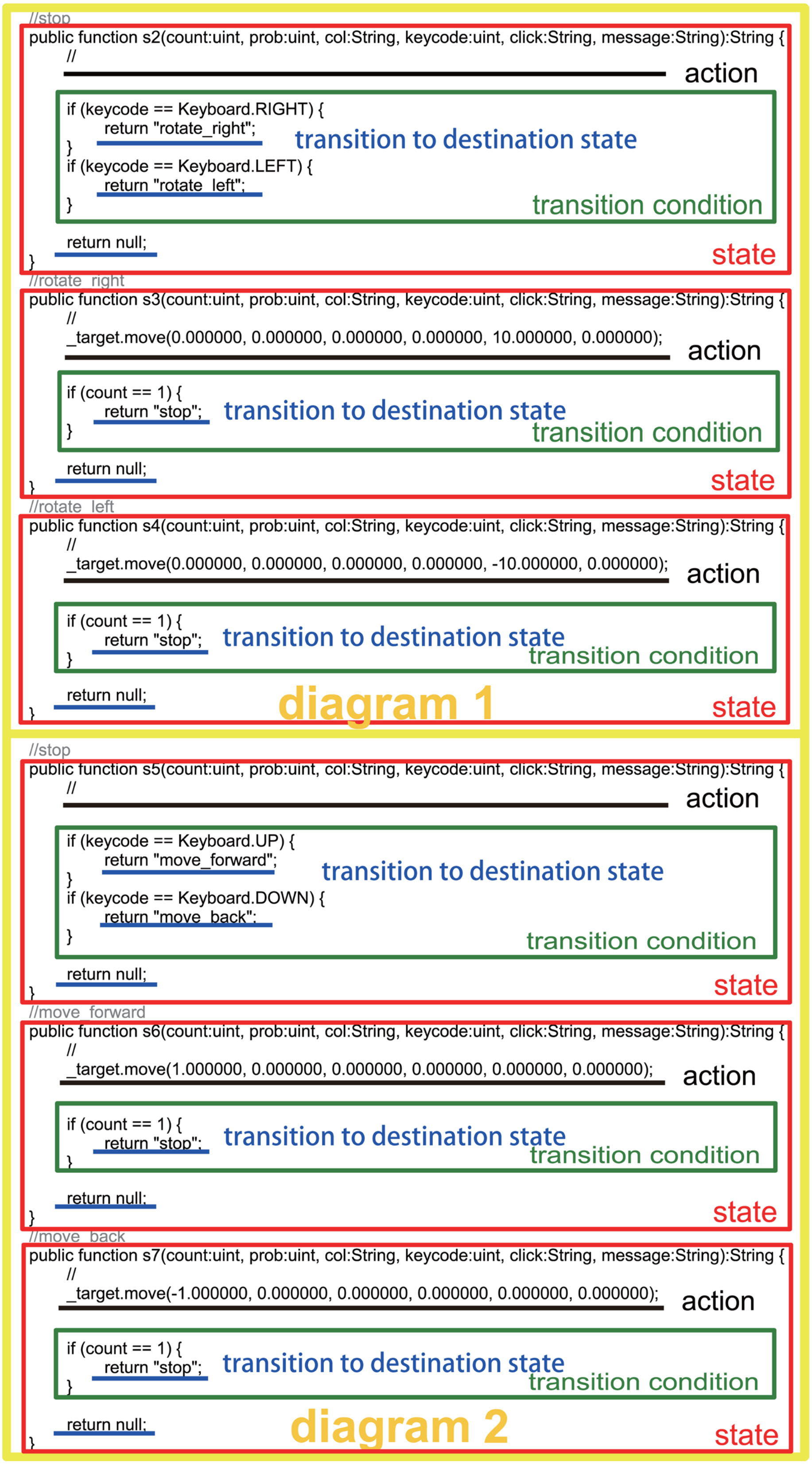


Figure 8. Character class.

former class and a function to remember the current state. Inheriting the Characters class, the Character class implements the state transition diagrams in the form illustrated in Figure 9 for the example character definition in Figure 4. As in the red block of Figure 8, each state is represented by a function that applies actions in the state to the 3D object which is appearance of the Performer class, and returns the name of the next state at the parts underlined in blue. In the green block, the destination state is determined as the return value according to the transition conditions. If no conditions match, the state remains the same (for the null return value).

The Character class retains the name of its appearance by which the Performer class



```
//stop
public function s2(count:uint, prob:uint, col:String, keycode:uint, click:String, message:String):String {
    //

    if (keycode == Keyboard.RIGHT) {
        return "rotate_right";
    }
    if (keycode == Keyboard.LEFT) {
        return "rotate_left";
    }

    return null;
}
//rotate_right
public function s3(count:uint, prob:uint, col:String, keycode:uint, click:String, message:String):String {
    //
    _target.move(0.000000, 0.000000, 0.000000, 0.000000, 10.000000, 0.000000);

    if (count == 1) {
        return "stop";
    }

    return null;
}
//rotate_left
public function s4(count:uint, prob:uint, col:String, keycode:uint, click:String, message:String):String {
    //
    _target.move(0.000000, 0.000000, 0.000000, 0.000000, -10.000000, 0.000000);

    if (count == 1) {
        return "stop";
    }

    return null;
}
```

```
//stop
public function s5(count:uint, prob:uint, col:String, keycode:uint, click:String, message:String):String {
    //

    if (keycode == Keyboard.UP) {
        return "move_forward";
    }
    if (keycode == Keyboard.DOWN) {
        return "move_back";
    }

    return null;
}
//move_forward
public function s6(count:uint, prob:uint, col:String, keycode:uint, click:String, message:String):String {
    //
    _target.move(1.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000);

    if (count == 1) {
        return "stop";
    }

    return null;
}
//move_back
public function s7(count:uint, prob:uint, col:String, keycode:uint, click:String, message:String):String {
    //
    _target.move(-1.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000);

    if (count == 1) {
        return "stop";
    }

    return null;
}
```

Figure 9. Function representing state and transitions in the Character class.

gets an instance of the PartsFactory class (described later). The Performer class, called by the Mediator class by a regular time interval, calls the function representing the current state of the encapsulated Character class to apply the actions to its appearance. The return value of the function updates the current state if not null.

The PartsFactory class creates instances of the Object class in accordance with the atom 3D object definition and also composes the appearances of upper-level characters in accordance with the composing definition such as in Figure 3.

When the instance of the Performer class is created, 3D object name for specifying its appearance is passed to this class. Example atom 3D object names are listed in the light blue block in Figure 3. In the blue block of composing definition in Figure 3, the lower-level characters are registered with their relative positions to compose the upper-level character and also with their initial states.

The PartsFactory class creates (3D) Object instances, sets their initial positions, and returns these to the Performer class instance as illustrated in Figure 10. In the case that the Performer class instance represents a bottom-level character, this class returns an Object class instance. In the case that the Performer class instance represents an upper-level character, this class returns a set of Performer class instances.

The groups class retains pairs of a character and its initial state and creates instances of the Performer class retaining those characters when the they are forked. The group of characters in Figure 11 is the initial performers in this example animation.

The Main class sets a space where characters play inside in accordance with the world definition such as in Figure 3. As illustrated in Figure 12, the Main class inherits from the IslayBasicView class its auxiliary functions for displaying walls and catching user events such as key input and mouse click. The IslayViewport3D class displays the walls. The IslayInteractiveSceneManager class makes it possible to catch the mouse event.

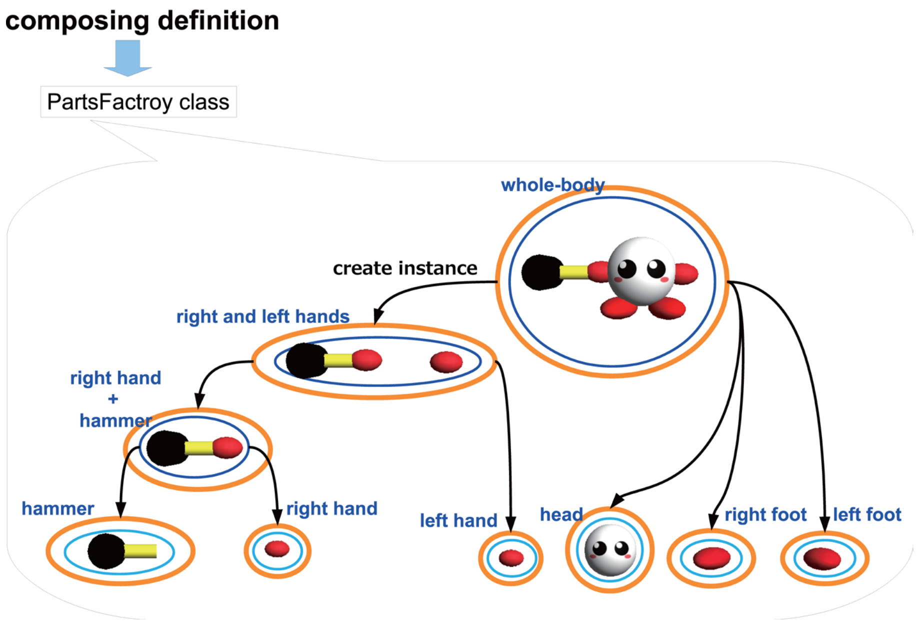


Figure 10. PartsFactoy class.

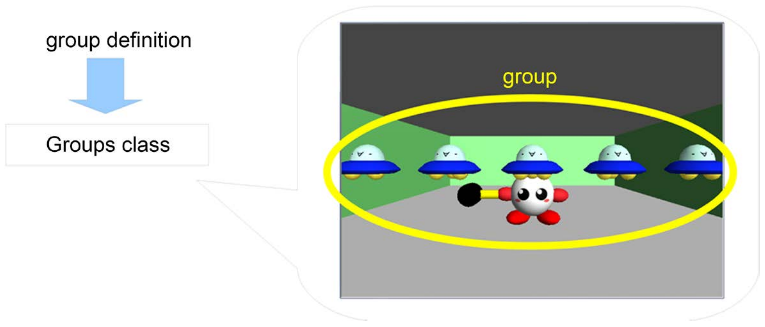


Figure 11. Groups class.

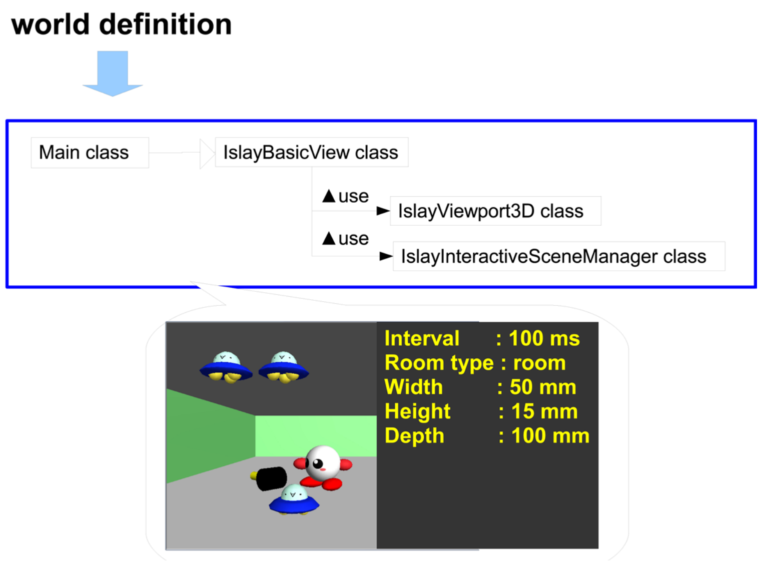


Figure 12. Main class.

4. Performance Evaluation

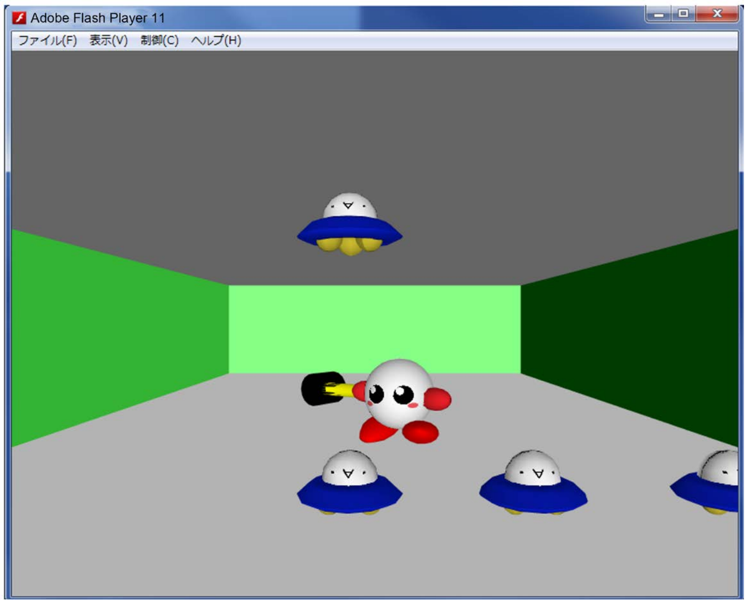
The developed translator converts an Islay 3D animation into ActionScript3, which is then compiled by the Flash compiler to yield SWF (small web format) file playable on the Flash platform.

Figure 13 shows screenshots of an example 3D animation running as a Flash content. The ball-like character walks around in the space according to the arrow key input and throws a hummer at the flying saucers when the space bar is pressed.

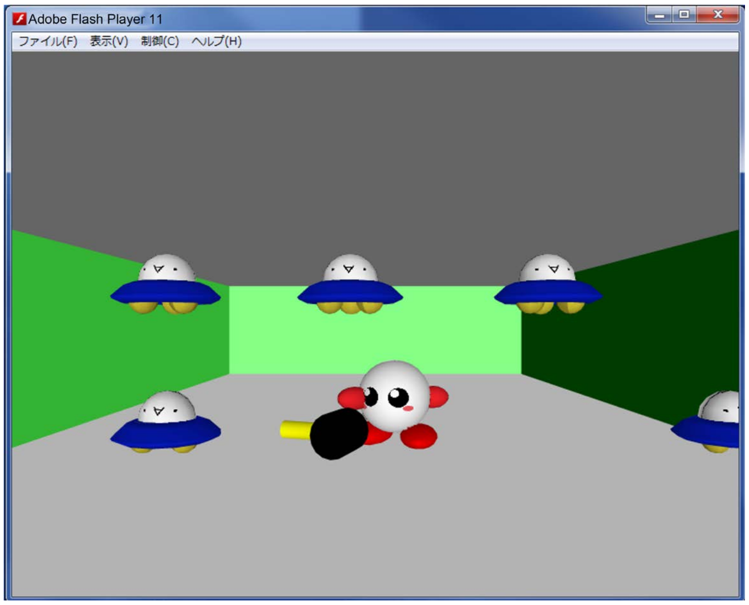
Table 1 shows the memory and CPU usages of the animation when played as a Flash content or as the original animation definition interpreted by the old built-in interpreter of Islay 3D written in JavaScript on the same PC of the specifications listed in Table 2. Table 1 indicates that the Flash content produced by the developed translator consumes only a quarter of the memory and a half of the CPU power consumed by the old interpreter on JavaScript.

5. Conclusion

We have developed a translator of Islay 3D animations into ActionScript3 for the



(a)



(b)

Figure 13. Screenshots of Flash animation.

Table 1. Performance comparison in terms of memory and CPU usages.

Browser	Firefox 48		Internet Explorer 11	
Content	Flash	JavaScript	Flash	JavaScript
Memory usage	105 - 120 MB	600 - 630 MB	130 - 145 MB	680 - 720 MB
CPU usage	16% - 18%	39% - 42%	17% - 18%	48% - 53%

Table 2. Specifications of the test platform.

Parts	Specifications
OS	Windows 7 Professional 64 bit
CPU	Core i7 2600, 3.4 GHz, 4 cores, 8 threads, 8 MB cache
Main memory	8 GB
Graphic card	AMD Radeon HD 6350
Display	Width 1920 × Height 1080 pixels

purpose of playing the animations on the Flash platform. A comparison of the memory and CPU usages for the same animation played as a Flash content produced by the developed translator or as the original animation definition interpreted by the old interpreter written in JavaScript has shown that the memory and CPU usages are much saved, quartered and halved, respectively. That means that the developed translator helps the smaller computers run the animation created by Islay 3D. On the other hand, for the creation of animations by the Islay 3D editor, we still have to use the larger and faster computers. That is a remaining problem to be solved in the future.

References

[1] Adobe Flash Player. http://www.adobe.com/products/flashplayer.html

[2] Papervision3D. https://en.wikipedia.org/wiki/Papervision3D

[3] Adobe Flash. http://www.adobe.com/products/flash.html

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

[5] Fowler, A., Fristace, T. and MacLauren, M. (2012) Kodu Game Lab: A Programming Environment. *The Computer Games Journal*, **1**, 17-28.

[6] 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.

[7] Kwong, D., Niibori, M., Okamoto, S., Kamada, M. and Yonekura, T. (2014) Islay 3D—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

[8] Nakagawa, M., Okamoto, S., Kamada, M. and Yonekura, T. (2006) Flash Movie Authoring Environment Based on State Diagram. *Proceedings of* 5*th ACM SIGCOMM Workshop on Network and System Support for Games* (*NetGames* 2006), Singapore, Article No. 45. http://dx.doi.org/10.1145/1230040.1230055

[9] Niibori, M., Arisawa, Y., Okamoto, S., Kamada, M. and Yonekura, T. (2012) An Authoring Tool for Flash Games in ActionScript3.0. *Proceedings of the* 15*th IEEE International Conference on Network-Based Information Systems* (*NBiS* 2012), Melbourne, 26-28 September 2012, 889-892.

[10] COLLADA—Digital Asset and FX Exchange Schema. https://www.khronos.org/collada/