Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 21.Б07-мм

Разработка rogue-подобной игры на OCaml

Артеменко Софья Алексеевна

Отчёт по учебной практике
в форме «Решение»

Научный руководитель:
ассистент кафедры системного программирования Д. С. Косарев

Санкт-Петербург
2023

Оглавление

Введение **3**

1. Постановка задачи **4**

2. Обзор **5**

3. Инструменты **6**

3.1. OCaml 6

3.2. Js_of_ocaml 6

3.3. Phaser 7

4. Реализация **8**

Заключение **16**

Список литературы **17**

Введение

Написание относительно простых игр является особенно актуальным для начинающих. Существует множество материалов, представляющих собой подробные пошаговые уроки.

Web-игры требуют от пользователя минимум усилий для запуска, что значительно расширяет аудиторию тех, кто захочет их попробовать. Также браузерные игры потенциально можно запускать на множестве разных устройств, а обновление версии происходит проще и не занимает много времени.

Так, для разработки браузерной 2D-игры естественно использовать JavaScript, так как этот язык широко распространен в данной области, и какой-нибудь движок для быстрого старта, например, Phaser, потому что он бесплатный, не требует особых инструментов для работы. В интернете можно найти обучающие книги по JavaScript в контексте Phaser [2], [3].

Однако, в JavaScript есть свои проблемы (например, типизация) [7], но не все типизированные языки, в свою очередь, хорошо запускаются в браузере (Java, Haskell). Некоторые языки обладают строгой статической типизацией и компилируются в JavaScript, например, OCaml в сочетании с Js_of_ocaml или ReasonML.

В этой работе мы применим OCaml и Js_of_ocaml для создания игры на движке Phaser.

1. Постановка задачи

Целью работы является составление библиотеки для разработки игры на движке Phaser и представление прототипа игры, по аналогии с той, что описана в книге для JavaScript.

Для её выполнения были поставлены следующие задачи:

- Осенний семестр: выбрать книгу и частично переписать по ней игру в синтаксис OCaml с помощью Js_of_ocaml.
- Весенний семестр:
    1. Полностью переписать игру и выделить все необходимое в библиотеку с биндингами к Phaser.
    2. Переписать игру ближе к стилю, свойственному функциональным языкам, а не JavaScript.

2. Обзор

Библиотек OCaml с биндингами к Phaser[1] как таковых в интернете найти не удалось.

В Git-репозитории[2] и на сайте OCamlverse[3] есть подборки с примерами созданных игр, а также ссылки на биндинги к библиотекам движков SDL (преимущественно), orx, raylib, OpenGL, Irrlicht и некоторых других. Данные движки используются вкупе с языками C и C++, следовательно, подразумевается компиляция в машинный код. Более того, C и C++ не подходят для web–разработки.

Также представлены игры, компилирующиеся в JavaScript, например, эмулятор CAMLBOY[4], но в нем использован только HTML5.

[1] https://photonstorm.github.io/phaser3-docs/index.html (Дата доступа: 2023–05–15)
[2] https://github.com/fccm/awesome-gamedev-ocaml (Дата доступа: 2023–05–15)
[3] https://ocamlverse.net/content/games.html (Дата доступа: 2023–05–15)
[4] https://github.com/linoscope/CAMLBOY/ (Дата доступа: 2023–05-15)

3. Инструменты

3.1. OCaml

OCaml [4] — язык функционального программирования, поддерживающий как функциональную, так и императивную и объектно-ориентированную парадигмы программирования. OCaml обладает рядом существенных преимуществ перед другими языками:

- Статистическая проверка типов;
- Параметрический полиморфизм;
- Функции первого класса;
- Сопоставление с образцом.

Также немаловажную роль в данном проекте имеет возможность OCaml компилироваться в JavaScript, так как именно это позволит создать web–игру.

3.2. Js_of_ocaml

Js_of_ocaml [8] — компилятор из OCaml в JavaScript, позволяющий запускать программы на OCaml в средах, типичных для JavaScript (браузер и Node.js). Иногда бывает полезно или необходимо вызывать JavaScript функции из OCaml. Суть в том, чтобы представлять объекты JavaScript в качестве абстрактных типов OCaml и оперировать ими, используя OCaml функции или образы JavaScript функций в OCaml.

API браузеров используется ради предоставляемой функциональности, а не ради предлагаемой парадигмы программирования. При проектировании образов функций можно повторять объектно-ориентированный стиль программирования, а можно пытаться получить функциональный API [1]. Также в некоторых случаях для использования API могут понадобиться типонебезопасные конструкции OCaml.

В качестве системы сборки используется Dune — система сборки для проектов OCaml, которая поддерживает компиляцию с помощью Js_of_ocaml.

3.3. Phaser

Phaser [6] — открытый HTML5 игровой фрэймворк, который представляет рендеринг WebGL и Canvas в настольных и мобильных веб–браузерах. Он доступен для изучения (благо, обучающей литературы в сети достаточно) и действительно используется в индустрии (на сайте представлены десятки игр). Phaser не требует использования каких–либо особых инструментов, достаточно загрузить соответствующие библиотеки, а не устанавливать особую платформу, как того требует, например, Unity.

4. Реализация

Выбранная книга состоит из 8 глав, каждая из которых содержит 3-5 примеров, которые идут по возрастающей сложности происходящего: сначала создается пустое подземелье, затем добавляется персонаж пользователя и враги с разной сложностью поведения, на последнем этапе подключается процедурная генерация уровней. В работе была сохранена описанная организация.

Первая глава посвящена общей информации о том, что из себя представляют rogue-подобные, почему браузерные игры это здорово и использование Phaser это оптимальный выбор. Во второй главе создается первый объект, обеспечивается его движение, подключается управление клавиатурой. В третьей главе идет речь об использовании тайловых карт, построении первого подземелья, добавлении персонажа игрока и менеджера ходов.

В общем случае структура кода при переходе на OCaml не меняется, разве что функции и основные объекты из библиотек Phaser нужно объявлять перед использованием, обозначая типы аргументов и возвращаемых значений в соответствии с правилами Js_of_ocaml [5].

Незначительные изменения претерпели следующие части реализации.

Для объекта сначала нужно объявить поле, только потом обращаться к нему в другом методе.

```
let helloText =
    add##bitmapText 400 300 (Js.string ''arcade'')
    (Js.string ''Hello Phaser'')
in
helloText##setOrigin 0.5
```

Тогда как в JavaScript этого не требуется:

```
helloText = add.bitmapText(400, 300, ''arcade'',
''Hello Phaser'').setOrigin(0.5);
```

Кроме того, во время исполнения Phaser отсоединяет пользователь-

ские методы от пользовательских объектов и присоединяет их ко встроенным объектам (Рис. 1). Из-за этого при вызове методов может неожиданно измениться указатель this. Поэтому при создании OCaml объектов необходимо передавать this явно, а именно тот объект, который станет будущим this.

```
class type game_object_creator =
  object
    method tilemap : config_map t → map t meth
  end

let curr_map : map Js.t ref = ref (Js.Unsafe.js_expr ''1'')

let dungeon this =
  object%js (that)
    ...
    method initialize () =
      console##log
      ((Js.Unsafe.eval_string {|obj => Object.keys(obj) |} :
      _ → _) this);
      console##log ''this'';
      console##log
      ((Js.Unsafe.eval_string {|obj => Object.keys(obj) |} :
      _ → _) that);
      console##log ''that'';
      let make : game_object_creator t =
        (Js.Unsafe.eval_string {|x => x.make |} :
         _ → _) this
      in
      ...
      let (map2 : map t) = make##tilemap current_map_config
      in curr_map := map2;
      ...
  end
```

```
let caml_scene =
  object%js (this)
    ...
    method create () =
      ...
      let a = (Dungeon.dungeon this)##initialize () in
      ...
      a
  end
```

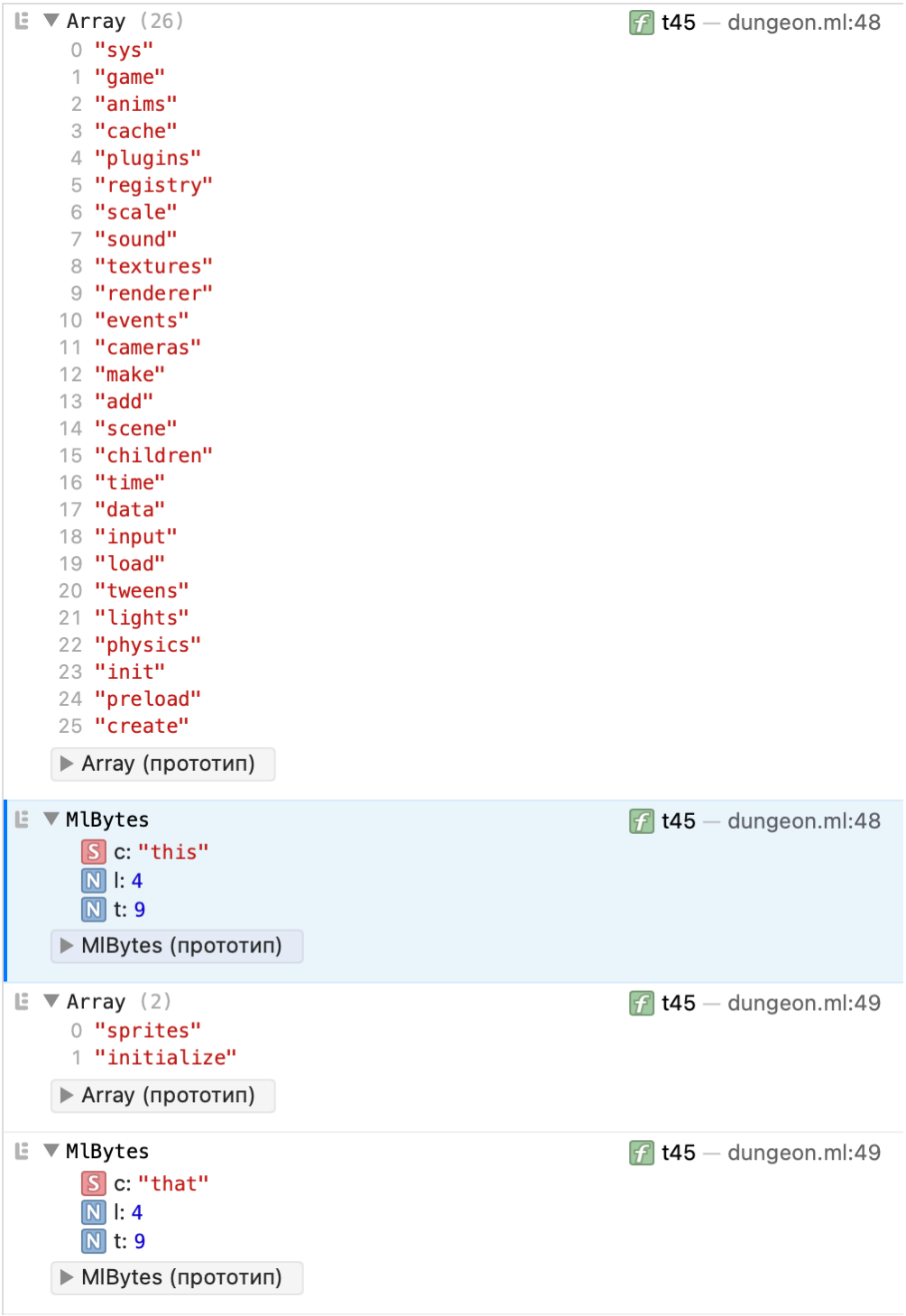


Рис. 1: Для проверки корректности указателей нужно вывести в консоль разработчика соответствующие им методы. Первый, необходимый нам, обладает множеством методов (в том числе ”make”, который помог обнаружить данную ошибку), а второй, созданный OCaml, всего двумя, имеющимися у объекта.

В четвертой главе предлагается провести рефакторинг последнего примера из предыдущей главы, создать первого монстра и осуществить возможность взаимной атаки.

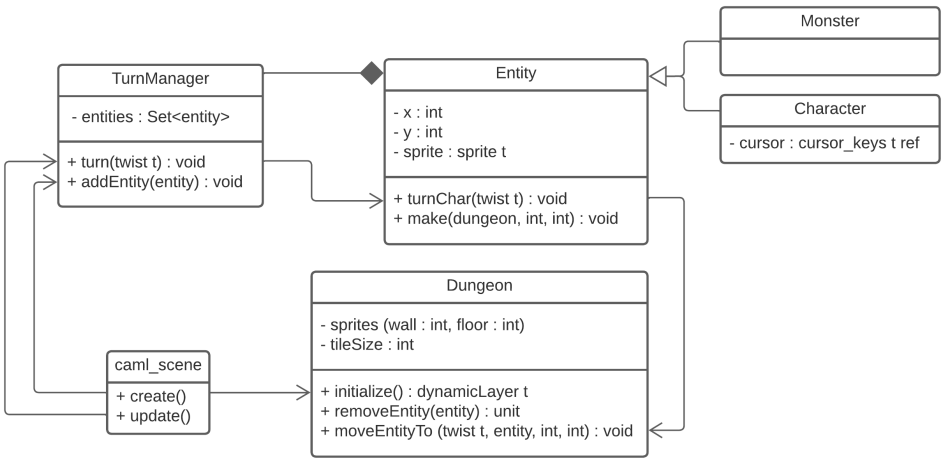


Рис. 2: Часть UML-диаграммы игры, не включающая библиотечных объектов и функций.

Сначала был проведен рефакторинг, что впоследствии упростило работу с сущностями. Изменился способ перемещения игрока, ранее он был лишь плиткой (tile), то есть движение осуществлялось простой заменой последней. Сейчас же сущность становится спрайтом (sprite), так как мы хотим использовать Tweens: процесс в анимации, который заключается в создании промежуточных кадров для более плавного изображения. Более того, так сущность может передвигаться без привязки к чему-либо, как бы поверх. В связи с этим нужно было лишь добавить новое поле sprite заранее описанного типа sprite t, переделать инициализацию игрока путем добавления спрайта на карту с помощью game_object_creator.

Следует уделить большее внимание Tweens. Tweens является видом объектов класса tweens_manager, для создания которого необходима конфигурация. Одним из полей является функция onComplete, которую мы сами должны описать. Здесь сложность заключается в том, что Tweens прикрепляется к dungeon с помощью метода add, для этого необходимо передавать ссылку на dungeon, который уже потерян вследствие перехода по функциям turn → turnChar → moveEntityTo, поэтому

необходимо "протянуть" ссылку через все эти методы (Рис. 2).

```
class type twist = object end

let caml_scene =
  object%js (this)
    ...
    method update () =
      if TurnManager.tm##over () == true then
        TurnManager.tm##refresh ();
      TurnManager.tm##turn (this :> twist t)
  end

let tm =
  object%js (this)
  ...
  method turn : twist Js.t → unit = fun twist →
      ...
      SS.iter (fun x → if not (x#overChar ())
                then x#turnChar twist) this##.entities;
      ...
end

class character =
  object (self : 'self)
    ...
    (Dungeon.dungeon twist)##moveEntityTo
      twist self stuff##.newX stuff##.newY;
    ...
  end
```

Далее, в игре появляется первый монстр, которого тоже нужно добавлять в список, который, в свою очередь, нужен для менеджера ходов. Так, было необходимо добавить базовый класс сущности, который

никак не был описан в JavaScript версии игры, классы игрока и монстров стали дочерними (Рис. 2). Для того, чтобы монстр мог преследовать персонажа игрока (постоянно двигаться по направлению к клетке, на которой он находится), используется алгоритм поиска пути. Соответствующая библиотека подключается одной строчкой в файле html, классы с полями и методами следует описывать заранее, то есть точно так же, как и с Phaser.

Следующая проблема заключается в том, что для того, чтобы двигаться по направлению к игроку, небходимо иметь доступ к нему. В JavaScript можно просто обратиться к игроку через подземелье, у нас же он никак не закреплен, поэтому пришлось отдельно выделить персонажа игрока в глобальную переменную модуля Dungeon, создав ссылку, проблем с этим не возникает, так как он единственный. Однако, если сделать ссылку на объекта модуля Player, получится циклическая зависимость, поэтому подходит только Entity, что, в целом, ни на что не влияет.

Для того, чтобы сущности не могли находиться вместе на одной клетке, написана функция entityAtTile, которая принимает координаты, и, перебирая все сущности, проверяет, есть ли на интересующей клетке какой-либо персонаж, и возращает либо false, либо entity, что невозможно в OCaml. Для этих целей подходит использование типа Option, который может быть или None (undefined), или Some (любым самостоятельно заданным типом). В JavaScript происходит итерация по for, в OCaml-версии игры так сделать не получится, так как из списка элемент по номеру взять невозможно, поэтому была попытка использовать итератор с флагом, однако не unit-функция, каким является присвоение флагу другого значения, не может быть в итераторе. Так, был применен fold с аккумулятором типа entity.

В последнем выполненном примере после того, как очки здоровья (healthPoints) монстра стали отрицательными, необоходимо удалить его из списка менеджера ходов и соответствующий спрайт с карты. Данное действие должно произойти по окончании анимации. Для этого в Tweens есть ранее упомянутое поле onComplete, которое является функ-

цией. В предыдущем примере, где удалять сущность не требовалось, функция onComplete была описана как метод, при этом все исполнялось корректно. Однако, при добавлении Tweens с исполнением внешней функции удаления игрока из onComplete игра прекращала работу. Была допущена ошибка в определении onComplete как метода, так как на самом деле эта функция является полем.

Заключение

Была выбрана книга Roguelike Development with Javascript Алвеса Гарсии и частично создан прототип игры[5] по ней (первая и вторая, третья главы из восьми), а также все использованные на данный момент функции и типы Phaser были выделены в библиотеку.

[5] https://github.com/chilison/jsoo_phaser_template.git (Дата доступа: 29.05.23)

Список литературы

[1] Daniel Bünzli. A JavaScript FFI for idiomatic OCaml programming. — 2022. — URL: https://erratique.ch/software/brr/doc/ffi_manual (дата обращения: 2022-12-15).

[2] Garzia Andre Alves. Roguelike Development with JavaScript. — Apress, 2020. — URL: https://doi.org/10.1007/978-1-4842-6059-3.

[3] Lodriguez Oscar. Let's Build a Multiplayer Phaser Game. — Apress, 2019. — URL: https://doi.org/10.1007/978-1-4842-4249-0.

[4] Madhavapeddy Anil, Minsky Yaron. Real World OCaml: Functional Programming for the Masses. — 2 edition. — Cambridge University Press, 2022.

[5] Ocsigen. Js_of_ocaml - Reference Manual. — 2022. — URL: https://erratique.ch/software/brr/doc/ffi_manual (дата обращения: 2022-12-15).

[6] Photon Storm. Phaser - HTML5 Game Framework. — 2021. — URL: https://photonstorm.github.io/phaser3-docs/index.html (дата обращения: 2022-12-15).

[7] Simpson Kyle. You don't know JS - types & grammar. — Sebastopol, CA : O'Reilly Media, 2014. — Aug..

[8] Vouillon Jérôme, Balat Vincent. From bytecode to JavaScript: the Js_of_ocaml compiler // Software: Practice and Experience. — 2014. — Vol. 44, no. 8. — P. 951–972. — https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.2187.