Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»

Кафедра математической
кибернетики и компьютерных наук

РАЗРАБОТКА ГРАФИЧЕСКОГО ИНТЕРФЕЙСА ПРИЛОЖЕНИЯ НА ДИНАМИЧЕСКОМ ЯЗЫКЕ ПРОГРАММИРОВАНИЯ RUBY

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 4 курса 451 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Шаткова Вадима Дмитриевича

Научный руководитель
к. ф.-м. н. 05.06.2017 С. В. Миронов

Заведующий кафедрой
к. ф.-м. н. 05.06.2017 С. В. Миронов

Саратов 2017

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Ruby — это динамический язык программирования с открытым исходным кодом [1] с упором на простоту и продуктивность. Он обладает элегантным синтаксисом, который приятно читать и легко писать.

Именно из-за простоты этого языка, в большинстве случаев, свои первые игры программисты и любители предпочитают писать на нем. Поклонники Ruby называют его красивым, искусным языком. И в то же время они говорят, что он удобный и практичный.

Итак, останавливаясь на теме разработки компьютерных игр, следует отметить, что для этого языка были специально написаны игровые библиотеки, облегчающие работу программистов и любителей при написании (разработке) компьютерных игр. Существуют следующие библиотеки: Gosu, Chingu, Gamebox, ray, Chipmunk, Rubygame, metro и другие [2].

Без сомнений, лидером является библиотека Gosu. Она известна своей простотой в использовании и производительности, являясь одной из предпочтительных библиотек для разработки 2D игр.

Целью данной работы является освоение механизма построения графического интерфейса для заранее разработанного программного продукта. В нашем случае — это игра «Морской бой», написанная на языке программирования Ruby с выводом всей информации в консоль и полностью отсутствующей графической частью.

Общая постановка задачи: реализовать графический интерфейс для приложения «Морской бой» на языке Ruby с использованием вспомогательных библиотек, а также доработать и улучшить искусственный интеллект в игре. Для решения поставленной задачи необходимо:

1. исследовать возможности библиотеки Gosu;
2. исследовать задачи, связанные с реализацией графического интерфейса для приложения «Морской бой»;
3. разработать набор предложений по реализации графического интерфейса и провести его реализацию;
4. разработать набор предложений по улучшению функционала приложения и внести изменения.

1 Реализация графического интерфейса с помощью Gosu

В данном разделе мы рассмотрим возможные способы и средства (инструменты) для реализации заданного графического приложения.

В качестве исходного приложения была дана система «Морской бой», являющаяся решением одной из задач курса «Языки программирования» студентов направлений «Программная инженерия» и «Фундаментальная информатика и информационные технологии». Система представляет из себя консольное приложение. Общение пользователя с системой происходит посредством текстового диалога.

Приложение представляет из себя модифицированный вариант классической игры «Морской бой».

В рамках дисциплины «Языки программирования» студентам предлагается описать собственную стратегию игры «Морской бой» и запрограммировать ее в качестве алгоритма поведения бота (запрограммировать «интеллект» компьютера). После завершения студентами выполнения этого задания, на одном из занятий проводится «соревнование» стратегий.

Идея для настоящей работы возникла как желание повысить интерес к выполнению задания посредством повышения зрелищности итогового соревнования.

1.1 Структура исходного приложения

Описание исходного приложения состоит из 5-ти классов:

— классы `Field` и `BattleField` описывают игровое поле;
— класс `Ship` описывает корабль;
— класс `Player` описывает игрока с его стратегиями;
— класс `Game` контролирует ход игры для заданных игроков.

Рассмотрим все классы подробнее:

Объект класса `Field` содержит лишь информацию о занятом и свободном пространстве на игровом поле. Его методы позволяют разместить ссылку на объект в клетках игрового поля, запросить информацию о свободном пространстве;

Каждый представитель класса `Ship` описывает корабль, состоящий из заданного количества клеток на заданном игровом поле. Корабль имеет информацию о своем здоровье и расположении на игровом поле (если он уже

помещен на поле). Методы корабля позволяют получить информацию о нем и изменить некоторые его свойства, включая его расположение на игровом поле;

Представители класса `BattleField` представляют собой игровое поле с кораблями объектами класса `Ship`. Объект класса `BattleField` кроме информации об игровом поле имеет доступ к информации о кораблях. Такой объект основной объект отвечающий за состояние поля в игре. Объект `BattleField` взаимодействует со связанными с ним кораблями: он может напрямую изменять их свойства, так же как и корабли, при желании, могут напрямую изменять свойства относящегося к ним объекта `BattleField`;

Представитель класса `Player` отвечает за поведение игрока в ходе игры. Основные методы класса представляют собой стратегию действия игрока: стратегию размещения кораблей на поле перед началом игры, стратегию хода в части перемещения кораблей на игровом поле, стратегию выбора координат для очередного выстрела. Объект класса может накапливать статистику о ходе игры, которую он может использовать при выборе стратегии. Объекты класса `Player` не могут вносить изменения в игровое поле напрямую, а лишь могут отвечать на запросы извне.

Объект класса `Game` является связующим звеном между участниками игры и следит за ходом ее выполнения. Объект `Game` отправляет запросы игрокам и игровым полям и перенаправляет ответы на них.

1.2 Инструменты реализации графического приложения

Будем реализовывать графический интерфейс для программы с помощью ниже перечисленных средств.

Библиотека Gosu — вспомогательная библиотека с открытым исходным кодом для разработки 2D игр. Именно с помощью этой библиотеки будет разрабатываться графическая часть нашей игры.

Ocra (приложение One-Click Ruby) — создает исполняемые файлы Windows из исходного кода Ruby. Исполняемый файл представляет собой самораспаковывающийся исполняемый файл, содержащий интерпретатор Ruby, исходный код и любые дополнительные библиотеки ruby или DLL.

Gosu является библиотекой с открытым исходным кодом для разработки 2D игр, для языков программирования Ruby и C++. Разработали библиотеку Julian Raschke и Jan Lucker [3]. Библиотека Gosu известна своей простотой в использовании и производительности, являясь одной из предпочтительных

библиотек для разработки компьютерных игры на языках программирования Ruby и C++.

Кроме того Gosu включает в себя демо для интеграции с RMagick, Chipmunk и OpenGL.

Библиотека предлагает легкий объектно-ориентированный интерфейс для доступа к общим ресурсам:

— аппаратное ускорение 2D-графики и текста;
— аудио файлы в различных форматах;
— ввод данных с клавиатуры, использование мыши и геймпада.

Библиотека предлагает обширную дополнительную функциональность, включая сети, продвинутые операции при работе с цветами, математические функции, региональные операции и многое другое.

Gosu нацелена на то, чтобы содержать все, что нужно для написания игры на языках программирования C++ или Ruby, при этом позволяя пользователю забыть о низком уровне и шаблонном коде.

Для реализации графического приложения нам понадобится использовать следующие встроенные классы библиотеки: [5]

— `Gosu::Window`. Основной класс, который служит основой стандартного приложения Gosu. Управляет инициализацией всех основных компонентов Gosu и обеспечивает временную функциональность;
— `Gosu::Font`. Будет использоваться для рисования текста в объекте Window. Шрифты идеально подходят для небольших текстов, которые меняются регулярно.
— `Gosu::Image`. Обеспечивает функциональность рисования статических изображений;
— `Gosu::Sample`. `Sample` - это короткий звук, который полностью загружен в память, может воспроизводиться несколько раз подряд и предлагает очень гибкие параметры воспроизведения. Не для воспроизведения музыки;
— Gosu::Song. Для воспроизведения музыки. Поддерживает аудио форматы WAV и OGG. Второй использовать будет разумнее, так как файлы будут весить в десятки раз меньше, если бы мы использовали формат WAV;
— `Gosu::Color`. Для определения цветов и дальнейшего использования при рисовании текста.

1.3 Исследование задач, связанных с реализацией графического интерфейса приложения

Чтобы реализовать графический интерфейс для приложения, нам необходимо будет исследовать следующие задачи:

- создание и интеграция графической части с исходным программным продуктом, без внесения изменений в исходный код программы;
- оптимизация и решение проблем по ходу разработки;
- тестирование на этапе разработки графического приложения;
- внесение изменений и нововведений.

Чтобы разработка графического интерфейса была максимально качественной и удобной, можно выделить следующий набор предложений по реализации графического интерфейса:

- разбиение исходного файла программы на библиотеки, которые будем подключать в основном файле игры, через который будет проходить запуск;
- создание дополнительных библиотек/файлов с набором новых или переопределенных функций, чтобы не вносить изменения в исходный код исходного продукта;
- создание файлов с подключением ресурсов для графического интерфейса, определения констант, рисования графической части приложения;
- использование скриншотов сторонних интерфейсов игр в качестве референса для разработки собственного.

2 Реализация графического приложения «Морской бой»

Данный раздел представляет собой практическую часть работы. Здесь описаны основные классы и методы которые мы используем для реализации графической части приложения, доработки и улучшения искусственного интеллекта в игре.

2.1 Разработка графической составляющей

Для начала стоит определить, что вообще мы собираемся выводить на экран и из каких частей будет состоять интерфейс. Можно выделить три части:

Меню игры:

- логотип;
- задний фон;
- кнопки меню.

Во время игры:

- логотип;
- задний фон, с наложенными поверх сетками игрового поля для игроков;
- блоки с текстом состояния игры (над обоими полями);
- никнеймы/имена игроков и информация о количестве кораблей в данный момент времени;
- изображения кораблей и информация о здоровье;
- иконки попадания/промахов/уничтожения/выбора клетки;
- блок/окно паузы с кнопками;
- блок/окно действий с кнопками;
- текст подсказки;
- выделение проигравшего игрока.

Конец игры:

- текст статуса (кто победил);
- блок/окно действий с кнопками.

За референс возьмем интерфейс игры «Battleship: Facebook Game» [4], которая была разработана специально перед премьерой фильма «Морской бой» 2012 года. После выхода фильма игра была удалена, никаких ресурсов от нее, кроме пары скриншотов не осталось. Именно один из таких скриншотов станет основой для интерфейса нашей игры.

Опуская мелкие детали, вроде изображений попаданий/промахов, возь-

мемся за изображения кораблей. Будут использованы исключительно сторонние ресурсы, так как на рисование кораблей с нуля уйдет не мало времени, к тому же нужен хороший опыт в дизайне.

2.2 Иерархия классов графического приложения

Исходный код игры дан одним файлом. Для удобства создадим папку libs в папке с программой, в которую поместим файлы со всеми классами исходного приложения, кроме `Game`. Все файлы назовем также, как называются классы, описанные в этих файлах. В файле же с основным классом `Game` подключим созданные библиотеки с другими классами, а также библиотеку Gosu. Кроме того, была подключена дополнительная библиотека Unicode для работы с кириллическими символами. Файл `Field.rb` подключен только в файле `BattleField.rb`, так как используется только там.

Для реализации графической части приложения нам понадобятся следующие методы в классе `BattleField`:

— метод `remains` для получения массива данных о расположении кораблей игроков на игровом поле;
— метод `shoot` для получения информации о состоянии выстрела. Причем, этод метод придется переопределить, для внесения некоторых изменений;
— метод `fleet`, который послужит для создания простенького метод получения информации о количестве кораблей для последующего вывода;
— метод `game_over?` для определения победителя и проигравшего.

И в классе `Player`:

— метод `ship_move_strategy` для перемещения/вращения кораблей на игровом поле;
— метод `shot_strategy` для выстрела по полю противника.

2.3 Подключение ресурсов и определение констант

Для определения констант и подключения ресурсов создадим дополнительные файлы с названиями `Constants.rb` и `Resources.rb`. В `Constants.rb` зададим постоянные переменные, которые в дальнейшем пригодятся нам при рисовании элементов интерфейса. А в файле `Resources.rb` создаем класс `ResourceLoader` для удобной загрузки ресурсов и их использования.

2.4 Основной класс реализации графической части приложения

Для реализации графической части, создадим файл `Drawer.rb` и в нем класс `Drawer`. В этом классе определим следующие основные методы:

- `game_menu` — игровое меню с фоном и кнопками;
- `help_menu` — раздел управления с показом кнопок управления в игре;
- `game_paused` — окно паузы и конца игры;
- `players_avatar` — размещение аватаров (картинок) игроков и пометка проигравшего игрока;
- `name_players` — размещение никнеймов (имен) игроков;
- `ships_count` — размещение информации о количестве кораблей;
- `ships_player` — основной метод размещения кораблей на поле с указанием их здоровья;
- `state` — состояние игры (ожидание, попадание, не попал, потопил, выиграл или проиграл). Рисуется как текст, так и изображения попаданий-/промахов на игровом поле;
- `move_player` — управление кораблями. Выбор кораблей, окно действий;
- `shot_player` — размещение иконки выбора клетки на противоположном поле для выстрела.

2.5 Состояние игры и состояние управления кораблями

Для того, чтобы рисовать нужные нам части приложения (меню, игра, пауза и т.д.), необходимо определить переменные.

Создадим глобальную переменную `@game_state`, которая будет принимать следующие значения:

- `:menu` — меню игры;
- `:in_play` — сама игра;
- `:stopped` — пауза;
- `:game_end` — конец игры.

И глобальную переменную `@move_state`, которая необходима в одиночной игре для чередования действий игрока при управлении кораблями. Переменная `@move_state` может принимать следующие значения:

- `:select_ship` — выбор корабля с помощью кнопок, определенных в классе Game;
- `:select_option` — корабль выбран, выбираем действие;

— :option_rotate — корабль выбран, выбрано вращение, выбираем точку, в которой хотим повернуть корабль;
— :option_move — корабль выбран, выбрано перемещение, выбираем куда перемещать корабль (вперед или назад).

2.6 Построение интерфейса и запуск приложения

Теперь вернемся в основной файл приложения Game.rb и подключим все необходимые файлы.

В классе Game в методе инициализации:
— создаем окно с размерами, заданными в файле констант;
— загружаем ресурсы и определяем глобальную константу @drawer, связанную с классом Drawer, для удобного обращения к методам заданного класса;
— заносим в переменную @players массив с информацией об игроках, их игровом поле;
— заносим в переменную @players массив с информацией об игроках, их игровом поле;
— инициализируем глобальные переменные, которые понадобятся нам ниже: @preres, @hitshots, @game_state, @game_over и @gamespeed и другие.

Включаем курсор мыши в методе needs_cursor?.

Рисуем все наши графические элементы в зависимости от состояния игры, передавая необходимые значения в качестве аргументов (например информацию о количестве кораблей или их положении) в методе draw.

Определяем нажатия клавиш в методе button_down:
— Escape — выход/пауза;
— P — пауза/остановка игры;
— + или - — изменение скорости игры (для автоматического режима - игра ботов);
— стрелочки или (W,S,A,D) — основное управление (переключение кнопок, выбор кораблей и т.д.).

В методе reset настраиваем и размещаем корабли игроков.

Создаем метод рестарта/новой игры под названием gamerestart?, методы паузы/остановки игры gamepaused?

Метод botstrategy выбора оптимальной стратегии, в зависимости от

количества и типа кораблей противника и метод, сканирующий количество текущих ходов и в случае завершения стратегии, генерируется новая:

Метод `gamespeed?` изменения скорости игры.

Методы `next_option!`, `prev_option!` и `select_option` переключения кнопок и подтверждения выбора.

Метод `start` служит запуском всего процесса игры. В нем определяем игроков, лечим корабли, перемещаем, в глобальную переменную записываем текущее состояние первого игрока, текущий выстрел. Если идет серия удачных выстрелов, то в глобальную переменную `@hitshots` заносим массив с текущим выстрелом, чтобы рисовать изображения попаданий последовательно, а не по новой. Если серия выстрелов не удачная, то переменную очищаем.

Для успешного запуска, определяем метод `update`.

Так как `update` выполняется при каждом новом запуске приложения раньше метода `draw`, то пропускаем первый запуск метода `start` с помощью условий состояния игры.

Запуск приложения выполняем следующим образом:

```
1 p1 = Player.new("Ivan")
2 p2 = Player.new("Feodor")
3 g = Game.new(p1,p2)
4 g.show
```

3 Внесение изменений в функционал исходного приложения

3.1 Изменение поведения компьютера в игре в случае использования стандартной стратегии

Чтобы игра пользователя с компьютером была интересной, увлекательной и быстро не наскучила, компьютер не должен уступать в игре даже профессиональным игрокам. Можно выделить следующий набор предложений по улучшению искусственного интеллекта в игре:

- подбирать стратегии в зависимости от количества и типа кораблей противника;
- постараться максимизировать попадания по кораблям противника;
- после первого попадания по кораблю проводить целевые обстрелы, а в случае промаха продолжать поиск корабля в ближайших координатах;
- использовать различные варианты оптимальных стратегий в одной игре.

Для достижения перечисленных целей были рассмотрены существующие стратегии поиска и уничтожения кораблей противника [6–9].

Прежде чем переходить к стратегиям нападения, хотелось бы затронуть тему с обстрелами после успешного попадания по кораблю противника. Стратегия выбора координат выстрела реализована в методе `shot_strategy`.

Данный метод не идеален. Вся проблема в том, что после неудачного выстрела координаты очищаются и бот снова стреляет случайно, забывая о том корабле, по которому он попал совсем недавно.

Исправим это, создав глобальную переменную `@bothunter`, значение которой равно двумерному массиву с первым элементом, принимающим последние координаты успешного попадания по кораблю противника и вторым элементом, принимающим число неудачных попыток (промахов).

Как только компьютер попадает по кораблю игрока и делает промах, включается режим поиска корабля противника до тех пор, пока число промахов не превысит значения, равного 10-ти.

Если значение `bothunter[0]` не пустое, то вызывается специально созданный метод `bot_hunter_point`, на который передаем координаты успешного выстрела. Вокруг координат «чертится» поле размером 5 на 5, в котором случайно выбираются координаты и делается выстрел. Игроку будет тяжело вывести корабль за пределы этого поля, особенно, если ему мешают другие корабли.

В случае, если в это поле попадают другие корабли игрока, то велика вероятность, что компьютер, если и не попадет по предыдущему кораблю, то точно заденет новый. После успешного попадания поле «чертится» вокруг новых координат.

Если количество промахов, как сказано было ранее, превышает 10, режим отключается и компьютер возвращается к первоначальной стратегии (случайному выстрелу).

Попробуем рассмотреть и реализовать новые стратегии.

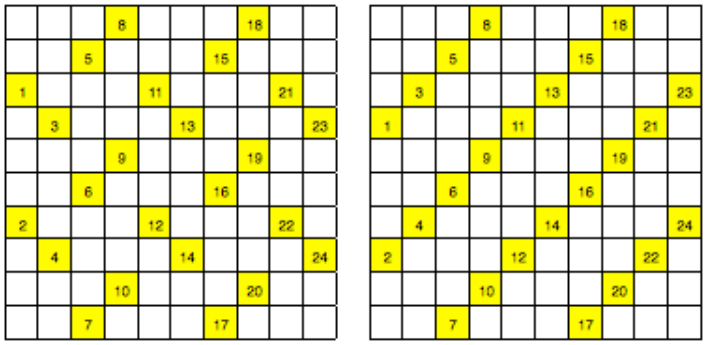

Рисунок 1 – Стратегии: Оптимальная стрельба

Первые две стратегии гарантируют попадание по большим кораблям (от 4 клеток) максимум за 24 выстрела (см. рисунок 1).

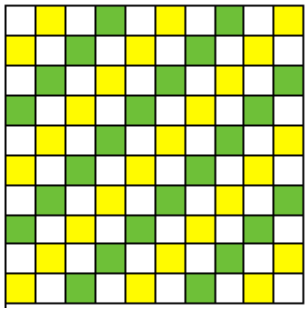

Рисунок 2 – Стратегии: Оптимальная стрельба

Перейдем к стратегии, которая поможет нам находить мелкие корабли, меньше 4 клеток (см. рисунок 2).

Кроме того, было добавлено еще несколько стратегий. Полный список добавленных стратегий:

— `bot_strategy_bigship1` и `bot_strategy_bigship2` — поиск больших кораблей за 24 хода;
— `bot_strategy_smallship` — поиск мелких кораблей за 26 ходов;
— `bot_strategy_center` — обстрел центра и по диагоналям за 20 ходов;

— `bot_strategy_hline` — обстрел игрового поля горизонтальными параллельными линиями за 49 ходов;
— `bot_strategy_vline` — обстрел игрового поля вертикальными параллельными линиями за 49 ходов.

ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы было реализовано графическое приложение «Battleship Future». Были решены следующие задачи:

- исследованы возможности библиотеки Gosu;
- исследованы задачи, связанные с реализацией графического интерфейса для приложения «Морской бой»;
- разработан набор предложений по реализации графического интерфейса и проведена его реализация;
- был доработан и улучшен искусственный интеллект бота, который использует набор заданных стратегий, выбирая оптимальную, в зависимости от количества и типа кораблей противника и проведено тестирование готового продукта.

Разработанное приложение может быть использовано в качестве тренажера при реализации алгоритма собственной стратегии студента в рамках выполнения задания курса «Языки программирования» направлений «Программная инженерия» и «Фундаментальная информатика и информационные технологии». Кроме того, разработанный программный продукт может быть использован на «соревновании» стратегий с целью повышения зрелищности получения результатов таких соревнований.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Язык программирования Ruby [Электронный ресурс]. — URL: https://www.ruby-lang.org/ru/ (дата обращ. 06.05.17). Загл. с экр. яз. рус.

2 Game libraries - The Ruby Toolbox [Электронный ресурс]. — URL: https://www.ruby-toolbox.com/categories/game_libraries (дата обращ. 08.05.17). Загл. с экр. яз. англ.

3 Gosu, 2D game development library [Электронный ресурс]. — URL: https://www.libgosu.org/ (дата обращ. 12.05.17). Загл. с экр. яз. англ.

4 Militaristic Ships Icons Set Navy Ammunition - Shutterstock [Электронный ресурс]. — URL: https://www.shutterstock.com/ru/image-vector/militaristic-ships-icons-set-navy-ammunition-287667191 (дата обращ. 19.05.17). Загл. с экр. яз. англ.

5 Module: Gosu — Documentation for jlnr/gosu (master) [Электронный ресурс]. — URL: http://www.rubydoc.info/github/jlnr/gosu/Gosu (дата обращ. 14.05.17). Загл. с экр. яз. англ.

6 Оптимальный алгоритм игры в морской бой / Хабрахабр [Электронный ресурс]. — URL: https://habrahabr.ru/post/180995/ (дата обращ. 01.06.17). Загл. с экр. яз. рус.

7 3 Ways to Win at Battleship - wikiHow [Электронный ресурс]. — URL: http://www.wikihow.com/Win-at-Battleship (дата обращ. 02.06.17). Загл. с экр. яз. англ.

8 Battleship [Электронный ресурс]. — URL: http://www.datagenetics.com/blog/december32011/ (дата обращ. 02.06.17). Загл. с экр. яз. англ.

9 Battleships решение [Электронный ресурс]. — URL: http://www.conceptispuzzles.com/ru/index.aspx?uri=puzzle/battleships/techniques (дата обращ. 03.06.17). Загл. с экр. яз. рус.

05.06.2017