2. Taranchuk, V. B. Development of interactive teaching materials for computer mechanics. / V. B. Taranchuk, M. A. Zhuravkov // Вестник Белорусского государственного университета. Серия 1, Физика. Математика. Информатика. – 2016. – № 3. – С. 97–107.

3. Таранчук, В. Б. Примеры создания и использования интеллектуальных учебных материалов / В. Б. Таранчук // Проблемы повышения эффективности образовательного процесса на базе информационных технологий : сб. материалов XII Междунар. науч.-практ. конф., Минск, 25 апр. 2019 г. / Белорус. гос. ун-т информатики и радиоэлектроники ; редкол. : Ю. Е. Кулешов [и др.]. – Минск : БГУИР, 2019. – С. 175–180.

4. Таранчук, В. Б. Практические аспекты разработки, сопровождения, использования интеллектуальных информационно-образовательных ресурсов / В. Б. Таранчук // Информатизация образования и методика электронного обучения : материалы III Междунар. науч. конф., Красноярск, 24–27 сент. 2019 г. : в 2 ч. / Сиб. федер. ун-т ; под общ. ред. М. В. Носкова. – Красноярск : Сиб. федер. ун-т, 2019. – Ч. 1. – С. 116–121.

5. Таранчук, В. Б. Методические и технические аспекты разработки адаптивных интеллектуальных обучающих систем / В. Б. Таранчук // Информатизация образования и методика электронного обучения: цифровые технологии в образовании : материалы VII Междунар. науч. конф., Красноярск, 19–22 сент. 2023 г. / Краснояр. гос. пед. ун-т ; под общ. ред. М. В. Носкова. – Красноярск : Краснояр. гос. пед. ун-т, 2023. – С. 1361–1365.

УДК 004.021

С. Н. ТКАЧ
Беларусь, Брест, БрГУ имени А. С. Пушкина

СПОСОБЫ ПРИМЕНЕНИЯ МЕХАНИЗМА «ЛЮБЫЕ КОМПОНЕНТНЫЕ БЛОКИ» В APP INVENTOR

В последнее время в учреждениях образования стали популярны визуализированные языки программирования (Scratch, APP Inventor и проч.) для изучения технологий программирования. С их помощью

удается в легкой игровой форме развивать логическое и алгоритмическое мышление обучающихся. При этом визуализированные языки содержат многие понятия классического программирования. Онлайн-платформа APP Inventor позволяет создавать приложения для телефонов или планшетов, используя визуальные блоки и другие инструменты визуализированных языков.

При написании приложений на визуальных языках программирования возникает необходимость создания однотипного кода для нескольких одинакового типа компонентов. Например, на сцене располагается девять кнопок *Кнопка1* – *Кнопка9*. При щелчке на каждой кнопке ее цвет должен стать красным. Конечно, можно создать один обработчик события *Кнопка1.Щелчок* (рисунок 1), а затем продублировать его 8 раз, изменяя входящие в него ссылки *Кнопка1* на ссылки на другие *Кнопки*.

Рисунок 1 – Примеры однотипных блоков программного кода для одинаковых компонентов

Однако в APP Inventor использован механизм «Любые компонентные блоки», реализующий принцип «Не повторяйся». Эти блоки позволяют обратиться к компонентам не по имени, а по признаку отношения их к одному классу (например, ко всем кнопкам сразу) или вхождения их в один список.

Вместо того, чтобы создавать много повторяющегося кода, можно использовать специальные блоки, называемые блоками *«Любой компонент»*. Чтобы получить доступ к блокам *Любого компонента*, нужно в режиме *Блоки* нажать на кнопку «+» слева от названия *«Любой компонент»*, затем выбрать компонент, например, *ДругаяКнопка*.

В раскрывшемся списке можно получить доступ к компонентам одного и того же класса (кнопки, акселерометры, изображения), а также к любым блокам этих компонентов, реализующим свойства, методы, события компонентов. Каждому из трех основных типов блоков компонентов, т. е. событиям, методам и свойствам, соответствуют блоки из раз-

дела *«Любой компонент»*. Рассмотрим несколько способов применения механизма *«Любые компонентные блоки»*.

1. Применение одного программного кода одновременно ко всем компонентам определенного класса, располагающимся на сцене.

Пусть, например, при изменении выбора *Флажка1* все кнопки, расположенные на сцене, должны стать зеленого цвета. В этом случае в обработчике события *Флажок1.Изменено* вместо обращения к каждой из кнопок по отдельности можно воспользоваться блоком *«Для каждого элемента в списке»* из *Управления* (рисунок 2).

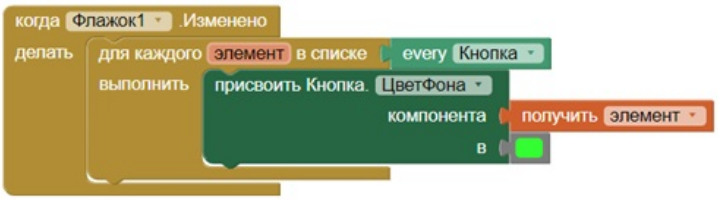

Рисунок 2 – Использование механизма *«Любые компонентные блоки»* для всех кнопок на сцене одновременно

Вместо ссылки на список в блок *«Для каждого элемента в списке»* в этом случае помещается ссылка на любую кнопку *«every Кнопка»*. Блок *«every Кнопка»* находится в перечне блоков *«Любой компонент»* – *«ДругаяКнопка»*. Параметр *«элемент»* внутри блока *«Для каждого элемента в списке»* является ссылкой на *Кнопку*, с которой сейчас работает блок, то есть в результате на каждую кнопку на сцене.

В результате применения блока *«Для каждого элемента в списке»* с параметром *«every Кнопка»* программный код после слова *«выполнить»* будет применен одновременно ко всем компонентам одного вида (например, кнопкам), расположенным на сцене.

2. Применение одного программного кода к любому компоненту определенного класса, расположенному на сцене.

Такой метод опирается на использование обработчиков событий компонента. Создается один обработчик события, например, щелчок по кнопке, где в качестве ссылки на компонент указывается не конкретный компонент *Кнопка1*, а общее название компонентов данного класса *Кнопка* (рисунок 3).

Программный код, прописанный в обработчике события, срабатывает не одновременно для всех компонентов одного класса, а только тогда,

когда наступает соответствующее событие. Таким образом программный код не применяется одновременно для всех компонентов.

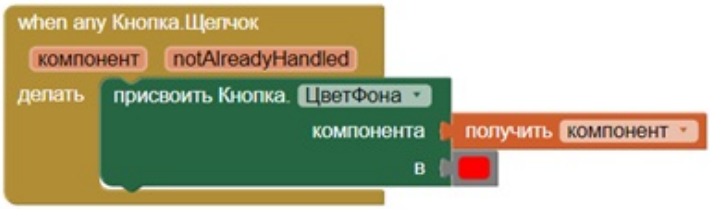

Рисунок 3 – Использование механизма *«Любые компонентные блоки»* для тех кнопок на сцене, для которых наступило событие *Щелчок*

3. Применение одного программного кода к компонентам определенного класса, перечисленным в *Списке*.

В случае, если программный код должен быть применен лишь к некоторым компонентам определенного класса на сцене, такие компоненты необходимо перечислить в списке (рисунок 4). В приведенном на рисунке 4 примере список создается как глобальная переменная *НекоторыеКнопки*. Элементами списка являются блоки-ссылки на соответствующие компоненты.

Рисунок 4 – Использование механизма *«Любые компонентные блоки»* для тех кнопок на сцене, которые перечислены в списке

Каждый исполняемый блок компонента состоит из трех частей: изменяемый компонент, часть компонента, которой манипулируют, и входы. Вместо того, чтобы привязывать блок к определенному компоненту, они позволяют сделать код более общим, предоставляя любой компонент одного и того же типа в качестве входа. Это позволяет, например, создать список кнопок *НекоторыеКнопки* и изменить *ЦветФона* только определенным кнопкам одновременно с помощью цикла *«Для каждого элемента в списке»* (рисунок 4).

Таким образом, в APP Inventor механизм *«Любые компонентные блоки»* реализует инкапсуляцию логики в некоторой конструкции программирования. Такая конструкция заменяет дублированный код для нескольких компонентов, что является эффективным по многим параметрам, в том числе и при необходимости внесения изменения в код программы.

УДК 004.4:519.17

Л. А. ЯРМОЛИК, В. А. ШЕИНА, А. И. ЖУК, Е. Н. ЗАЩУК
Беларусь, Брест, БрГТУ

ОСОБЕННОСТИ ИЗУЧЕНИЯ ТЕОРИИ ГРАФОВ В СИСТЕМЕ WOLFRAM MATHEMATICA

Целью настоящей работы является практическая реализация теории графов в одной из универсальных математических систем Wolfram Mathematica. Рассматриваются способы представления и задания графов, встроенные функции для работы с графами и их тестирования, способы нахождения кратчайшего пути [1]. Приведены рекомендации по использованию Mathematica при обучении теории графов [2, 3].

Визуально граф может быть представлен в виде конечного множества вершин, которые могут быть изображены точкой, и конечного множества их парных связей – линий, соединяющих соответствующие вершины (ребра). В пакете предусмотрены встроенные функции Graph и RandomGraph, которые позволяют изменять цвет, форму и размер вершин, цвет и стиль линии ребер, указывать вес ребер и отображать его на графе [2].

Например, на рисунке 1 сформирован следующий граф из шести вершин, при наведении курсора на который появляется информационное табло, которое позволяет без непосредственного введения встроенных функций вывести информацию о графе на экран. Например, проверить, является ли граф связным или гамильтоновым, содержит ли петли и циклы, вывести степени входа и выхода вершин и список ребер, матрицу смежности и инцидентности и др.