In this issue:

Alice and the Introductory Programming Course: An Invitation to Dialogue

Daniel V. Goulet
University of Wisconsin Stevens Point
Stevens Point, WI 54481 USA

Donald Slater
Carnegie Mellon University
Pittsburgh, PA 15213 USA

Abstract: Alice, a 3D visual graphics environment, represents a breakthrough in teaching object-oriented computing by making objects visible. The object-oriented paradigm, though intuitive in its general form, is, for most students, a new way of thinking. The question is: How can we, as educators, make the most of Alice's unique teaching environment, and what can we do to enhance student learning? The objective of this paper is two fold: (1) to create a dialogue about innovative and effective ways to use Alice as a teaching and learning tool, and (2) to exhibit an approach for relating the activities (features) of Alice to the teaching and learning requirements of the object-oriented paradigm.

Keywords: Alice, Introduction to programming, Object-oriented paradigm, Pedagogy, Introductory programming course, CS1

This issue is on the Internet at **http://isedj.org/7/50/**

EDSIG activities include the publication of ISEDJ and JISAR, the organization and execution of the annual ISECON and CONISAR conferences held each fall, the publication of the Journal of Information Systems Education (JISE), and the designation and honoring of an IS Educator of the Year. • The Foundation for Information Technology Education has been the key sponsor of ISECON over the years. • The Association for Information Technology Professionals (AITP) provides the corporate umbrella under which EDSIG operates.

Alice and the
Introductory Programming Course:
An Invitation to Dialogue

Daniel V. Goulet
dgoulet@uwsp.edu
Computing & New Media Technologies
University of Wisconsin-Stevens Point
Stevens Point, WI 54481 USA

Donald Slater
dslater@cmu.edu
School of Computer Science
Carnegie Mellon University
Pittsburg, PA 15213 USA

Abstract

Alice, a 3D visual graphics environment, represents a breakthrough in teaching object-oriented computing by making objects visible. The object-oriented paradigm, though intuitive in its general form, is, for most students, a new way of thinking. The question is: How can we, as educators, make the most of Alice's unique teaching environment, and what can we do to enhance student learning? The objective of this paper is two fold: (1) to create a dialogue about innovative and effective ways to use Alice as a teaching and learning tool, and (2) to exhibit an approach for relating the activities (features) of Alice to the teaching and learning requirements of the object-oriented paradigm.

Keywords: Alice, Introduction to programming, Object-oriented paradigm, Pedagogy, Introductory programming course, CS1

1. INTRODUCTION

Students who now take an introductory programming class are often exposed to the same tools, problems, and pedagogical approaches that have been around for the last 20 years. There has been significant change in the languages taught, and the programming paradigms used in these courses. But today's students have a different experience with computers than students of 20 years ago.

In the introductory programming text (Miler, 1987), the first example of a Pascal program that students saw was "Hello World".

```
PROGRAM Example (INPUT, OUTPUT);
BEGIN
      WRITELN(OUTPUT, 'Hello world)
```

```
END.
```

In the introductory programming text (Horstmann, 2008), the first Java program that students today see is "Hello World".

```
public class HelloPrinter
{
      public static void
main(String[] args)
      {
            Sys-
tem.out.println("Hello, World!");
      }
}
```

The output to the console, that was reasonably exciting to students of 20 years ago, now looks like a text message that today's students currently receive on their cell

phone. Students have different expectations of what they can and should be able to do with a computer.

The object-oriented paradigm is the fundamental construct for modern programming languages, and both teaching and learning this paradigm are difficult tasks at best. As ever, illustrations, using graphics, diagrams, charts, cartoon, etc., are more effective than excessive text. One of the oldest challenges of object-oriented teaching is that students are exposed to concepts, concepts that are taught through other concepts. What students need are hands-on experiences allowing the mind-shift towards object-oriented thinking to occur. (Alice, 2007) has opened a door to make both teaching better and learning more effective.

In a paper entitled *Evaluating the Effectiveness of a New Instructional Approach*, (Moskal, 2004) reports that at risk computer science majors, i.e., students with limited previous programming experience, in preparatory courses for CS 1 that used curricular materials with the Alice software exhibited improved performance (average grade from C to B in CS 1), and more students went on to CS 2. (Without the Alice experience, only 47% of these students went on to CS 2. With the Alice experience, 88% of the students went on to CS 2).

Alice is both an innovative software tool and a pedagogical approach designed to allow traditional programming concepts to be more easily taught and more readily understood by today's students.

The question is: How can we, as educators, make the most of Alice's unique teaching environment and what can we do to enhance / increase student learning?

In this paper the authors pursue two objectives:

1.   to create a dialogue about innovative and effective ways to use Alice as a teaching and learning tool, and

2.   to exhibit one strategy for relating the activities of Alice to the teaching / learning of requirements of the object-oriented paradigm, with Java as the example target language.

2. THE OBJECT-ORIENTED PARADIGM

The object-oriented paradigm, as seen by the authors, has the following characteristics:

- The world is viewed as a collection of objects.

- An object is a realization [instance] of a class.

- A class is a definition or template for creating objects.

- Objects know things.

- Objects know how to do things.

- Objects are assigned responsibilities, and when asked, carry out that responsibility.

- Objects interact by passing messages.

- An object-oriented program is a collection of interacting objects.

- Simple object-oriented programs implement a 3-tier architecture
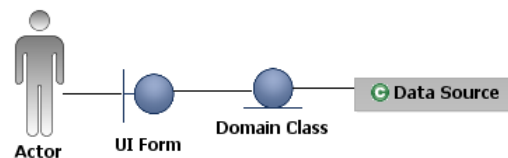
Figure 1: Author's View of 3-Tier Architecture

We use our view of the object-oriented paradigm and our understanding of Alice to create and exhibit the following mapping: (i) identify the object-oriented paradigm characteristic, (ii) show the characteristic's realization in Alice, (iii) for concreteness and to focus the discussion, map the Alice realization into Java. Before we can do this, however, we

- need to have a basic understanding to the question: "What is Alice?", and

- have to understand the strengths and limitations of Alice when used as the authors are proposing.

3. WHAT IS ALICE?

Alice populates a 3D microworld with tangible and visible objects, supporting the creation of movies and interactive games. These movies and games, or programs, are created by dragging and dropping instructions, expressions, and control structures using an integrated GUI. This not only eliminates common syntax errors, but allows for the rapid building of an experiment in an interactive manner.

The visual feedback provided by the interaction of the objects in the microworld allows changes in the properties, or state, to be easily seen. This same feedback also allows errors in the design or implementation of the story or game to be readily identified.

The Alice system provides a powerful, modern programming system that supports methods, functions, variables, parameters, recursion, arrays, and events. Alice seeks to provide a means to help overcome four primary obstacles encountered by beginners in early programming courses:

1. The perils of program creation, particularly in navigating the complexities of programming languages syntax.

2. The difficulty in mastering the complex tools (programming environments) used in creating programs in modern programming languages.

3. The difficulty in seeing the ongoing process and intermediate results of computation as the program runs.

4. The lack of a rich, motivating, and engaging problem set to stimulate the learner's curiosity and desire to master the skills necessary for program creation.

The figures in the Appendix provide a brief overview of the Alice interface and its use. Figure 2 shows the interface of the Alice software, which includes the Object Tree, a listing of all the objects in the current world, the Details Pane which lists the properties, methods, and functions of the selected object in the Object Tree, the World View which displays the world being created, the Editor for creating the program, and the Events Editor, for creating interactive worlds.

Figures 3 and 4 illustrate how to create new worlds in Alice, and how to add objects to the world from the Object Gallery. Figure 5 shows how to write program code in the editor by dragging available behaviors from the Details Pane.

4. STRENGTHS AND CHALLENGES OF TEACHING THE OBJECT-ORIENTED PARADIGM USING ALICE

The strengths of Alice, as seen by the authors, in helping students to develop a solid understanding of the Object-Oriented Paradigm are as follows:

Conceptual Strengths

After working with Alice, students should understand:

- The distinction between a class and those objects that are a realization of the class.

- That there are often many instances of a particular class in a world, all with their own properties and behaviors.

- The distinction between properties and behaviors.

- That multiple objects, operating independently and yet cooperatively within the world, make unique and important contributions to the completion of the world's mission (story, interactive game, program.)

- That messages are used to prompt the behaviors of or elicit information from particular objects.

- That every message must be addressed to a particular object.

- And have seen that the same message, sent to different objects, can promote different, yet related behavior from each object.

- That a programmer may create new behaviors for an object, expanding the set of behaviors for that object.

- The role parameters play in qualifying the message to the object, allowing the object to be more specific in the behavior that is generated.

- That the use of parameters allows the creation of new methods that are more

general purpose, allowing an object to exhibit the same behavior in different contexts.

- The difference between messages that elicit behaviors from an object (methods in Alice terms), and messages that elicit a response to a query (functions in Alice terms.)

- Conditional control structures which may be sequenced and nested in ways to handle complex alternatives.

- Relational and Boolean operations, generate appropriate Boolean expressions, and know that iterative control structures come in different forms.

- Different iterative control structures, how to successfully use these control structures in problem solving, and choose the appropriate iterative structure in their program design and implementation.

- Type, and its significance in modern programming languages.

- Recursion intuitively, and have had the chance to implement one recursive solution.

- The idea of the variable and the role variables can play in program design and implementation.

Intangible Strengths

Experience seems to indicate that working with Alice helps develop persistence in creating solutions to problems and the debugging of solutions that have gone awry. This seems to come from the visual nature of the Alice environment. Students can see that they are making progress toward their goal, the solution, story, or game. They can see the impact of the addition, subtraction, or modification of any one statement in their code, and whether such a change is getting them closer to their desired outcome or further away. They see that they are making progress, are visually rewarded as they make progress, and develop a sense that further effort is justified and will pay off.

They get intrinsic motivation in telling their story, or building their game, having something to share with others in the class also supports this notion of persistence because they are working on something that is engaging and motivating.

Persistence in debugging is also developed, for many of the same reasons that persistence in creating the code is developed. The students can see their mistakes; they can interpret what has gone wrong by analyzing the action as it unfolds on the screen. The fact that visual mistakes in the Alice environment are often amusing reduces the frustration that students may otherwise feel in having their code go bad.

Self confidence is developed as students realize that they are capable of telling a complex story or building a game that their friends want to play. They recognize that they have worked hard, but that it is good work, and they are often surprised to realize that they can work harder, and more effectively than perhaps they realized.

Interest in computing is often developed, as students realize that developing a set of programming tools will allow them to do cool things that are interesting. They realize that programming is not about developing the discipline to solve intrinsically uninteresting problems, but rather developing skills to do cool things.

Syntax and Typing Challenges

As the students move from the safety and support of the Alice development environment and start to work with a professional programming language, they confront the same problems all introductory programming students encounter when faced with the syntax and rigors of a formal programming language.

The students are often confronted with the complexities of a full blown, professional programming environment. But even those programming environments, developed to support introductory programming classes, possess some intricacies that are not easily and intuitively mastered.

The advantage of working with Alice is that confronting both of these obstacles comes after students have had a chance to develop the conceptual understanding of many important ideas in introductory programming, and have been able to develop the self-confidence and persistence to overcome these hurdles. Alice also has some tools to help with this transition, including the ability

to display Alice code in traditional Java form, including braces, semicolons, and parentheses.

But the hurdles are there nevertheless, and the instructor has to be prepared to support the students through this transition from Alice to the next language in the course. It is our experience that it takes students about a week to become comfortable typing Java / C++ code. In addition, the time necessary to learn how to use a particular IDE must also be factored in to this transition period.

Design Challenges

The Alice model uses storyboards, as seen in the professional development of movies, animated features, and interactive games as the design tool for building Alice projects. Work is still being done determining how this design paradigm may be carried forward through the rest of the introductory programming course.

Mediate the Transfer Challenge

Instructors often assume that concepts that students master during the Alice portion of the course will automatically be mapped to related topics in the second part of the course. Unless the instructors explicitly and repeatedly make the connection between the work in Alice with the new work being done with Java / C++, the students will generally not make the connection on their own.

Students like to see a course as unrelated topical units. This reduces the perceived workload in the entire course for the student. Once they have taken and passed the Alice exam, they no longer have to worry about Alice. There is some resistance to going back and learning topics that students think were already covered, even though those topics are now being explored in more depth, with more nuance and subtlety, which students would prefer not to have to pay attention to in any circumstance. The instructor should make sure that the students understand the pedagogical approach being employed. That there will be a turning back and new exploration of topics already introduced to develop a deeper and more comprehensive understanding of these topics.

When students are explicitly supported in mediating the transfer of content under-standing in Alice to the content being explored in Java / C++, when they understand the pedagogical approach being employed, they seem to be able to develop the hoped-for deeper understanding, with many "Ah-Hah!" moments. They also tend to appreciate the effort the instructor is now perceived to be putting forth in supporting their learning.

5. TOPICS LIGHTLY INTRODUCED OR NOT INTRODUCED IN ALICE

Alice is object based, more than object-oriented. There are some topics important in an introductory programming course that Alice does not or cannot cover in great depth. Thus the instructor must put more focus on these areas in the Java / C++ portion of the course. This focus will usually be in deeper explanation, practice, labs and assignments.

Alice only lightly covers mutable variables, and the expression editor of Alice makes the creation of complex numeric and Boolean expressions cumbersome. There is some exploration of Numeric types, but there is no distinction made between integer and floating point numbers.

The dirty little secret in Alice is that all references are global, scoping is lightly enforced. The instructor, in making sure that no examples are shown where the global nature of the references is exposed, can finesse this. But this takes careful thought and planning of examples.

There is a very tight coupling between a particular object and its reference. A reference may not be reassigned to a different object during runtime. There is no notion of the *this* operator, or the implicit *this* in Alice. Every message is sent to a specific reference. There is no indirect object reference in Alice, for example, the use of an indexed array variable to access the members of a particular object.

Some topics are just not covered in Alice. There is no creation of a class definition from scratch. There is no explicit idea of constructors. Dynamic object creation does not exist in Alice. (There is no mechanism to *new* an object during runtime.) There are no interfaces, no notion of class members using *static*, polymorphism.

These topics must be carefully attended to in the Java / C++ portion of the course.

6. THE AUTHORS CONTRIBUTION TO THE DIALOGUE: ALICE FOLLOWED BY JAVA – THE TRANSFER

In the tables found in APPENDIX A, we will identify each characteristic as defined in our earlier stated view of the Object-Oriented Paradigm. We will than show how that characteristic is or is not implemented in Alice. We will then exhibit our view of a mapping into Java, providing the reader with a concrete example of one pedagogical transfer.

7. SUMMARY AND THE INVITATION

The object-oriented paradigm of modern programming languages is both difficult to teach and difficult for students to learn. Our objective in this paper was to use Alice as a tool to teach the object-oriented paradigm, and to begin a dialogue on how to use Alice in this educational environment. Alice can be used in many different ways to teach many different topics, i.e., story telling, 3D animation, gaming, and programming. We exhibited how the content and concepts in Alice give a concrete realization to object-orientation. This realization, we believe, can form a very solid and conceptual foundation for the teaching and learning of the object-oriented paradigm, and thus to provide a better understanding of object-oriented programming and object-oriented analysis and design.

We invite others to bring forth their ideas and join the discussion of innovative and effective ways in how Alice can be used as a teaching / learning tool.

8. REFERENCES:

Alice. (2007) The latest version of Alice software and online galleries of 3D models can be downloaded from http://www.alice.org.

Dann, Wanda P., Cooper, Stephen, Pausch, Randy. (2006) Learning to Program with Alice. Pearson-Prentice Hall. Upper Saddle River, NJ.

From Method Summary for the Math class. http://java.sun.com/j2se/1.3/docs/api/ . Accessed: 30 July 2007.

Horstmann, Cay. (2008) Big Java. John Wiley & Sons, Inc. Hoboken, NJ. p 13.

Miler, Philip L. and Lee W. Miler. (1987) Programming by Design: A First Course in Structured Programming. Wadsworth Publishing Company. Belmont, CA. p 95.

Moskal, Barbara, Deborah Lurie Stephen Cooper. (2004) "Evaluating the Effectiveness of a New Instructional Approach." Proceedings of the 35 SIGCSE Technical Symposium on Computer Science Education. SIGCSE 2004. pp 75-79.

APPENDIX A

Characteristic: A class is a definition or template for creating objects

| *Alice Realization:*	*Mapping into Java:*
Pre-defined Classes:	Pre-defined Classes:

Alice Realization:

Pre-defined Classes:

Alice groups pre-defined classes into 'Galleries'. The Galley is a collection of all available 3D models that can be used in an Alice microworld.

User Defined Classes:

Alice allows for the extension of pre-defined classes. 'save object' would create a new 'joePenguin' class.
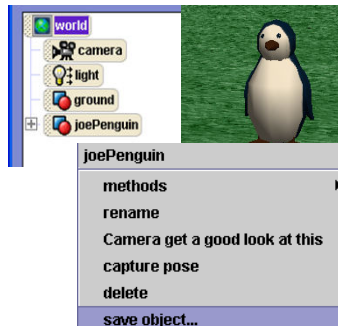
Alice does not allow for completely new, user-defined classes.

Mapping into Java:

Pre-defined Classes:

Java groups pre-defined classes into 'packages'. Packages are collections of similar classes that perform the same function as a single gallery set in Alice.

```
java.util package

java.awt package

java.math package

java.io package
```

User Defined Classes:

Java allows for the extension of pre-defined classes. Using the Alice example and assume Penguin is a pre-defined class, then:

```
public class joePenguin extends
Penguin
```

would extend the Penguin class.

Java permits the construction of new, user-defined classes. Again, using the Alice example and assume Penguin is not a pre-defined class, then:


```
public class Penguin{  }
```

Characteristic: An Object is a realization [instance] of a class

Alice Realization:

Objects in Alice are added by selecting one of the models in the Gallery, and the user is asked if they wish an instance of that class to be placed in the world. Multiple realizations may be added to the world in this way, each with its own attributes and behaviors.

Mapping into Java:

Objects in Java are added by declaring the object and then sending a message to the class to execute its constructor.

> Declaring and sending a message to the Penguin class:

```
Penguin joePenguin;

joePenguin = new Penguin ();
```
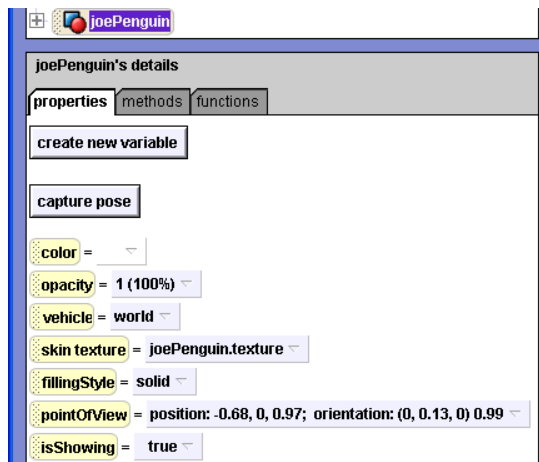
The Penguin Class:

```
public class Penguin()

    public Penguin ()

    {   }
```

Characteristic: Objects know things

Alice Realization:

Every object in Alice has a set of 'properties' that identifies 'what the object knows'. Selecting the Properties tab in the Details section for 'joePenguin', we have:


Mapping into Java:

What an object in Java 'knows' is what is declared in its attribute list. Using some of what joePenguin 'knows' in Alice, we can write:

```
public class Penguin()

/*  Knows things */

private String color;

private float opacity;

private String vehicle = world;

private String skinTexture;

private String fillingStyle;

private Boolean isShowing;
```

Characteristic: Objects know how to do things

Alice Realization:

Every object in Alice has a pre-defined set, or primitives, of 'methods' and 'functions' identifying what it 'knows how to do' and are listed in the Details windows. Methods are behaviors that the object can perform, and functions answer questions about the object and its relationship to other objects in the microworld. For 'joePenguin', we have the following partial list:
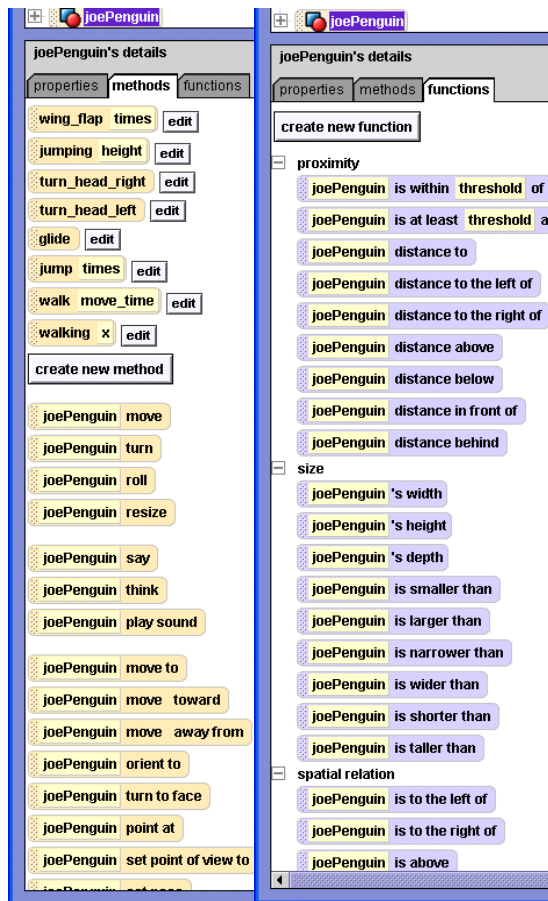
Objects in Alice can also have user-defined 'things it knows how to do'. Using the 'create new method' and 'create new function' in the above screen capture allows the programmer to define new methods and functions that become part of that instance's knowledge base. This new knowledge is not automatically transferred to other realizations of that class in the microworld or in the Gallery.

Mapping into Java:

Pre-defined objects instantiated from pre-defined packages have pre-defined 'methods', identifying what it 'knows how to do'. For the Math class taken from (Method Summary), we have the following partial list:

| **Method Summary**	
static double	**abs**(double a) Returns the absolute value of a `double` value.
static float	**abs**(float a) Returns the absolute value of a `float` value.
static int	**abs**(int a) Returns the absolute value of an `int` value.
static long	**abs**(long a) Returns the absolute value of a `long` value.
static double	**acos**(double a) Returns the arc cosine of an angle, in the range of 0.0 through *pi*.

Objects in Java, in general, have two groups of things it 'knows how to do'. One group is usually known as the standard methods – the 'getters' and the 'setters'. And one group is usually known as the custom methods. For the Penguin defined above, a custom method might be:

```
public void dance(int,seconds)
   {  }
```

Characteristic: Objects are assigned responsibilities, and when asked, carry out that responsibility

Alice Realization:

Object can manipulate their properties in the set up of the microworld, or may be accessed and / or manipulated during the program executions. 'joePenguin', below, is changing his 'isShowing' during run time.
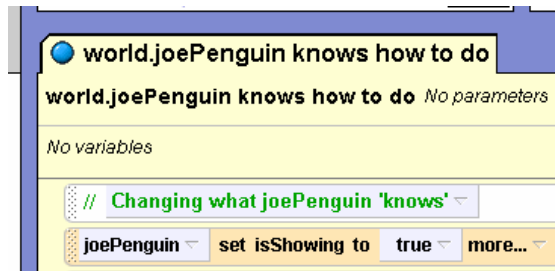
When assigned a different responsibility, the Alice object can carry out that responsibility. The object may be the primary actor in the microworld (the subject) receiving a message as a method, mediated with the dot (.) operator, with accompanying parameter values to clarify the behavior specified in the message (how far to move, for example). The object may receive a request for information it has, by receiving message as a function, also mediated with the dot operator, and appropriate parameter values. For example, we can give joePenguin the responsibility to 'glide'. And when asked to 'glide', joePenguin will carry out that responsibility by 'gliding' according to the 'glide' method.
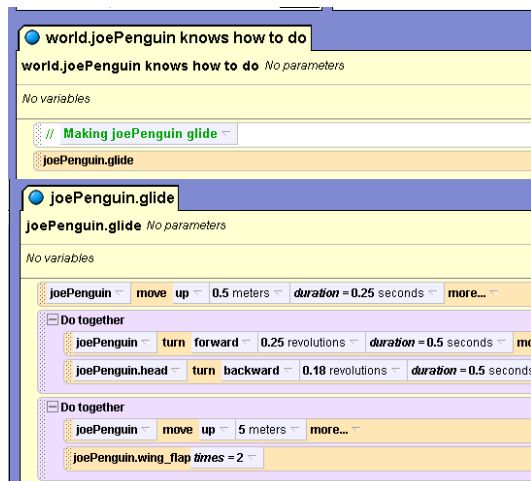
Mapping into Java:

Objects in Java can manipulate their attributes at compile time using default values, or during run time using the 'setter' methods. Carrying the Alice example of joePenguin into Java, we could have:

public class Penguin

{

/* Knows things */

Boolean isShowing = false;

/* set methods */

public void setIsShowing(Boolean, s)

{ isShowing = s;}

}

When assigned a different responsibility, the Java object can carry out that responsibility by executing a custom method for that responsibility. Again, following the Alice example of joePenguin, and assuming some liberties in the Java environment, we can make joePenguin glide. First the request to the joePenguin object to 'glide':
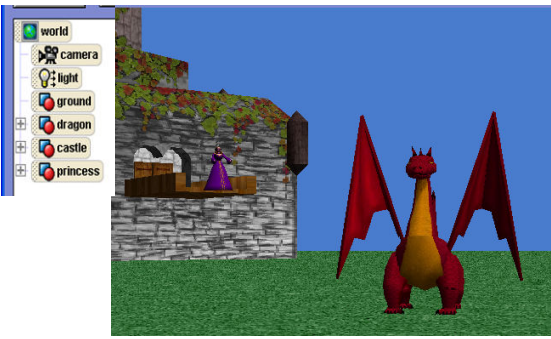
```
joePenguin.glide;
```

Now the way joePenguin 'glides' as an object of the Penguin class:

```
public void glide ()

    {

    penguin.move(char u, float d,
    float,t);

    penguin.turn(char f, float d,
    float t);

    penguin.head.turn(char     b,
    float d, float t);

    penguin.move(char u, float,d);

    penguin.wing.flap(int n);

    }
```

Characteristic: Objects interact by passing messages

Alice Realization:	*Mapping into Java:*
In Alice, an object, like joePenguin is composed of object subparts, i.e. a head, body, right and left legs, right and left wings. 'joePenguin', above, passes the message to the wings to flap two times.	In Java, when one object requests another object to carry out some action, the first objects sends a 'message' in the form of a call to a method in the second object.

Characteristic: An object-oriented program is a collection of interacting objects

Alice Realization:	*Mapping into Java:*
In Alice, the objects in the microworld work together to implement the story or game. The subparts of an object will work together to create a specific animation.	In Java, everything is a class, and the objects instantiated from those classes create the environment for the Java program. The Java program defines the way these objects interact.

Characteristic: The world is viewed as a collection of objects

Alice Realization:	*Mapping into Java:*

Alice creates a 'microworld' that holds all the objects under consideration. The object tree identifies all the objects contained in the microworld.
 	
	Java creates its world by defining classes and then instantiating objects. For example:
	
	`/*  Defining classes  */`
	`public class Customer() {}`
	`public class Boat() {}`
	`/*  Instantiating Customers  and Boats  */`
	`firstCustomer = new Customer();`
	`secondCustomer = new Customer();`
	`boatOne = new Boat();`
	`boatTwo = new Boat();`

Characteristic: Simple object-oriented programs implement a 3-tier architecture
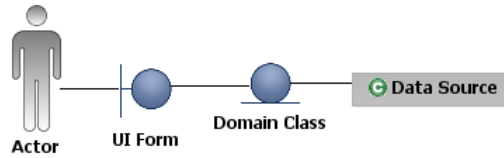
The UIForm, or 1st-Tier, is some user interface that provides information to and receives information from the set of Domain Classes.

The Domain Classes, or 2nd-Tier, are the template classes that are needed to implement the particular scenario being addressed.

The Data Source, or 3rd-Tier, is where persistent information is stored, most easily thought of as a database.

For our discussion of object-orientation and due to some limitations of Alice, we will restrict our remarks to the first two tiers – UIForm and Domain Classes. For ease of explanation, the UIForm will take the form of a Director of a play in Alice and as a Controller in Java. The role of each is to 'direct' / 'control' the action by sending messages to the 2nd-Tier, the Domain Classes. In each case, the Director's / Controller's sole responsibility is to 'direct' – object-one, do this; object-two, do that. In this way, the 1st-Tier is loosely coupled to the 2nd-Tier, implementing the idea of the tiered architecture.

Alice Realization:
The Alice world starts with a call to 'myFirstMethod'.

Assign 'myFirstMethod' the role of 'director'.
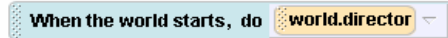
Using a slightly different example from joe-Penguin, we have 'director' giving instructions for the execution of Scene 1.

Mapping into Java:

The Java world starts with a call to 'main'.

```
public static void main(String[]
args)
```

'main' will take on the role of the 'controller', and directs the action. Here, we will use a very simple example to get the concept across.

```
/*  declare two objects  */
Customer aCustomer;
Boat aBoat;
/*  Instantiate the objects  */
aCustomer = new Customer();
aBoat = new Boat();
/*  Direct the action  */
aCustomer.setName(String Sally);
aBoat.setRegNumber(Int 1234);
aBoat.assignBoatToCustomer();
```
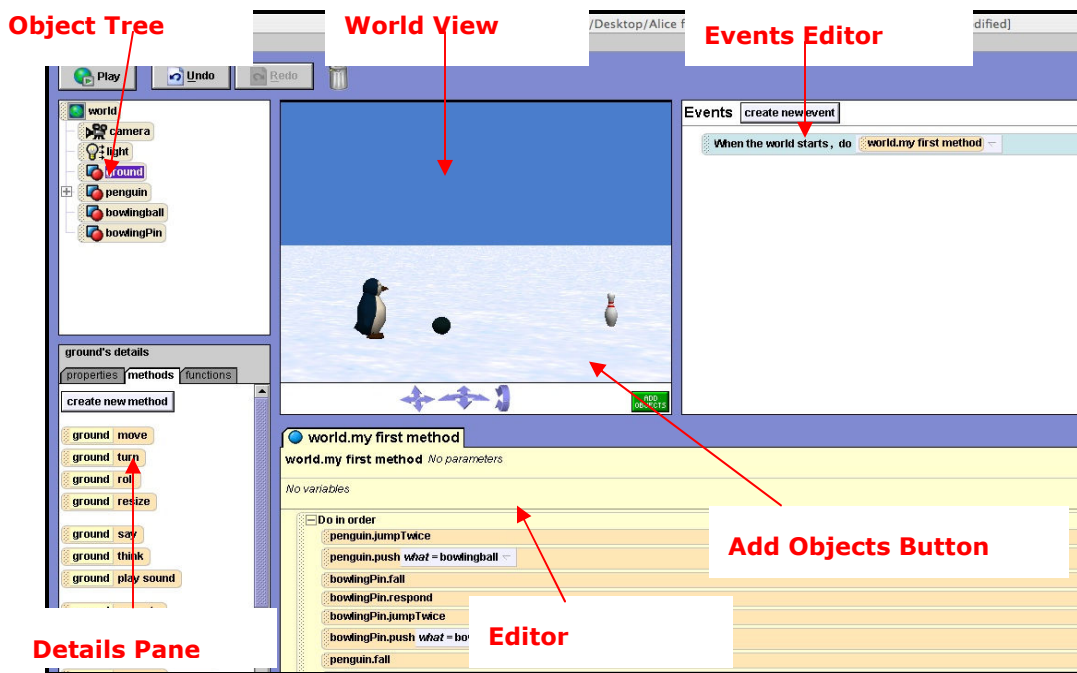
APPENDIX B

Figure 2: The Alice Interface
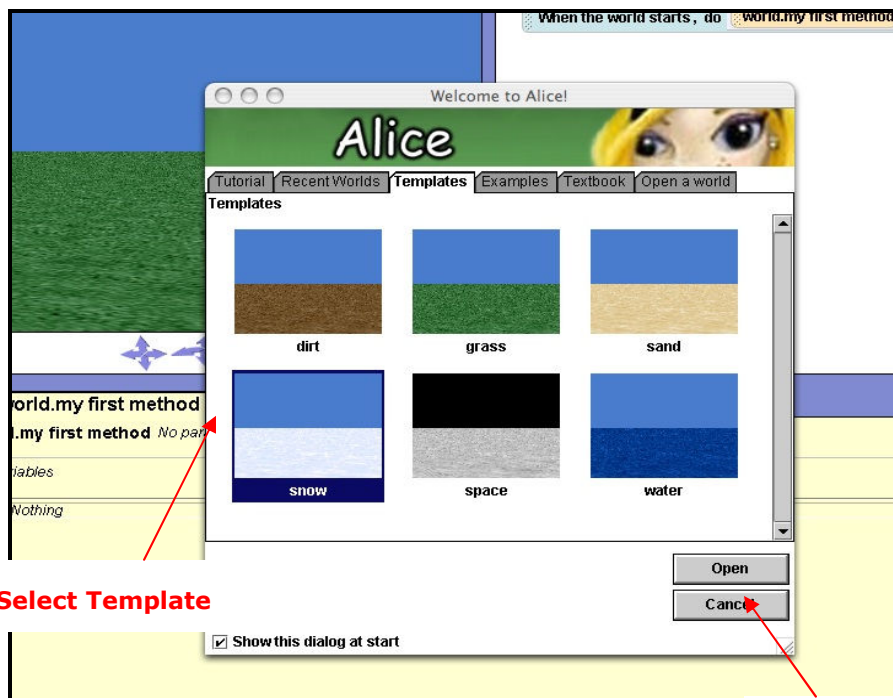
FROM FILE MENU, SELECT NEW WORLD

Figure 3: Creating a Microworld

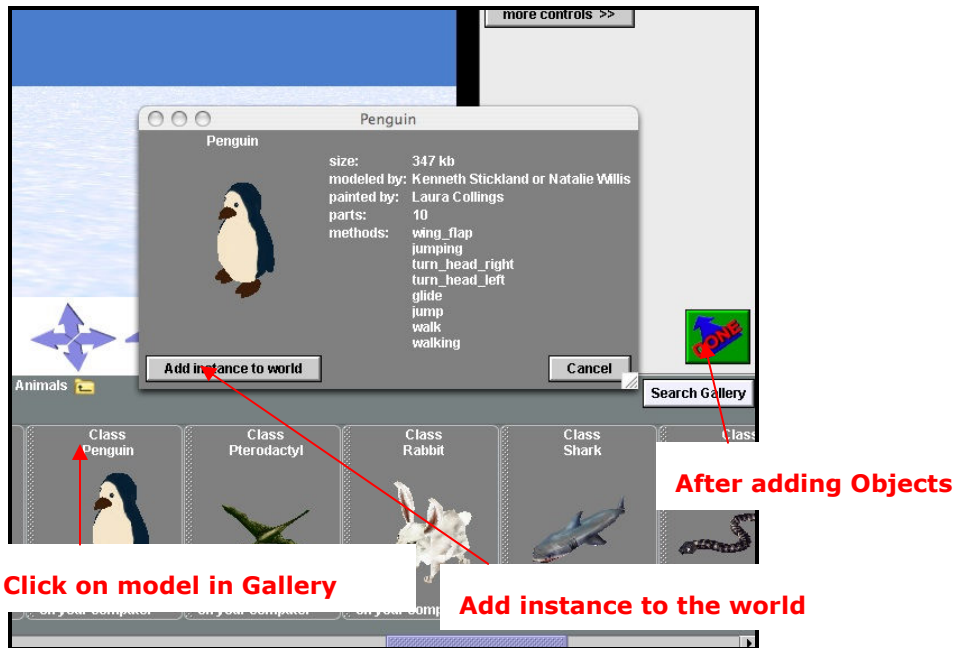*CLICK THE ADD OBJECTS BUTTON, AND FIND THE CLASS MODEL IN THE GALLERY*
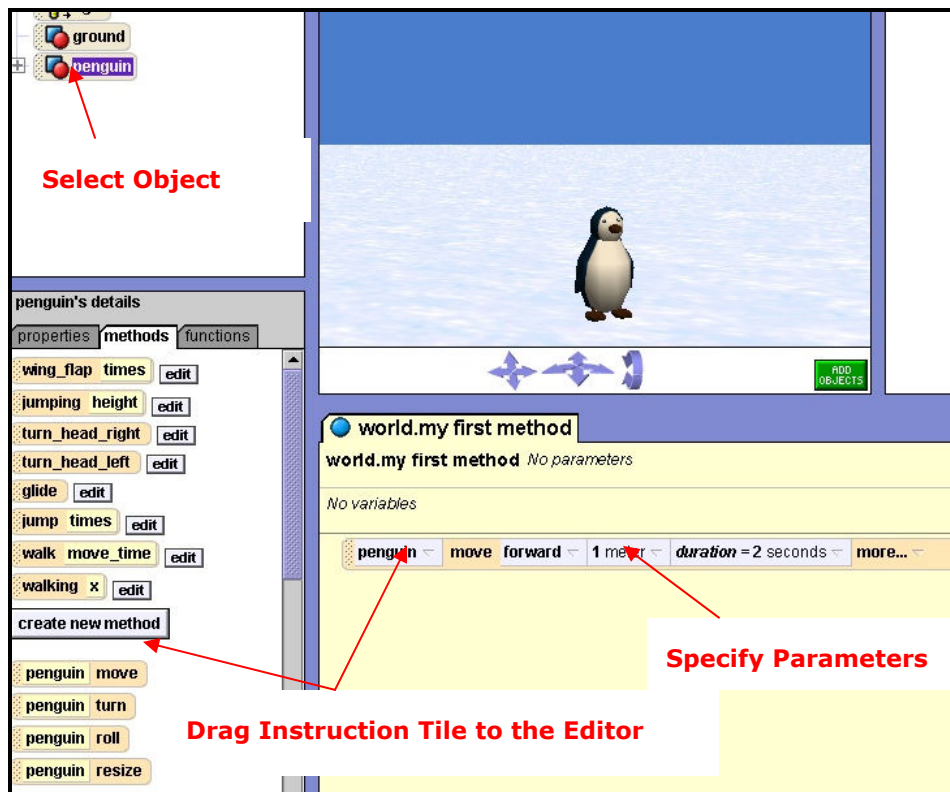

Figure 4: Adding an Object to the World

Figure 5: Creating the Program Code