Вестник Сыктывкарского университета.
Серия 1: Математика. Механика. Информатика. 2022.
Выпуск 1 (42)
Вестник Сыктывкарского университета.
Серия 1: Математика. Механика. Информатика. 2022; 1 (42)

ИНФОРМАТИКА

Научная статья УДК

539.3
https://doi.org/10.34130/1992-2752_2022_1_61

Разработка языка разметки на основе XML Вадим

Александрович Мельников[1] **, Андрей Викторович Ермоленко**[2]
[1,2]Питирим Сорокин Сыктывкарский государственный университет,[1] ea74@list.ru

Аннотация. Современные подходы в области разработки программного обеспечения предполагают не только функциональность разрабатываемого продукта, но и удобство, понятность и привычность интерфейсов. Сегодня разработанное программное обеспечение может использоваться на различных устройствах, с различными конфигурациями, а пользователям для работы с ним может понадобиться и другой язык. Для решения проблемы универсальности в области 2D-игр предлагается подход, использованный при разработке пользовательского интерфейса для Sad Lion Engine. В рамках этого подхода предполагается использовать язык разметки Sad Lion Markup Language, описание и использование которого приведено в статье.

Ключевые слова: пользовательские интерфейсы, C++, мобильная разработка, языки разметки

Для цитирования: Мельников В.А., Ермоленко А.В. Разработка языка разметки на основе XML. *Вестник Сыктывкарского университета, Серия 1: Математика. Механика. Информатика*, 2022, № 1 (42), с. 61_ 73. https://doi.org/10.34130/1992-2752_2022_1_61

Язык разметки SadLion Markup Language

Вадим Андреевич Мельников[1] **, Андрей Васильевич Ермоленко**[2]
[1,2] Сыктывкарский государственный университет имени Питирима Сорокина

Аннотация. Современные подходы в области разработки программного обеспечения (ПО) предполагают не только функциональность разрабатываемого продукта, но и удобство, понят- ность и привычность интерфейсов. На сегодняшний день раз- рабатываемое ПО может использоваться на различных устрой- ствах, с различными конфигурациями, а также пользователям может быть необходим другой язык для работы с ПО. Для ре- шения вопроса универсальности в области 2D-игр предлагается подход, использованный при разработке пользовательского ин- терфейса игрового движка Sad Lion Engine. В рамках данного подхода предполагается использовать язык разметки Sad Lion Markup Language, описание и использование которого приведе- но в статье.

Ключевые слова: пользовательский интерфейс, Си++, разработка мобильных приложений, языки разметки

Для цитирования: Мельников В. А., Ермоленко А. В. Разработка языка разметки на основе XML // *Вестник Сыктывкарского универ- ситета. Сер. 1: Математика. Механика. Информатика.* 2022. Вып. 1 (42). С. *61-73.* https://doi.org/10.34130/1992-2752_2022_1_61

Введение

Пользовательский интерфейс так же важен, как и программный код. По оценкам разработчиков, конечный пользователь в первую очередь обращает внимание на интерфейс программы, а не на ее функциональность. Поэтому вопросам интерфейса всегда уделяется особое внимание.

После появления первой версии Unix в 1969 году программы были рассчитаны на работу в текстовых терминалах, и текстовый интерфейс был нормой. В середине 80-х годов стали появляться графические пользовательские интерфейсы (далее GUI), разработкой которых с начала 70-х годов занимался исследовательский центр Xerox Palo Alto Research Center. Этот исследовательский центр повлиял на разработку интерфейса Apple Macintosh, а через него и интерфейсов Microsoft Windows [1].

Сегодня необходимо не только создать пользовательский интерфейс для программы, но и спроектировать его с учетом интернационализации, ведь большинство современных программ локализуются на множество различных языков. А при разработке для мобильных устройств необходимо также учитывать существование на рынке устройств с различным соотношением сторон экрана, так как помимо уже привычного соотношения 9:16, распространенного среди телефонов, и 3:4, используемого для планшетов, существует также соотношение 9:19,5 для смартфонов Apple iPhone X .[1]

Для создания пользовательских интерфейсов на мобильных устройствах используются различные методы разработки макетов. Один из самых простых способов - использовать встроенный движок браузера (обычно такой компонент называется WebView) и писать код с использованием JavaScript и HTML, что позволяет Apache Cordova [2]. Но такой подход приведет к тому, что приложение будет изначально медленным, раздражая пользователей. Второй путь - создание нативных приложений для Android [3] и iOS [4] с помощью стандартных средств верстки или кроссплатформенного Xamarin [5]. Оба способа имеют общие черты и очень похожи на старую технологию Windows Presentation Foundation (WPF) [6]. На первый взгляд может показаться, что технология, которая будет описана ниже, изобретает HTML и JS заново, но следует отметить, что при использовании HTML и JS на довольно слабых (по сравнению с ПК на базе архитектуры x86) мобильных платформах это приводит к увеличению энергопотребления и снижению производительности приложений. Николай Армоник в своем докладе "Как мы сделали самую большую игру на Defold за 1 год" на конференции DevGAMM Moscow 2018 приводит цифры по производительности их приложения: 20-40 % времени кадра занимает выполнение Lua-скриптов, Lua - это интерпретируемый язык, как и JavaScript, и они сопоставимы по производительности. Отказ от интерпретируемых языков и написание всей логики на C++ значительно повышает производительность приложения.

Нативный подход к разработке вынуждает разработчиков поддерживать несколько совершенно несовместимых кодовых баз - одну на Java или Kotlin для Android, другую на Objective-C или Swift для iOS в случае мобильных приложений. Если вы разрабатываете для Windows, вам придется добавить UWP. Частично проблему решает использование Xamarin Forms [5], которые позволяют обобщить большую часть кода и только в особых случаях писать код с использованием Xamarin Native. Отдельно стоит технология Flutter [7], в которой нативные элементы пользовательского интерфейса не используются, а элементы рисуются сразу на

[1]По сайтам: https://deviceatlas.com/blog/most-used-smartphone-screen- resolution, https://www.apple.com/ru/newsroom/2017/09/the-future-is-here-iphone- x/

на экране с помощью Skia[2] - библиотеки векторной графики Google. Такой подход очень похож на проектирование интерфейсов в играх. Для программирования во Flutter используется язык Dart, который, как и JavaScript, ориентирован на веб-приложения и пользовательский интерфейс. Dart не такой мощный, как C++, а в играх производительность имеет решающее значение.

Приведенный обзор современного подхода к созданию интерфейсов показывает, что востребованной задачей является создание универсальных, легко программируемых интерфейсов, адаптирующихся к различным устройствам и экранам, с разным соотношением сторон. Для решения этой задачи предлагается использовать игровой движок Sad Lion Engine (SLE) [8] пользовательского интерфейса, созданный с помощью C++, на основе разработанного языка разметки для максимальной производительности приложения. Соответственно, целью статьи является описание возможностей разработанного интерфейса игрового движка SLE. Стоит отметить, что существует мощный фреймворк Qt [9] для разработки приложений для ПК и мобильных платформ, но он изначально не рассматривался из-за непомерной стоимости для небольшой студии разработчиков.

Такой подход применим как к бизнес-приложениям, так и к любым играм (2D, 3D, AR, VR), ведь даже в 3D-играх интерфейс остается двухмерной текстурой, размещенной в пространстве.

Обзор публикаций в научно-технической литературе показывает, что, несмотря на стремительное развитие языков и сред программирования, разработчики мобильных игр сталкиваются с достаточно большим количеством разнообразных трудностей при создании игровых приложений и интерфейсов. Например, общие проблемы игрового дизайна, в том числе и при разработке мобильных приложений, поднимаются в работах [10; 11]. Разработчики рассматривают различные подходы к решению проблем, связанных с интерфейсами. В работе [12] авторы представляют язык XVGDL для описания игр, основанный на расширяемом языке разметки XML, подчеркивая при этом универсальность XVGDL, описывая синтаксис и составные части XVGDL, а также примеры их использования. В исследовании [13] авторы представляют подход к автоматизированному созданию классических многопользовательских 2D-игр на базе Android, основанный на концепции проектирования на основе моделей. Популярной тенденцией также остается разработка легких по производительности игровых движков. Однако при этом возникают вопросы их развития, внедрения новых технологий и функциональности, предоставляемой современными и перспективными устройствами [14].

[2]https://skia.org/ - Сайт библиотеки векторной графики Skia.

Первая версия языка разметки для Unity3D

Первая версия Sad Lion Markup Language (SLML) полностью работала в Unity3D. Для создания интерфейса использовалась система компонентов [10] и набор компонентов самого движка Unity3D. Жизненный цикл приложения также был основан на Unity3D [11]. Появление SLML в Unity3D обосновано сложностью создания качественного адаптивного интерфейса с помощью стандартных средств движка. Довольно сложно выровнять элементы пользовательского интерфейса так, чтобы они хорошо смотрелись на разных соотношениях экранов.

Появление SLML и SLE началось с решения проблемы больших размеров исполняемых файлов. При использовании спрайтов Unity3D встраивал их в распакованный исполняемый файл, что приводило к значительному увеличению размера приложения. При дальнейшей разработке выяснилось, что добавление новых объектов также приводит к сильному увеличению размера исполняемого файла. Для решения этой проблемы со спрайтами был создан специальный инструмент для создания атласов в PNG-файлах. Решение проблемы с объектами потребовало создания собственной системы создания объектов в сцене, минуя Unity3D, что и послужило причиной появления SLML.

SLML не создает объекты заранее, все описания хранятся в XML-файлах, объекты задаются на лету при запуске игры. Такой подход позволил избежать встраивания кода объектов в бинарный код, вместо этого они генерируются в соответствии с описанием из XML-файла по мере необходимости.

В SLE масштабирование происходит после создания всех объектов. В Unity3D игра выполняется в цикле Awake-Start-Update, что приводит к следующей проблеме: необходимо создавать объекты в фреймворке, в котором нет возможности создавать до начала цикла. Кроме того, настраивать ширину-высоту с помощью якорей в Unity3D не очень удобно.

Описанные выше трудности указали на необходимость разработки собственного ядра SLE.

Реализация в рамках SLE

SLML основан на XML [17]. В рамках SLE были решены проблемы, связанные с жизненным циклом приложения в Unity3D, жизненный цикл SLE предполагает полное создание дерева компонентов до первого вызова Awake любого из компонентов, указанных в XML-файле. Использование библиотеки Skia позволило добавить ряд новых функций в систему пользовательского интерфейса.

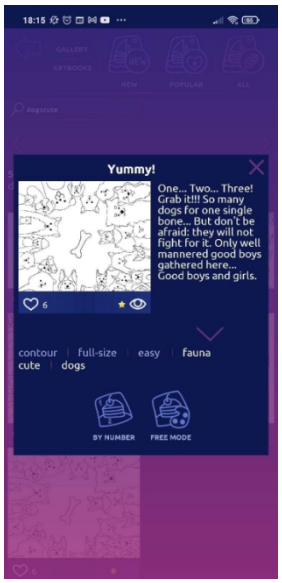

Рис. 1. Экран с соотношением сторон 9:16

Основная идея SLML заключается в использовании адаптивной верстки, при которой изначально задается несколько (обычно три) верстоk, а затем пользовательский интерфейс интерполируется на основе одной из существующих верстоk. Если существует взаимосвязь, для которой качественная интерполяция не работает, то она может быть реализована отдельно, не затрагивая сам движок.

Продемонстрируем преимущества SLE на примере проекта Big Coloring Book[3] , используя макет формы, выполненный для одного и двух коэффициентов. В текущей версии SLML дополнительные макеты получаются с помощью переменных, хранящихся в JSON-файле и подставляемых при разборе XML-файлов.

Возьмем за основу соотношение сторон, близкое к 9:16 (Xiaomi Redmi 4X), результат показан на рис. 1. На рис. 2 показано, как будет выглядеть интерфейс в соотношении, близком к 3: 4 (DEXP E170), полученный путем пропорционального масштабирования интерфейса, созданного для соотношения 9:16. Сразу же бросается в глаза сильное нарушение пропорций, всплывающее окно по ширине значительно больше, чем по высоте, а также элементы начинают наезжать друг на друга.

На рис. 3 показан результат, полученный при использовании отдельного соотношения 3:4. Такой подход позволил нам получить более качественный результат для каждого соотношения сторон, пропорции элементов и расстояния между ними сохраняются. По мнению профессиональных дизайнеров, в этом случае с помощью оптического выравнивания изображение становится визуально сбалансированным и гармоничным для восприятия пользователем

[3]Приложение в магазине Google Play - https://play.google.com/store/apps/details?id= sadlion.games.bigcoloringbook.

Рис. 2. Экран с соотношением сторон 3:4, полученный путем масштабирования из макета с соотношением сторон 9:16

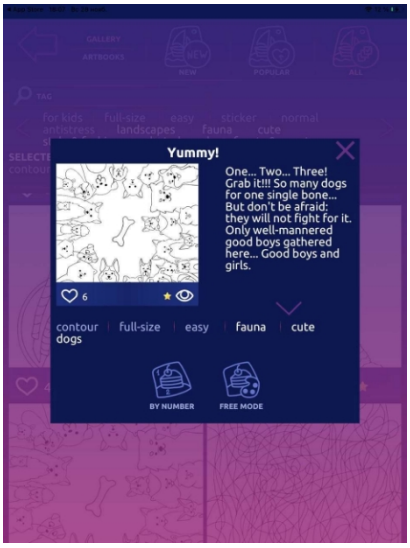

Рис. 3. Экран с соотношением сторон 3:4

на интуитивном уровне. Блок с текстом имеет достаточно свободного пространства вокруг себя и связан с точки зрения визуальной эргономики. Если соотношение сторон не из списка тех, что реализованы заранее, то произойдет пропорциональное масштабирование от ближайшего соотношения. Это позволяет учесть особенности соотношения сторон экрана, большую ширину и, наоборот, высоту, и расположить элементы интерфейса более точно.

В SLM есть несколько основных тегов, которые обеспечивают базовую функциональность:

- *Form* - это корневой тег для файла макета;
- *Row* - это тег ряда, который принимает всю ширину и заданную высоту родителя, ряды размещаются один за другим;
- *Col* - это тег колонки, который принимает заданную ширину и всю высоту родителя, колонки располагаются одна за другой;
- *Layer* - это тег слоя, который занимает все пространство родительского слоя и помещается в точку (0, 0) относительно родительского слоя, слои перекрывают друг друга;
- *Button* - это тег кнопки, который дает возможность обрабатывать жесты пользователя, такие как касания и движения пальцем;
- *Изображение* представляет собой растровый спрайт;
- *Текст* - это однострочный текст;

Теперь рассмотрим наиболее важные атрибуты для тегов:

- *width* - в случае тега Form может иметь только абсолютное значение в пикселях, задает ширину экрана, в случае остальных Col может быть выражен в процентах и задает ширину колонки, в остальных случаях игнорируется;
- *height* - в случае тега Form может иметь только абсолютное значение в пикселях, задает высоту экрана, в случае остальных тегов Row может быть выражен в процентах и задает высоту строки, в остальных случаях игнорируется;
- *name* - задает строковое имя объекта, по которому его можно найти в коде C++;
- *aspectRatioFitter* - способ управления соотношением сторон объекта, теперь поддерживается только FitInParent;
- *components* - атрибут, содержащий имена компонентов, которые должны быть подключены к объекту, в версии C# это было сделано с помощью отражения, в C++ - с помощью регистра инстанциаторов объектов;

- *text* - строковый атрибут, задающий значение тегу Text;
- *группа onUp, onDown, onClick* - атрибуты событий кнопок, в которых хранятся имена обработчиков событий. Данная функциональность реализуется с помощью реестра обработчиков событий в каждой форме;
- *sprite* - строковые константы в формате "atlas/index", где atlas - название атласа, а index - номер спрайта в атласе.

Это список основных атрибутов, остальные атрибуты обеспечивают только визуальные эффекты, такие как градиенты, рамки и т.д. Они представляют интерес только для целей документации. Ниже приведен пример фрагмента исходного кода для проекта Big Coloring Book:

```
<Row height="24.5">
   <Col color="255/0/255/0">
   <Text name="txtMyWorks"
      font-size="28"
      font="Ubuntu-Bold"
      color="+/lightBlue"/>
   </Col>
</Row>
```

Заключение

После перехода с Unity3D на Xamarin C# [8] функциональность движка и SLML была увеличена, жизненный цикл адаптирован к требованиям SLE и SLML, а скорость загрузки приложений значительно возросла.

После перехода на C++ производительность значительно выросла, но гибкость и простота разработки на C# были утрачены. В C++ нет отражения, и все компоненты и обработчики приходится регистрировать вручную, а IDE Android Studio также оставляет желать лучшего в плане производительности по сравнению с Visual Studio от Microsoft: долго компилируется, нестабильно работает отладчик (при работе со смартфоном xiaomi redmi 4 отладчик отключается в точке стоп), медленно подсказывает возможные имена функций (а при размере приложения в 20 тысяч строк невозможно держать все в памяти), но Xamarin C# не обладал достаточной стабильностью.

Язык SLML находится в постоянном развитии и продолжает меняться вместе с SLE, из последних изменений отметим функции, касающиеся

позиционирование объектов и смещение соседних объектов при изменении высоты одного из них.

Список источников

1. **Раго А. С., Стивенс В. Р.** Продвинутое программирование в среде UNIX. Addison-Wesley Professional, 2013. 1032 p.

2. **Кэмден К. Р.** Апачи Кордовы в действии. Шелтер-Айленд: Manning, 2016. 230 p.

3. **Торнсби Дж.** Дизайн пользовательского интерфейса Android. Бирмингем: Packt Publishing, 2016. 356 p.

4. **Беннетт Г., Качмарек С., Лис Б.** Свидт 4 для абсолютных новичков. Phoenix: Apress, 2018. 317 p.

5. **Петцольд К.** Кроссплатформенное программирование на C# для iOS, Android и Windows. Редмонд, Вашингтон: Microsoft Press, 2013. 1161 p.

6. **Петцольд К.** Приложения = Код + Разметка. Руководство по Microsoft Windows Presentation Foundation. Редмонд: Microsoft Press, 2006. 1002 p.

7. **Ветряная мельница Э.** Флаттер в действии. Остров Шелтер: Manning, 2020. 368 p.

8. **Мельников В.А.** Процесс разработки движка для 2D-игр и интерфей- сов Sad Lion Engine // *Вестник Сыктывкарского университета. Сер.1: Математика. Механика. Информатика*, 2019. Вып. 4 (33). С. 21-37.

9. **Dogsa T., Meolic R.** A C++ App for Demonstration of Sorting Algorithms on Mobile Platforms // *Международный журнал интерактивных мобильных технологий*, 2014 Том 8, № 1. URL: https://www.online-journals.org/index.php/i-jim/article/view/3464/2940 (дата обращения: 17.07.2020).

10. **Cao J., Cao Y.** Применение интерфейса взаимодействия человека и компьютера в дизайне игр // *Communications in Computer and Information Science*, 2017, 714, pp. 103-108. DOI: 10.1007/978-3-319-58753-0_16.

11. **Zaina L. A. M., Fortes R. P. M., Casadei V., Nozaki L. S., Paiva D. M. B.** Предотвращение барьеров доступности: Руководство по использованию паттернов проектирования пользовательского интерфейса в мобильных приложениях // *Journal of Systems and Software*, 2022, vol. 186. DOI: 10.1016/j.jss.2021.111213.

12. **Quinones, J. R., Fernandez-Leiva, A. J.** XML-Based Video Game Description Language // *IEEE Access*, 2020, vol. 8, pp. 4679-4692. DOI: 10.1109/ACCESS.2019.2962969.

13. **Дерахшанди М., Колахдуз-Рахими С., Троя Ж., Лано К.** Модельно-управляемый фреймворк для разработки классических многопользовательских 2D настольных игр на базе андроид // *Автоматизированная разработка программного обеспечения*, 2021, 28 (2). DOI: 10.1007/s10515-021-00282-1.

14. **Park, H. C., Baek, N.** Design of SelfEngine: A Lightweight Game Engine // *Lecture Notes in Electrical Engineering*, 2020, vol. 621, pp. 223-227. DOI: 10.1007/978-981-15-1465-4_23.

15. **Вест М.** Эволюционируйте свою иерархию [Онлайн] // Ковбойское программирование. URL: http://cowboyprogramming.com/2007/01/05/evolve-your-heirachy/ (дата обращения: 22.08.2020).

16. **Хокинг Дж.** Unity в действии: Разработка мультиплатформенных игр на C#. Manning, 2018. 400 p.

17. **Лисовский К.Ю.** Разработка XML-приложений на языке Scheme // *Программирование и компьютерные программы 28*, с. 197-206 (2002). https://doi.org/10.1023/A:1016319000374

Ссылки

1. **Раго А. С., Стивенс В. Р.** *Продвинутое программирование в среде UNIX*. Addison-Wesley Professional, 2013. 1032 p.

2. **Кэмден К. Р.** *Апачи Кордовы в действии*. Шелтер-Айленд: Manning, 2016. 230 p.

3. **Торнсби Дж.** *Дизайн пользовательского интерфейса Android*. Бирмингем: Packt Publishing, 2016. 356 p.

4. **Беннетт Г., Качмарек С., Лис Б.** *Свидт 4 для абсолютных новичков*. Phoenix: Apress, 2018. 317 p.

5. **Петцольд К.** *Кроссплатформенное программирование на C# для iOS, Android и Windows*. Редмонд, Вашингтон: Microsoft Press, 2013. 1161 p.

6. **Петцольд К.** *Приложения = Код + Разметка. Руководство по Microsoft Windows Presentation Foundation*. Редмонд: Microsoft Press, 2006. 1002 p.

7. **Ветряная мельница Э.** *Флаттер в действии*. Остров Шелтер: Manning, 2020. 368 p.

8. **Мельников В. А.** Процесс разработки ядра игрового движка для 2D игр и интерфейсов Sad Lion Engine. *Вестник Сыктывкарского университета. Сер.1: Математика. Механика. Informatika* [Bulletin of Syktyvkar University. Серия 1: Математика. Механика. Информатика], 2019, 4 (33), с. 21-37. (In Russ.).

9. **Догса Т., Меолик Р.** Приложение на C++ для демонстрации алгоритмов сортировки на мобильных платформах. *Международный журнал интерактивных мобильных технологий*, 2014. Vol. 8, No 1. Доступно: https://www.online-journals.org/index.php/i-jim/article/view/3464/2940 (accessed: 17.07.2020).

10. **Cao J., Cao Y.** Применение интерфейса взаимодействия человека и компьютера в дизайне игр. *Коммуникации в компьютерных и информационных науках*, 2017, 714, с. 103-108. DOI: 10.1007/978-3-319-58753-0_16.

11. **Зайна Л. А. М., Фортес Р. П. М., Касадей В., Нозаки Л. С., Пайва D. M. Б.** Предотвращение барьеров доступности: Руководство по использованию паттернов проектирования пользовательского интерфейса в мобильных приложениях. *Журнал систем и программного обеспечения*, 2022, том 186. DOI: 10.1016/j.jss.2021.111213.

12. **Куинонес, Х. Р., Фернандес-Лейва, А.** X. Язык описания видеоигр на основе XML. *IEEE Access*, 2020, vol. 8, pp. 4679-4692. DOI: 10.1109/ACCESS.2019.2962969.

13. **Дерахшанди М., Колахдуз-Рахими С., Тройя Ж., Лано К.** Основа для разработки классических многопользовательских 2D настольных игр на базе андроид, управляемая моделями. *Автоматизированная разработка программного обеспечения*, 2021, 28 (2). DOI: 10.1007/s10515-021-00282-1.

14. **Парк, Х. К., Бэк, Н.** Дизайн SelfEngine: A Lightweight Game Engine. *Lecture Notes in Electrical Engineering*, 2020, vol. 621, pp. 223-227. DOI: 10.1007/978-981-15-1465-4_23.

15. **Вест М.** Эволюционируй свою иерархию [Онлайн] *Ковбойское программирование*. Available: http://cowboyprogramming.com/2007/01/05/evolve-your-heirachy/ (accessed: 22.08.2020).

16. **Хокинг Дж.** *Unity в действии: Разработка мультиплатформенных игр на C#.* Manning, 2018. 400 p.

17. **Лисовский, К.Я.** Разработка XML-приложений на языке Scheme. *Программирование и компьютерное программное обеспечение 28*, 197-206 (2002). https://doi.org/10.1023/A:1016319000374

Сведения об авторах / Информация об авторах Вадим Андреевич Мельников / Вадим Алексеевич Мельников аспирант / Аспирант
Сыктывкарский государственный университет им. Питирима Сороки-на / Питирим Сорокин Сыктывкарский государственный университет
167001, Россия, г. Сыктывкар, Октябрьский пр-т, 55 / 167001, Россия, Сыктывкар, Октябрьский пр-т, 55

Андрей Васильевич Ермоленко / Андрей Васильевич Ермоленко к.ф.-м.н., доцент, заведующий кафедрой прикладной математики и компьютерных наук / к.ф.-м.н. физико-математических наук, доцент, заведующий кафедрой прикладной математики и информатики Сыктывкарский государственный университет им. Питирима Сороки-на / Питирим Сорокин Сыктывкарский государственный университет
167001, Россия, г. Сыктывкар, Октябрьский пр-т, 55 / 167001, Россия, Сыктывкар, Октябрьский пр-т, 55

Статья поступила в редакцию / Статья была представлена 12.03.2022 Одобрено после рецензирования / Одобрена после рецензирования 24.03.2022 Принято к публикации / Принято к публикации 25.03.2022