Edith Visual Editor

Graham Baker
Walker Bohannan
Steve Marx
Vikram Nilakantan
Jessica Penick
Eli Spiegel

September 30, 2013

This document describes how a user of Edith, an educational software designed to teach young computer scientists the basics of programming, will be able to interact with the program by developing a story. The visual interface of Edith is designed to graphically represent bits of code and let the user manipulate physical objects instead of using text. Edith should effectively teach young programmers some of the building blocks of programming. The visual interface will support the user creating new methods, control structures, and conditional statements by dragging and dropping graphic representations of code together. The visual-editor must interact directly with the Object and Story creator in order to implement the user's visual abstraction of code. When the user completes the story, the visual interface will be able to convert the shapes and symbols into JavaScript. This visual representation will give young coders instant gratification encouraging them to progress to text-based programming.

Introduction

Edith is educational software designed to introduce new computer science students to programming. New students of any age can learn basic object-oriented programming concepts prior to learning a particular programming language in detail (syntax, data types and structures, etc.). Edith provides a graphical user interface through which users can create a "script" that animates a 2-D character, or sprite, provided by the program. The user creates the script by piecing together provided blocks of program structure in a functional way. Users may create games and share their creations with others. Similar programs (e.g., Alice) have been shown to increase the interest level and retention of students taking their first programming classes. Edith will be written primarily in JavaScript and will be available over the internet. After initial development, the software will be open-source to allow further development and additional educational opportunities.

The Visual Editor (VE) module will provide a visual programming language editor (in essence, an integrated development environment(IDE)). The VE will allow users to arrange programming elements to create an animation sequence; programming elements will include methods, variables, control statements, etc. The VE will, while enforcing syntax rules, convert the program into JavaScript, which will be used by other modules. The VE aims to create an interface that it is easy to use, and following its name will create a view that is always immediately ready to be seen. Since it will be using JavaScript, there will be no time to wait for compiling to see if the code "works," on the view of the end-user. Rather, the code, or sequence of events defined by the user will be always ready to run unless a script notifying the user otherwise.

The VE will be able to directly interact with two other Edith modules, Object Creator and Story Creator, while still being able to run independently. It will provide the Object Creator an interface for defining objects, and the Story Creator an interface for defining animations. In short, the VE will be able to run its own display view and integrate into the views of other Edith modules. The VE will provide not only an interface for the end-user to create their own code, but also for the other modules to implement code provided by the visual editor.

The user will be able to export his program in JavaScript/JavaScript Object Notation (JSON) format. JSON is an efficient data interchange text format that is language independent. The purpose of this is to again encourage learning of coding to our target users – young students who wish to learn more about what they are doing as far as coding, what their changes look like from a lower level than the visual editing.

Use Case 1	**Edit variable**
Primary Actor:	End-User
Preconditions:	Add method which does have a variable parameter
Postconditions:	Variable is defined.

Main Success Scenario:

1. User selects variable
2. User changes the parameter
3. Check typ

Extensions:

3.a Invalid type data:

1. Incorrect input.

1.a Boundary warning.

1.b Wrong type.

2. User returns to step 1 or exits.

3.b Compatible type:

1. System accepts input and saves parameter.

Non-Functional Requirements Needed:

1. Learning Experience.
2. Usability.

Use Case 2	**Dragging and dropping functionality for methods**
Primary Actor:	End-User
Preconditions:	There is a method for which it is possible to be selected.
Postconditions:	Method is now ready to be used.

Main Success Scenario:

1. User creates new variable and selects it.
2. They can now drag and drop the variable in the boundaries provid

---

Extensions:

2.a Invalid drop location:

1. The user attem ts to drag the variable outside of acceptable boundaries, the variable will no be ”locked” inside of boundary.

Non-Functional Requirements Needed:

1. Learning Experience.
2. Usability.

Use Case 3	**Instantiating a conditional statement**
Primary Actor:	End-User; d pper
Postconditions:	A conditional is instantiated.

Main Success Scenario:

1. A user drags and drops a conditional statement
2. They change the parameter (e.g. if this)
3. The inside of the conditional is then dragged and dropped (e.g. then that).

Non-Functional Requirements Needed:	• Learning Experience. • Usability.

Use Case 4	**Instantiating a Boolean Operator**
Primary Actor:	End-User
Postconditions:	A boolean operator is instantiated

Main Success Scenario:

1. A user drags and drops an operation (e.g. and, or, not)
2. the user sets the two variables or expressions

Non-Functional Requirements Needed:

1. Learning Experience.
2. Usability.

Use Case 5	**Connecting actions**
Primary Actor:	End-User
Preconditions:	There are two or more actions on the development board.
Postconditions:	The methods are connected.

Main Success Scenario:

1. The method is draged and droped by the user above or below the action they want to connect to
2. The user releases the method

Non-Functional Requirements Needed:	• Learning Experience. • Usability.

Use Case 6	**Save a Program**
Primary Actor:	End-User

Preconditions:	Add method which does have a variable parameter.
Postconditions:	Variable is defined.

Main Success Scenario:

1. The user selects save this adds the current development board to a list that the user can access

Extensions:

2.a Unnamed program:

1. The user will name the program

Non-Functional Requirements Needed:	• Usability.

Use Case 7	**Delete a method**
Primary Actor:	End-User
Preconditions:	There are methods on the development board
Postconditions:	Selected methods are deleted

Main Success Scenario:

1. The user selects a method or group of methods
2. The user selects to delete the selected items

Extensions:

2.a Invalid type data:

1. Incorrect input
2. User returns to step 1 or exits

Non-Functional Requirements Needed:	• Learning Experience. • Usability.

Use Case 8	**Run the Program (play)**
Primary Actor:	End-User
Preconditions:	Methods have been added to the development board.
Postconditions:	The program has been run and the state is maintained.

Main Success Scenario:

1. user selects the play button 
2. the program is compiled and if there are no errors the program is run.

Extensions:
2.a Compile time errors:

1. Does not run the program
2. Highlights error for user 

Non-Functional Requirements Needed:	• Learning Experience. • Usability.

Use Case 9	**Pause the Program** 
Primary Actor:	End-User
Preconditions:	The program is running.
Postconditions:	The state of the program when it was paused is maintained.

Main Success Scenario:
1. the user pauses the program
2. the program stops and maintains the current state 

Non-Functional Requirements Needed:	• Learning Experience. • Usability.

Use Case 10	**Instantiating a Loop**
Primary Actor:	End-User
Preconditions:	Add method which does have a variable parameter.
Postconditions:	A loop is instantiated

Main Success Scenario:
1. The user drags and drop a loop to the development board
2. The user then inputs the conditionals for the loop and its exit conditions 

Extensions:
2.a Invalid input:

1. User returns to step 1 or exits

Non-Functional Requirements Needed:	• Learning Experience. • Usability.

Use Case 11	**Creating a new Actio**
Primary Actor:	End-User
Preconditions:	None.
Postconditions:	If first method, the box is made, if attaching to another method, the boxes are connected

Main Success Scenario:

1. The user selects the new method button
2. Drags the method into the workbox
3. then the method prompts the user to give arguments for that specific action
4. If arguments are compatible, the method is created

Extensions:

2.a Invalid input:

1. User prompted to re-enter arguments

Use Case 12	**Exporting Project**
Primary Actor:	End-User
Preconditions:	Project contains atleast one method/action
Postconditions:	If successful, the user should get a message on how to share the file with another user.

Main Success Scenario:

1. The user selects the Export button
2. The system generates an export-ready file for the user
3. The system gives the user options on how to share the file

Extensions:

2.a No methods existed.

1. User prompted to add a method/action before trying to export again.

Non-Functional Requirements Needed:	• Usability.

Use Case 13	**Importing Project**
Primary Actor:	End-User
Preconditions:	User has a valid export file to import
Postconditions:	If successful, the visual editor should load the actions/methods present in the export file

Main Success Scenario:

1. The user selects the Import button
2. The system reads and loads in methods/actions from file
3. The user is now available to edit/add methods or actions to the file

Extensions:

1 Export file missing or corrupt

1.a User prompted to re-upload a file with the correct file structure.

Non-Functional Requirements Needed:	• Usability • Learning Experience

Use Case 14	**Add Sprite Object**
Primary Actor:	End-User
Preconditions:	None.
Postconditions:	New Sprite Object has been added to the visual editor and methods/actions can be added to it.

Main Success Scenario:

1. The user selects the New Sprite button
2. The user fills in basic details of the Sprite
3. The editor updates with the ability to add new methods/actions to the Sprite.

Extensions:

1 None.

Non-Functional Requirements Needed:	• Usability • Learning Experience

Use Case 15	**Delete Sprite Object**
Primary Actor:	End-User

Preconditions:	The selected Sprite object exists
Postconditions:	Selected Sprite object has been removed from visual editor.

Main Success Scenario:

1. The user selects a Sprite from the visual editor.
2. The user clicks on the Remove Sprite button 
3. The sprite and its associated methods/actions are removed from the visual editor

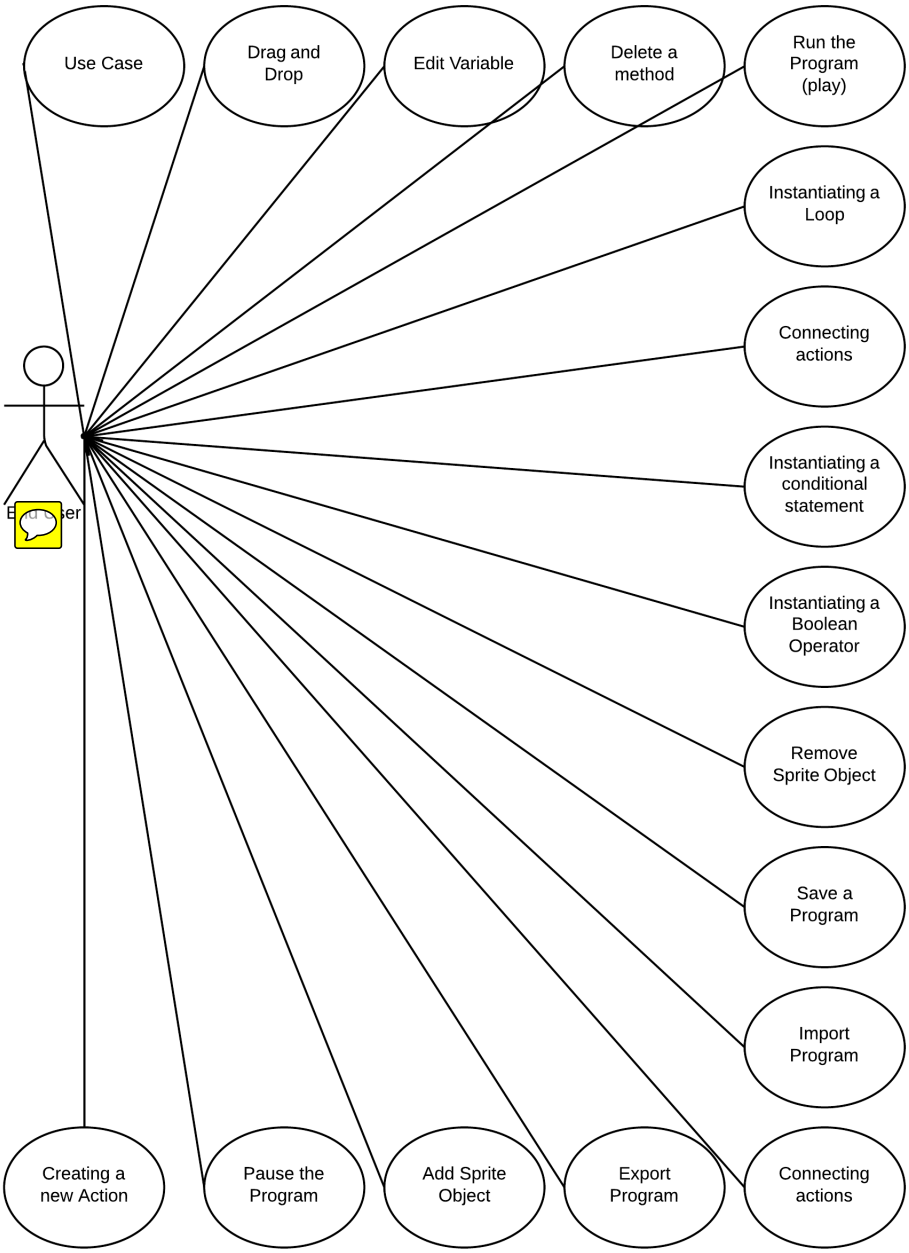
Extensions:

1 User selects 'Remove Sprite' when none exists

1.a. System tells user that a Sprite must be first created to remove it.

Non-Functional Requirements Needed:	• Usability • Learning Experience

Use Case Diagram



Non-Functional Requirements

Usability

Training should not be required for users to effectively use the module. Module documentation and basic age-specific computer knowledge should provide sufficient guidance. The module should allow the user to recover from errors with informative and understandable error messages.

Documentation

The module will provide documentation that allows users to easily begin using the system.

Modifiability

The module will have the flexibility to allow feature expansion over time.

Privacy

The user's work will remain private/secure unless the user chooses to share it.

Learning Experience

In order to support the learning of young computing students, the module will provide a sufficiently "pleasant" interface.

Platform

Users will be able to run the module from Windows or Mac

Open Source

After initial development, the source code will be open and redistributed with improvements by anyone.