ИНФОРМАТИКА

Вестник Сыктывкарского университета.
Серия 1: Математика. Механика. Информатика.
Выпуск 4 (33). 2019

УДК 539.3

ПРОЦЕСС РАЗРАБОТКИ ДВИЖКА ДЛЯ 2D ИГР И ИНТЕРФЕЙСОВ SAD LION ENGINE

В. А. Мельников

Статья описывает процесс поиска решений проблем, возникающих при разработке игр, на основе опыта, полученного в Sad Lion Studio. Рассматриваются готовые движки и описан путь появления собственного ядра игрового движка Sad Lion Engine.

Ключевые слова: C++, разработка игр, игровые движки, интерфейсы, архитектура игрового движка.

1. Введение

Разработка игр является технически сложным процессом. Современные игры выпускаются на множестве устройств, что остро ставит проблему совместимости и высокие требования к производительности. Несмотря на постоянно растущие мощности современных устройств, большое количество пользователей продолжает использовать даже сильно устаревшие модели, которые необходимо поддерживать. А после завершения разработки начинается цикл поддержки и развития

продукта, в течение которого приходится адаптировать игру под новые устройства, для использования самых современных технических возможностей. Со временем возникает необходимость добавлять новый функционал или модифицировать старый. Для снижения трудозатрат во время поддержки архитектура игры должна быть изначально хорошо спроектирована.

Что такое хорошая архитектура? На мой взгляд, основной признак хорошей архитектуры — простота внесения изменений в неё. В любые современные игры и программное обеспечение будут вноситься изменения, и затраты времени и средств для внесения этих изменений должны быть минимальны [1].

С точки зрения информатики игры — нестрогие, интерактивные, основанные на агентах системы компьютерной симуляции реального времени. Игры также имеют сходство с итеративными численными моделями: игровой цикл выполняется множество раз, и на каждом шаге различные системы вычисляют и обновляют свое состояние [2].

Во время разработки игр накапливается достаточно большая кодовая база, и её отдельные части можно переиспользовать при разработке других проектов для сокращения трудозатрат. Примером такого переиспользования могут послужить классы для рендеринга графики, работы с файловой системой на различных операционных системах, работа с аудиосистемой, чтение пользовательского ввода. Эти классы создают слой абстракции, который можно назвать игровым движком. Игровой движок или может быть оформлен в виде полноценной среды разработки со своим редактором сцен, ресурсов и т. д., или же может быть встраиваемым в другие среды разработки. Примером первой парадиг-

мы могут послужить Unity3D[1], Unreal Engine[2], CryEngine[3], примером второй парадигмы. URL: MonoGame[4], Cocos-2D[5] (хотя Cocos-2D имеет и редактор и может относиться к первой парадигме).

Разные игровые движки предоставляют разный уровень абстракции от операционной системы, на которой будет работать игра. Например, Unreal Engine предоставляет возможность написания кода на C++ и настройки всего цикла рендеринга, а Corona SDK[6], который использует скриптовый язык Lua [3], полностью абстрагирует пользователя от прямого взаимодействия с графикой и памятью. Как правило, игровые движки предоставляют довольно высокоуровневую абстракцию от операционной системы. В настоящее время для разработки игр популярен C#. С одной стороны, это ведёт к тому, что придётся обслуживать только одну кодовую базу, которая будет работать на всех устройствах, поддерживающих движок. С другой стороны, чем сложнее абстракция, тем дороже она стоит с точки зрения времени выполнения.

Большое количество сложных конструкций, такие как наследование и виртуальные функции, будут продуцировать сложный машинный код, также полное осознание принципов работы такого движка довольно проблематично. Соответственно, на выходе получается исполняемый файл заведомо худшего качества, оптимизация которого вызывает много проблем. Использование, например, C# упрощает написание кода,

[1]Сайт Unity3S. URL: https://unity.com/ru
[2]Сайт Unreal Engine 4. URL: https://www.unrealengine.com/en-US/
[3]Сайт CryEngine. URL: https://www.cryengine.com/
[4]Сайт MonoGame. URL: http://www.monogame.net/
[5]Сайт Cocos-2D. URL: https://www.cocos.com/en/
[6]Сайт Corona SDK. URL: https://ru.coronalabs.com/

а также позволяет не задумываться о проблемах с утечками памяти, сборщик мусора их не допустит [4]. С другой стороны, такие языки значительно медленнее во время выполнения, плюс сам сборщик мусора занимает довольно много процессорного времени.

В SadLion Studio (ООО «СыкГеймЛаб»)[7] для разработки игр, ориентированных на мобильные платформы, изначально использовался Unity3D. По мере работы и решения различных проблем (большой вес выходного исполняемого файла, неудобные средства разработки пользовательского интерфейса, сложность развёртки и отладки на устройстве) решено разработать свой игровой движок, позволяющий:

- получить удобный доступ к нативным методам операционной системы;
- снизить объёмы межъязыкового взаимодействия;
- наладить быструю сборку и отладку;
- оптимизировать размер исполняемого файла;
- вёрстку адаптивных интерфейсов.

Разработка движка происходила в несколько этапов:

1. Рассмотрение существующих аналогов и анализ реализации поставленных требований к движку, если они выполнены.
2. Проектирование архитектур ядра движка.
3. Выбор языка реализации.

[7]Оифициальная страница Sdlion Stuido VK. URL: https://vk.com/sadliongames

4. Реализация, тестирование и отладка.

2. Аналоги

CryEngine и Unreal Engine являются большими движками со своими средами разработки для создания AAA-игр. Описание того, что такое AAA-игры и какие сложности встречаются при их разработке, можно найти в работе [5]. Также правообладатели этих движков берут плату в проценте от дохода игры после её запуска. По этим причинам эти два движка были отброшены как возможные инструменты для работы.

MonoGame работает на платформе Xamarin, которая предоставляет неплохой уровень доступа к системным функциям, но работа там происходит на языке C#, что снижает производительность, критичную для мобильных устройств.

Cocos-2D предоставляет слишком высокий уровень абстракции и на момент рассмотрения не имел поддержки рендерных систем Metal и Vulkan [6; 7].

Unity3D был изначальным инструментом для разработки, его минусы и плюсы будут рассмотрены далее.

3. Первая версия ядра движка Sad Lion Engine

Первая версия ядра движка Sad Lion Engine (SLE) была представлена в декабре 2018 года как плагин для Unity3D, которая решала две проблемы: большой исполняемый файл и неудобные средства для разработки пользовательского интерфейса. Для проекта Quiz Challenge[8], вы-

[8]Страница проекта Quiz Challenge в Google Play. URL: https://play.google.com/store/apps/details?id=urmobi.games.quizchallenge

полненном с использованием новых технологий, было достигнуто улучшение по размеру исполняемого файла приблизительно в три раза по сравнению с обычным инструментарием Unity3D.

На тот момент было реализовано две функции движка, которые заменили собой функции Unity3D:

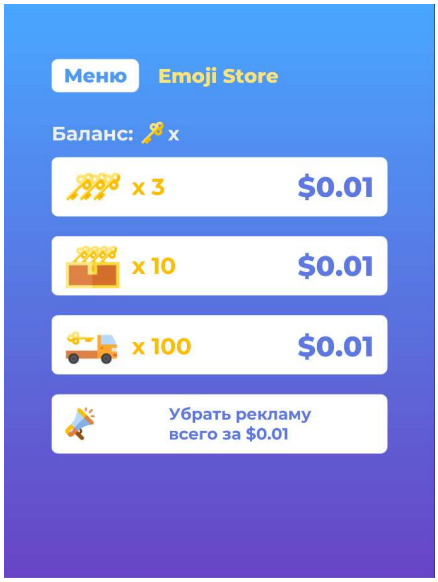
- нарезка и чтение атласов (группа графических ресурсов, сохранённых в виде одного изображения);

- модуль для чтения, принимающий на вход XML-файл и генерирующий по описанию игровые объекты. Модуль был спроектирован для поддержки адаптивности между соотношениями экранов (9:16, 3:4, 9.5:19) и интерполяции промежуточных соотношений. Множество соотношений не является заданным, оно расширяется или уменьшается под требования конкретного проекта [8, 9]. Результат работы модуля для вёрстки в трёх соотношениях можно увидеть на рис. 1 (экран магазина встроенных покупок в проекте Emoji Quest[9]).

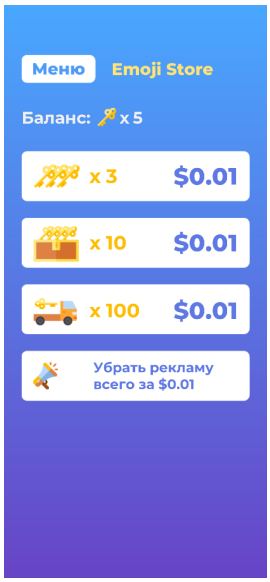
Описанная выше система полностью строилась на компонентной системе Unity3D и с использованием средств для разработки пользовательского интерфейса в Unity3D. Unity3D также является довольно большим движком, как Unreal Engine, к тому же с языком реализации игровой логики C#. Такой набор технологий приводит к большому объёму межъязыкового взаимодействия. Движок не предоставляется

[9]Страница проекта Emoji Quest в Google Play. URL: https://play.google.com/store/apps/details?id=com.novatikgame.emojiquest

9:16



3:4



9.5:19

Рис. 1. Вёрстка экранов в разных соотношениях

с исходным кодом, и решение проблем оптимизации на уровне движка и доработка под специфичные нужды невозможны. Большой про-

блемой являются затраты времени: сборка исполняемых файлов для мобильных платформ, даже для небольших проектов, таких как Quiz Challenge, занимает минимум десять минут.

Язык вёрстки бы сделан похожим на HTML, для упрощения подготовки специалистов к работе с ним. На текущий момент поддерживаются следующие теги: `Row, Col, Layer, Image, Text, Button`. Пример исходного файла для вёрстки всплывающего окна в проекте Emoji Quest приведён ниже.

```
1 <Form name="GameDescription" width="1080" height="1920">
2   <Layer>
3     <Row height="1920">
4       <Image color="0/0/0/128"/>
5     </Row>
6   </Layer>
7
8   <Layer>
9     <Row height="672.5"/>
10
11    <Row height="575">
12      <Col width="78"/>
13      <Col width="924">
14        <Image sprite="elems/1" useNativeSize="false"
              image-type="Sliced">
15          <Row height="80"/>
16
17          <Row height="230">
18            <Col width="89"/>
```

```
19
20              <Col width="751">
21                <Text name="txtGameDescription" text-
                     aligment="MiddleCenter" font="
                     Montserrat-Bold" best-fit="true" font-
                     max="50" color="69/68/120/255"/>
22              </Col>
23            </Row>
24
25            <Row height="169">
26              <Col width="89"/>
27              <Col width="751">
28                <Button name="btnStartGame" onClick="
                     OnClickStartGame" sprite="buttons/1"
                     useNativeSize="false">
29                  <Row>
30                    <Col width="250"/>
31                    <Col width="251">
32                      <Row height="17"/>
33                      <Row height="135">
34                        <Col>
35                          <Text name="txtStartGame" text-
                               aligment="MiddleRight" font="
                               Montserrat-Bold" text="1"
                               best-fit="true" font-max="100
                               " color="255/255/255/255"/>
36                        </Col>
```

```
37                          </Row>
38                        </Col>
39                      </Row>
40                    </Button>
41                  </Col>
42                </Row>
43
44              </Image>
45            </Col>
46          </Row>
47        </Layer>
48 </Form>
```

4. Вторая версия ядра движка Sad Lion Engine и Xamarin

Работа с Unity3D выявила следующие проблемы движка: частые «падения» среды разработки, отсутствие многих методов, стандартных для операционных систем, сложности с отладкой кода, который работает только на реальном устройстве. Эти проблемы обозначили необходимость реализации собственного ядра для игрового движка.

C++ на тот момент казался излишним, большая сложность разработки и высокая вероятность утечек памяти из-за ошибок, необходимость вести два полностью раздельных проектов в Android Studio и XCode привели к выбору C# + Xamarin [10] для Visual Studio. Такой подход позволял избежать использования двух сред разработки и держать всю кодовую базу в рамках C# в отличие от варианта с C++, где

для Android возникает использование Java для инициализации приложения и Objective-C для iOS.

Также вторая версия движка получила свою компонентную систему [11]. Получение конкретного компонента объекта и создание компонент было сделано с помощью шаблонов и рефлексии [4]. Реализация с помощью таких средств получалась довольно короткой по объёму кода.

На тот момент для реализации рендерного цикла была выбрана библиотека для векторной графики Skia[10], созданная Google. Библиотека полностью удовлетворяет потребности 2D игр: загрузка графических ресурсов, векторная графика, различные визуальные эффекты, например градиент, рисование текста, сглаживание. Как стало ясно позже, количество межъязыковых взаимодействий при таком подходе оказалось слишком большим, и приложения довольно часто «падали» без явных следов и сообщений об ошибках.

5. Третья версия ядра движка Sad Lion Engine

Третья и текущая версия ядра движка написана полностью на языке C++ с использованием современных средств управления памятью, потоками и синхронизацией потоков, таких как shared_ptr, unique_lock и т. д. [12; 13].

Изменение подхода к разработке в сторону низкоуровневого программирования позволило увеличить скорость работы приложений примерно на 30 % по сравнению со второй версией. Также концепция shared_ptr, в отличие от сборщика мусора, высвобождает память сразу же, как только на объект больше не осталось ссылок, и не нагружает

[10]Сайт Skia. URL: https://skia.org/

процессор процессом очистки.

В отличие от второй версии, где основным языком был C#, а к библиотеке рендеринга на каждом кадре происходили обращения через межъязыковое взаимодействие, в третьей версии рендеринг и логика написаны полностью на C++ и все ошибки и системные сигналы (SIGSEGV, SIGILL, SIGABRT и т. д.) отслеживаются, что позволяет пусть и с некоторыми сложностями, но полностью отслеживать «падения» [14]. Реализация создания компонент вместо рефлексии была заменена некоторым подобием паттерна «инъекция зависимостей» [15]. Переход на C++ также улучшил качество бинарных файлов и скорость их начальной загрузки.

Конечная версия компонентной системы имеет основной объект `SLObject`, который хранит в себе ссылки на компоненты с логикой. Базовый класс для компонентов `SLCompoent` наследуют все компоненты. По-умолчанию движок предоставляет пользователю три готовых компонента: `SLButton` для кнопок, `SLImage` для изображений, `SLText` для текста. Также есть ещё наследник класса компонентов — класс форм `Form`, этот класс наследуют компоненты-формы.

В третьей версии представлены следующие системы:

1. `XMLRenderer` — система для чтения XML-файлов и генерации деревьев объектов для форм. Автоматически создаёт компоненты, пользуясь инъекцией зависимостей. Расчитывает высоту и ширину объектов в процентах или пикселях.

2. `EngineContext` — основная система ядра движка. Предоставляет инициализацию и деинициализацию графики, доступ к ресурсам,

логированию и инициализирует часть других систем.

3. `PNGLoader` — система для загрузки атласов.

4. `KeyboardManager` — система для работы с клавиатурой.

5. `App` — система, реализующая цикл выполнения приложения.

6. Заключение

В результате работы было реализовано ядро игрового движка на языке C++. На данный момент разработка с ядром имеет довольно высокую сложность из-за особенностей языка. В дальнейшем планируется разработка инструментария для снижения сложности разработки. В последующих версиях запланирована реализация средств для предпросмотра вёрстки без компиляции и запуска приложения на реальном устройстве. Планируется замена Skia собственным ядром рендеринга с тесселятором для векторной графики, для возможности оптимизации конкретных сборок движка под нужды проектов.

Список литературы

1. **Nystrom R.** Game programming patterns. San Bernardino: Genever Benning, 2018. 345 p.

2. **Gregory J.** Game engine architecture, 3rd edition. Boca Raton: CRC Press, 2019. P. 1200.

3. **Иерузалимски Р.** Программирование на языке Lua. 3-е изд. М.: ДМК Пресс, 2016. 382 с.

4. **Nagel C.** Professional C# 6 and .NET Core 1.0. Indianopolis: Wrox, 2016. P. 1464.

5. **Шрейер Д.** Кровь, пот и пиксели. Обратная сторона индустрии видеоигр. М.: Бомбора, 2019. 363 с.

6. **Clayton J.** Metal programming guide. Addison-Wesley. 2018. 352 p.

7. **Sellers G.** Vulkan Programming Guide. The Official Guide to Learning Vulkan, Boston, Columbus, Indianapolis, New York, San Francisco, Amsterdam, Cape Town Dubai, London, Madrid, Milan, Munich, Paris, Montreal, Toronto, Delhi, Mexico City S?o Paulo, Sydney, Hong Kong, Seoul, Singapore, Taipei, Tokyo: Addison-Wesley, 2017. 480 p.

8. **Berg M., Duffy S., Moakley B., Van de Kerckhove E., Uccello A.** Unity games by tutorials. Razeware LLC. 2017. 634 p.

9. **Хокинг Д.** Unity в действии. М.; СПб.; Нижний новгород; Воронеж; Киев; Екатеринбург; Самара; Минск: Питер, 2018. 334 с.

10. **Petzold C.** Cross-platform C# programming for iOS, Android and Windows. Redmond, Washington: Microsoft Press. 1161 p.

11. **West M.** Evolve your hierarchy [Электронный ресурс]. Intel Game Dev. URL: https://software.intel.com/en-us/articles/parallel-techniques-in-modeling-particle-systems-using-vulkan-api (дата обращения: 16.11.2019).

12. **Stroustrup B.** The C++ programming languege 4th edition. Uppder Saddle River, Boston, Indianopolis, San Francisco, New York, Toronto,

Montral, London, Munich, Paris, Madrid, Capetown, Sydney, Tokyo, Singapore, Mexico city: Addison-Wesley. 2013. 1345 p.

13. **О'Двайр А.** Осваиваем C++17 STL. М.: ДМК, 2019. 351 с.

14. **Раго С. А., Стивенс Р. У.** Профессиональное программирование UNIX. М.; СПб.: Символ, 2014. 1100 с.

15. **Seeman M.** Dependency Infection in .NET. Shelter Island, New York: Manning Publications. 2011. 584 p.

Summary

Melnikov V. A. Development Process of game engine core for 2D games and interfaces Sad Lion Engine

Game engine core, written in C++, was released as a result of work. Currently, working with this core is quite difficult because of language specific. Further development of tools for reducing difficulty is planned. In next versions realization of tools for previewing of page layoputs also is planned. Also in future changing Skia with custom rendering core with tessalator for vector graphics is planned< due to needs of optimizations of concrete projects.

Keywords: C++, game development, game engines, interfaces, game engine architecture.

References

1. **Nystrom R.** *Game programming patterns*, San Bernardino: Genever Benning, 2018, 345 p.

2. **Gregory J.** *Game engine architecture*, 3rd edition, Boca Raton: CRC Press, 2019, p. 1200.

3. **Ierusalimschy R.** *Programmirovanie na yazyke Lua*, 3-e izdanie, (Programming in Lua. 3rd ed.), M.: DMK Press, 2016, 382 p.

4. **Nagel C.** *Professional C# 6 and .NET Core 1.0*, Indianopolis: Wrox, 2016, p. 1464.

5. **Schreier J.** *Krov, pot i pikseli. Obratnaya storona industrii videoigr* (Blood, Sweat, and Pixels: The Triumphant, Turbulent Stories Behind How Video Games Are Made), M.: Bombore, 2019, 363 p.

6. **Clayton J.** *Metal programming guide*, Addison-Wesley, 2018, 352 p.

7. **Sellers G.** *Vulkan Programming Guide. The Official Guide to Learning*, Vulkan, Boston, Columbus, Indianapolis, New York, San Francisco, Amsterdam, Cape Town Dubai, London, Madrid, Milan, Munich, Paris, Montreal, Toronto, Delhi, Mexico City S?o Paulo, Sydney, Hong Kong, Seoul, Singapore, Taipei, Tokyo: Addison-Wesley, 2017, 480 p.

8. **Berg M., Duffy S., Moakley B., Van de Kerckhove E., Uccello A.** *Unity games by tutorials*, Razeware LLC, 2017, 634 p.

9. **Hocking J.** *Unity in action*, M., SPb., Nizhnij novgorod, Voronezh, Kiev, Ekaterinburg, Samara, Minsk: Piter, 2018, 334 p.

10. **Petzold C.** *Cross-platform C# programming for iOS, Android and Windows*, Redmond, Washington: Microsoft Press, 1161 p.

11. **West M.** Evolve your hierarchy, Intel Game Dev, URL: https://software.intel.com/en-us/articles/parallel-techniques-in-modeling-particle-systems-using-vulkan-api (date of the application: 16.11.2019).

12. **Stroustrup B.** *The C++ programming languege*, 4th edition, Uppder Saddle River, Boston, Indianopolis, San Francisco, New York, Toronto, Montral, London, Munich, Paris, Madrid, Capetown, Sydney, Tokyo, Singapore, Mexico city: Addison-Wesley, 2013, 1345 p.

13. **O'Dwyer A.** *Osvaivaem C++17 STL* (Mastering the C++17 STL), M.: DMK, 2019, 351 p.

14. **Rago S. A., Stevens R. W.** *Professional'noe programmirovanie UNIX* (Advanced Programming in the UNIX Environment), M., SPb.: Simvol, 2014, 1100 p.

15. **Seeman M.** *Dependency Infection in .NET*, Shelter Island, New York: Manning Publication,. 2011, 584 p.

Для цитирования: Мельников В. А. Процесс разработки движка для 2D игр и интерфейсов Sad Lion Engine // *Вестник Сыктывкарского университета. Сер. 1: Математика. Механика. Информатика. 2019. Вып. 4 (33). С. 21–37.*

For citation: Melnikov V. A. Development Process of game engine core for 2D games and interfaces Sad Lion Engine, *Bulletin of Syktyvkar University. Series 1: Mathematics. Mechanics. Informatics*, 2019, 4 (33), pp. 21–37.

СГУ им. Питирима Сорокина *Поступила 05.12.2019*