Оглавление

ВВЕДЕНИЕ

Все любят игры. А те, кто это отрицает, просто стыдятся признаться. Не зря всё-таки за пятьдесят с лишним лет они превратились в многомиллионную индустрию с сотнями тысяч проектов на любой вкус и цвет.

Мы живём в замечательное время – игры сейчас доступны как никогда. А вместе с ними стали доступнее и инструменты разработки. Это, конечно, приносит свои проблемы – к примеру, наводнение Стима тоннами низкокачественных проектов в последнюю пару лет, или кошмарная захламленность Google Play и Appstore. Однако факт есть факт – любой школьник может просто взять и за несколько лет научиться делать вполне себе неплохие игры.

Можно, конечно, взять тот же C++ и резвиться с указателями долгими зимними вечерами, получая запредельные скорости работы. Однако давайте посмотрим на вещи реально – мало кто будет таким заниматься. Всё-таки на первом месте у любителей простота и скорость разработки. Да и C++ быстр подобно лани лишь в умелых руках. Дилетанты вроде меня только загрузят себе процессор на сто процентов и потратят кучу нервов, пытаясь это исправить.

И тут на сцену выходит Java со слоганом “Write once – run everywhere”. Проще и приятнее C++ в использовании, пускай медленнее, зато избавляет от массы головной боли. И, конечно же, нельзя забывать об отличной кроссплатформенности. Всё это делает Джаву довольно привлекательной для разработчика.

1. Обзор литературы

1.1 Спецификации игры

Игра является пошаговой стратегией, рассчитанной на PvP. Генерируется карта, на ней спавнятся две команды. Каждый игрок должен убить всех юнитов оппонента для победы.

1.2 Обзор аналогов

Я не могу назвать ни одного прямого аналога, поскольку и не пытался что-то напрямую клонировать. В прошлом довольно неплохо показал себя подход «возьми по маленькому кусочку от кучи игр, добавь своих идей – и получишь нечто оригинальное». Так что будет уместнее просто перечислить игры, из которых я черпал вдохновение и заимствовал идеи.

HOMM

Сравнение с такой монументальной серией игр, как Герои Меча и Магии (а в особенности третьей её частью) может показаться смешным. Так оно и есть, однако моя любовь к пошаговым стратегиям, как и львиная доля вдохновения, взяла корни именно оттуда.
Если же читатель по какой-то странной причине не знает о серии HOMM – это довольно комплексные игры в жанре пошаговой стратегии. Первая игра была выпущена в далёком 1995-ом, последняя, седьмая часть – в 2015-ом. Однако наибольшую популярность сыскал HOMM III, став, без сомнения, классикой. До сих пор на аналогах почившего GameSpy можно найти множество игроков.

Bitesize Wars

RTS с микрогеймплеем одного моего знакомого. Главная её фишка в том, что матчи длятся не больше нескольких минут, в отличие от Героев, где в оду карту можно играть неделями.

Worms

Пошаговая PvP-артиллерия, тоже в своё время заимевшая звание классики. За один ход можно управлять только одним червём, время ограничено. Думаю, аналогии ясны.

1.3 Инструменты для реализации GUI приложения

AWT

Самый первый GUI-фреймворк. Был довольно быстр и являлся частью JDK, однако плохо справлялся с кроссплатформенностью, да и много вещей просто не было реализовано.

Swing

Написан на основе AWT с упором на кроссплатформенность и простоту

использования. Теперь элементы GUI действительно ведут себя и выглядят одинаково на всех платформах, однако от этого пострадала скорость работы.

JavaFX

Позиционируется Oracle как замена Swing и активно развивается в данный момент. Поддерживает аппаратное ускорение, а также CSS и XML.

libGDX

Фреймворк, заточенный конкретно под разработку игр. Поддерживает портирование под практически все актуальные платформы, имеет открытый исходный код.

Для курсового проекта была выбрана Java в связке с JavaFX для реализации графического интерфейса.
Java платформонезависима и позволяет относительно легко портировать приложение под любую актуальную платформу. JavaFX же на данный момент является самой актуальной графической библиотекой на Java. Для игр, правда, он не так хорош, как, например, libGDX, но лучше подойдёт для страждущих писать на чистой Джаве. Да и не буду лукавить – когда я узнал о существовании libGDX, половина курсовой уже была написана, и переписывать львиную долю кода совершенно не хотелось.

Портирование на Андроид осуществляется с помощью JavaFXPorts – единственного адекватного решения на данный момент. Есть ещё Gluon, однако он совершенно не подходит для моих целей. Платный, и заточенный под прикладные приложения.

В случае с Linux и MacOS портирование вообще не требует никаких телодвижений – нужен лишь Java Runtime.

1.4 Протокол передачи данных

Для передачи данных между сервером и клиентом был использован протокол TCP. Он представляет собой поток данных с предварительной установкой соединения, что гарантирует доставку пакетов. Также он поддерживает повторную передачу пакетов, механизм, исключающий дублирование пакетов и уведомление и доставке. Он прост в использовании и нативно поддерживается на любой современной ОС, так что это неплохой выбор для первой попытки в онлайн. Он, конечно, медленнее, чем UPD, однако для пошаговой игры это не имеет никакого значения.

2. Структурное проектирование

2.1 Архитектура

Архитектура игры делится на две большие части: движок и игровую логику. Движок отвечает за контроль отрисовки, объектов и ввода. Игровая логика – за собственно игру, как бы странно это ни звучало.

Движок состоит из следующих классов:

Game
Главный игровой класс, отвечает за инициализацию окна игры и главный цикл.

GameWorld
Инициализирует игру и обновляет главные контроллеры.
Методы:
CREATE()
Инициализирует контроллеры, создаёт главное меню и загружает ресурсы.
UPDATE()
Обновляет контроллеры. Выполняется каждый шаг.

Obj
Контроллер объектов. Хранит список экземпляров и каждый шаг выполняет их события. Бонусом имеет полезные функции вроде подсчёта и поиска объектов.
Методы:
UPDATE()
Выполняет события всех игровых объектов.
objCount(oid obj)
Считает количество объектов заданного типа.
objDestroy(GameObject obj)
Удаляет объект.
objIndexCmp(GameObject obj,oid id)
Сравнивает индексы.

Obj.oid
Список типов объектов. Позволяет создавать перечисления однотипных объектов и легко идентифицировать их в общей куче.

GameObject
Абстрактный игровой объект. Должен быть в родителях у каждого игрового объекта.
Методы:
STEP_ BEGIN()
STEP()
STEP_END()
DRAW_BEGIN()
DRAW()
DRAW_END()
DRAW_GUI()

DESTROY()
* Описание см. ниже

ObjIter
Итератор игровых объектов. Служит для пробега по всем экземплярам.
Методы:
get()
Возвращает текущий объект.
end()
Проверяет, закончился ли список объектов.
inc()
Инкремент.

Draw
Работает с конвейером отрисовки, имеет множество функций рисования фигур и спрайтов.
Методы:
CREATE()
Инициализирующий метод.
UPDATE()
Обновление отрисовки.
setDepth(int depth)
Задание глубины отрисовки.
setColor(Color c)
Задание цвета отрисовки. Распространяется только на фигуры.
setAlpha(double a)
Задание прозрачности отрисовки. Распространяется только на фигуры.
draw(Node dr)
Универсальный метод рисования. Любая фигура JavaFX, в том числе и спрайты, может быть нарисована с его помощью.
drawCircle(), drawLine(), drawRectangle()
Рисование простейших фигур.
drawSprite(…)
Рисование спрайта.
d*rawPolyBegin(Polyline poly,Boolean outline)*
Начало рисования примитива.
polyAddPoint(double x,double y)
Добавление точки в примитива.
drawPolyEnd()
Конец рисования примитива.

Draw.df
Список флагов отрисовки.

Input
Следит за состоянием мыши.
Методы:
CREATE()
Инициализация контроллера.
UPDATE()
Обновление контроллера.
mouseClear()
Сброс нажатия кнопок мыши.

Mathe
Сборник часто используемых пользовательских функций.
Методы:
angleDifference(double ang1,double ang2)

Разница между двумя углами в градусах.
lcos(double len,double dir)
Косинус, умноженный на длину.
lsin(double len,double dir)
Синус, умноженный на длину.
lerp(double xp,double x,double xf)
Линейная интерполяция.
pointDirection(double x1,double y1,double x2,double y2)
Направление в градусах между двумя точками.
pointDistance(double x1,double y1,double x2,double y2)
Расстояние между двумя точками.
pointInRectangle(double xm,double ym,double x1,double y1,double x2,double y2)
Проверка, находится ли точка в прямоугольнике.
rectangleInRectangle(double x1a,double y1a,double x2a,double y2a,double x1b,double y1b,double x2b,double y2b)
Проверка, находится ли прямоугольник в прямоугольнике.
choose(T... arr)
Возвращает один из случайно выбранных аргументов.
irandom(int x), irandom(int x1,int x2)
Возвращает случайное целое число.
random(), random(double x), random(double x1,double x2)
Возвращает случайное вещественное число.
randomGetSeed()
Возвращает текущий сид.
randomize()
Задаёт сид, основанный на текущем времени.
randomPop()
Сохраняет текущий генератор рандома в стек.

randomPush()
Достаёт из стека генератор рандома.
randomSetSeed(long seed)
Задаёт сид.
rotateConvert(int x,int y)
Конвертирует единичный вектор в число от 0 до 3.
zerosign(double x)
Возвращает знак числа, или 1, если оно равно нулю.

Sprite
Спрайт с поддержкой анимаций.
Методы:
setOffset(int offx_arg,int offy_arg)
Задаёт смещение.
getWidth(), getHeight()
Возвращает высоту или ширину спрайта.
setBlendMode(BlendMode blend)
Задаёт режим смешивания.
setAlpha(double alpha)
Задаёт прозрачность.
setFrame(double frame)
Задаёт кадр.

Spr
Список спрайтов.

Camera
Игровая камера. Поддерживает движение, скейл и вращение.
Методы:
setPosition(double x,double y)
Задаёт координаты камеры.
get_x(), get_y()
Возвращает коорзинаты камеры.
setScale(double xscale,double yscale), setScale_x(double xscale), setScale_y(double yscale)
Задаёт скейл.
getScale_x(), getScale_y()
Возвращает скейл.
setScaleTar(double xscale,double yscale)
Плавно задаёт скейл.
setRotate(double rot)

Задаёт вращение.
getRotate()
Возвращает вращение.

Игровая логика состоит из следующих классов:

Lobby
Главное меню.
Методы:
createLocalGame()
Инициализирует локальный матч.
createNetworkGame(Cmd cmd)
Инициализирует онлайн-матч.
createBotGame()
Инициализирует матч против бота.
createAutomaticGame()
Инициализирует автоматический матч.
createReplayGame()
Инициализирует реплей.

Paper
Кусочек бумаги на фоне в меню.

Terrain
И песен певец, и на дуде игрец. Другими словами, основа всей игровой сессии. Отвечает за террейн, инициализацию сессии, интерфейс, перетаскивание камеры и таймер. Возможно, стоило бы приправить всю эту кучу щепоткой ООП, но диаграмма классов и так выглядит несколько устрашающе.

TerrainGenerator
Генерирует террейн из заданного сида.
Методы:
islandAdd(int x,int y,int h)
Добавляет остров в список генерации.
terrainGenerate(int w,int h)
Генерирует террейн.
terrainIslandSpineGenerate(int[][] terr,int spineDir,int bones,double x0,double y0,int hMax)
Генерирует основу для острова.
terrainSpineGenerate(int[][] terr)

Выполняет несколько вызовов terrainIslandSpineGenerate() для генерации террейна.
terrainBeefup(int[][] terr)
Генерирует полноценные острова из основ.
terrainBridgeAdd(int[][] terr,int sx,int sy,int fx,int fy)
Проводит мост между двумя точками.
treesGenerate(int[][] terr)
Генерирует деревья.
terrainAutotile(int[][] terr,Sprite[][] terrSpr)
Создаёт карту тайлов на основе сгенерированного террейна.

TileProp
Хранит в себе свойства для различных типов тайлов террейна.
Методы:
init()
Инициализирует свойства тайлов.
tileAdd(boolean ground,boolean passable,Sprite spr,int depth), tileAdd(boolean ground,boolean passable)
Добавляет новый тайл с заданными свойствами.
getSpr(int i)
Возвращает спрайт заданного тайла.
getDepth(int i)
Возвращает глубину заданного тайла.
isPassable(int i)
Проверяет, является ли тайл проходимым..
isGround(int i)
Проверяет, является ли тайл землёй.

TurnManager
Игра пошаговая, так что ей нужен менеджер, чтобы игроки ждали друг друга и ходили по очереди.
Методы:
playerAdd(Player player)
Добавляет нового игрока.
playerGet(int i)
Возвращает игрока с заданным индексом.
initiativeGive()
Передаёт инициативу следующему игроку.
initiativeTake()
Забирает инициативу у текущего игрока.
getCurrentPeasant()

Возвращает юнита с инициативой.
isCurrentPlayerLocal()
Возвращает, является ли текущий игрок локальным.

Player
Класс-родитель для всех игроков.
Методы:
peasantAdd(Peasant peasant,Logger logger)
Добавляет существующего юнита под контроль игрока.
peasantRemove(Peasant peasant)
Удаляет юнита.
initiativeGive()
Отдаёт инициативу.
endTurn()
Завершает ход.
isMyTurn()
Проверяет свой ход.
getCurrentPeasant()
Возвращает текущего юнита.

LocalPlayer
Локальный игрок. Управляется напрямую.

NetworkPlayer
Удалённый игрок. Получает и выполняет команды, приходящие с сервера.

BotPlayer
Игрок под управлением довольно глупого бота.
Методы:
AI()
ИИ бота.

ReplayPlayer
Игрок, повторяющий команды записанной ранее игры.

Logger
Записывает все действия юнитов в файл для возможности их воспроизведения. Также может парсить существующие логи.
Методы:
write(double x)
Записывает число в файл.

write(Cmd cmd)
Записывает атрибуты команды в файл.
read()
Читает лог и генерирует на его основе массив команд.
close()
Закрывает файл.

Entity
Любой объект, находящийся на террейне, должен являться сущностью. Это позволяет писать простые взаимодействия и упрощает поиск пути. Не относится к самим тайлам и террейну.
Методы:
tileStore()
Сохраняет тайл, на котором стоит и записывает вместо него непроходимую стену.
tileRestore()
Восстанавливает сохранённый тайл.

Peasant
Юнит под управлением игрока. Умеет рубить деревья и убивать врагов.
Методы:
staminaRefill()
Восстанавливает запас сил.
interact()
Взаимодействует с тайлом или сущностью.
pathCheck()
Проверяет, проходим ли построенный ранее путь.

MatchResult
Выводит результат матча и завершает игровую сессию.

Pathfinder
Модифицированный поиск пути A*. Используется для генератором, ИИ и юнитами.
Методы:
pathFind(int sx,int sy,int fx,int fy)
Строит путь между двумя точками. Если пути не существует, возвращает NULL.
pathConstruct(PathCell ptStart)
Выстраивает путь по наброску.
listFindLesser(ArrayList<PathCell> list)

Находит наименьший элемент в списке.
getDist(int sx,int sy,int fx,int fy)
Вычисляет очень приблизительное расстояние от одной точки до второй. Функция нужна лишь для относительной проверки расстояний, упор сделан в скорость вычислений.

PathCell
Служебный класс для алгоритма поиска пути. Хранит в себе отмеченную поиском клетку.
Методы:
getValue()
Возвращает своё значение, основанное на позиции.

PathPoint
Односвязный список, хранящий путь.
Методы:
add(int x_arg,int y_arg)
Добавляет новую точку в путь.
last()
Возвращает последний элемент пути.
length()
Возвращает длину пути.

Client
Обеспечивает взаимодействие с сервером.
Методы:
connect()
Соединяет приложение с сервером.
disconnect()
Отключается от сервера.

Reader
Отдельный поток, читающий команды, приходящие с сервера.
Методы:
run()
Поток чтения.
isReceived()
Проверяет, пришла ли команда с сервера.
read()
Считывает команду.
watch()

Считывает команду, но при этом не сбрасывает её в ридере.

Sender
Посылает объекты на сервер.
Методы:
send(Object obj)
Посылает объект на сервер.
receive()
Получает объект.
close()
Закрывает сокет.

Cmd
Команды, которыми обмениваются клиент и сервер. Они же используются для управления юнитами. Состоит из строки-команды и массива аргументов.
Методы:
cmp(String str)
Сравнивает команду со строкой.
get()
Возвращает атрибут команды.
size()
Возвращает количество аргументов.

Более подробную информацию о классах и их свойствах можно посмотреть в Javadoc (приложение Е).

Исходный код можно посмотреть на https://github.com/gnFur/Bomb_n_Shovel

2.2 Ядро приложения

Основа приложения состоит в следующем: в начале шага метод UPDATE() в GameWorld поочерёдно вызывает UPDATE() у классов Camera, Input, Obj и Draw.

При обновлении Camera изменяет свою позицию, если следит за каким-то игровым объектом. Input считывает и запоминает состояние мыши. В Obj поочерёдно выполняются события всех игровых объектов.
Каждый игровой объект содержит восемь событий:

- STEP_ BEGIN()
- STEP()
- STEP_END()
- DRAW_BEGIN()

- DRAW()
- DRAW_END()
- DRAW_GUI()
- DESTROY()

Порядок выполнения событий таков: сначала для всех объектов выполняется STEP_BEGIN(), затем STEP(), и так далее. В событиях STEP должна находиться основная логика, в DRAW() – рисование. Ничто не мешает, однако, рисовать прямо в STEP, однако там флаг отрисовки всегда задан на обычный DRAW(). Флаг отрисовки определяет, в какой последовательности будет рисоваться графика при вызове UPDATE() в Draw. К примеру, всё, что рисуется в DRAW_BEGIN() всегда будет рисоваться раньше, чем то, что рисуется в DRAW(). DRAW_GUI() в этом плане несколько особенный, поскольку рисует на отдельный слой ГУИ, на который не действует камера. Событие DESTROY() вызывается при удалении объекта.

3. Разработка структуры приложения

3.1 Обзор приложения

Управление осуществляется мышью. Главное меню содержит четыре режима игры: Локальный мультиплеер, Сетевой мультиплеер, Матч против бота и Автономный режим. В нижнем левом углу находится кнопка включения\выключения лимита времени. Рядом с ней – кнопка просмотра реплея предыдущего матча.

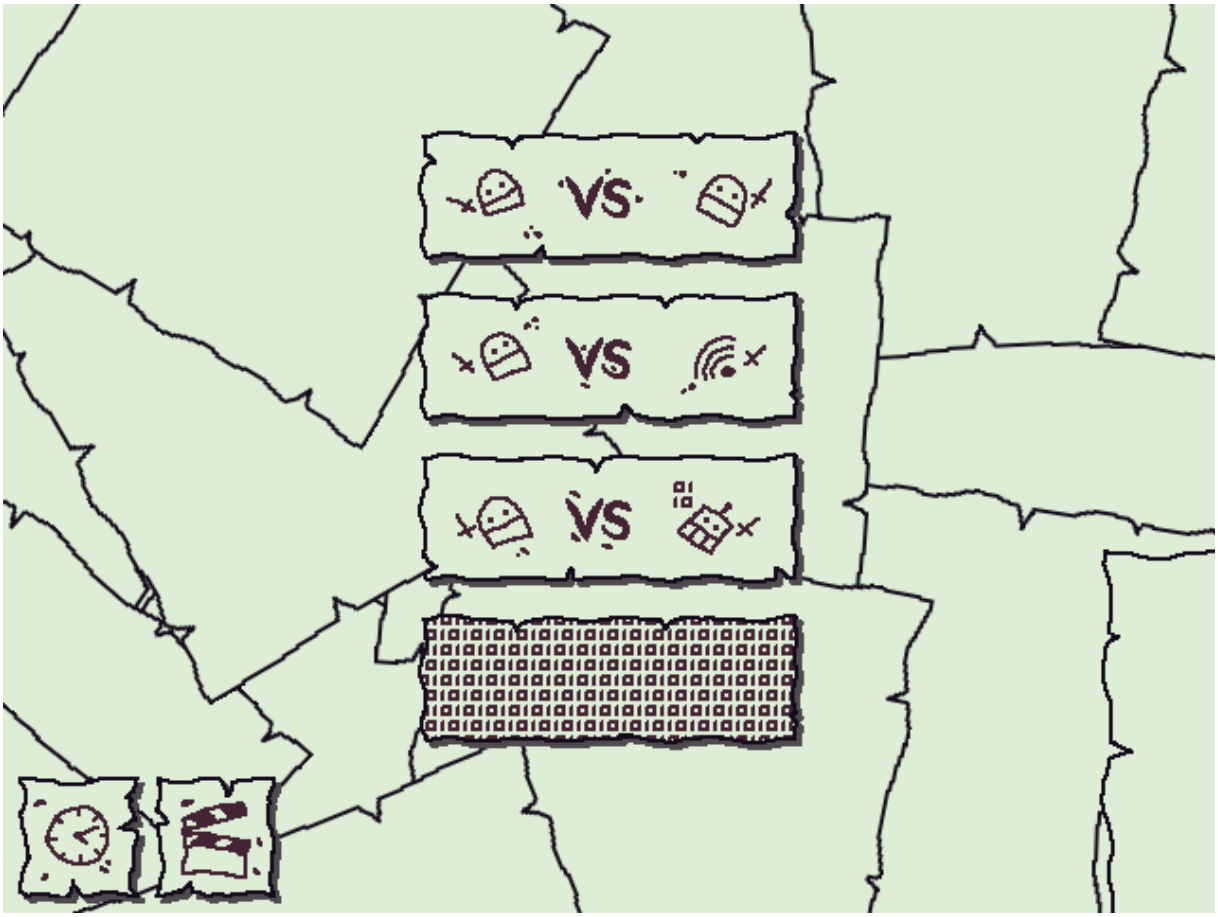

Рис. 3.1

При выборе сетевой игры игрок попадает в лобби, где ждёт, пока сервер соединит его с другим игроком. В левом нижнем углу имеется кнопка выхода обратно в главное меню, в центре – милый котик.

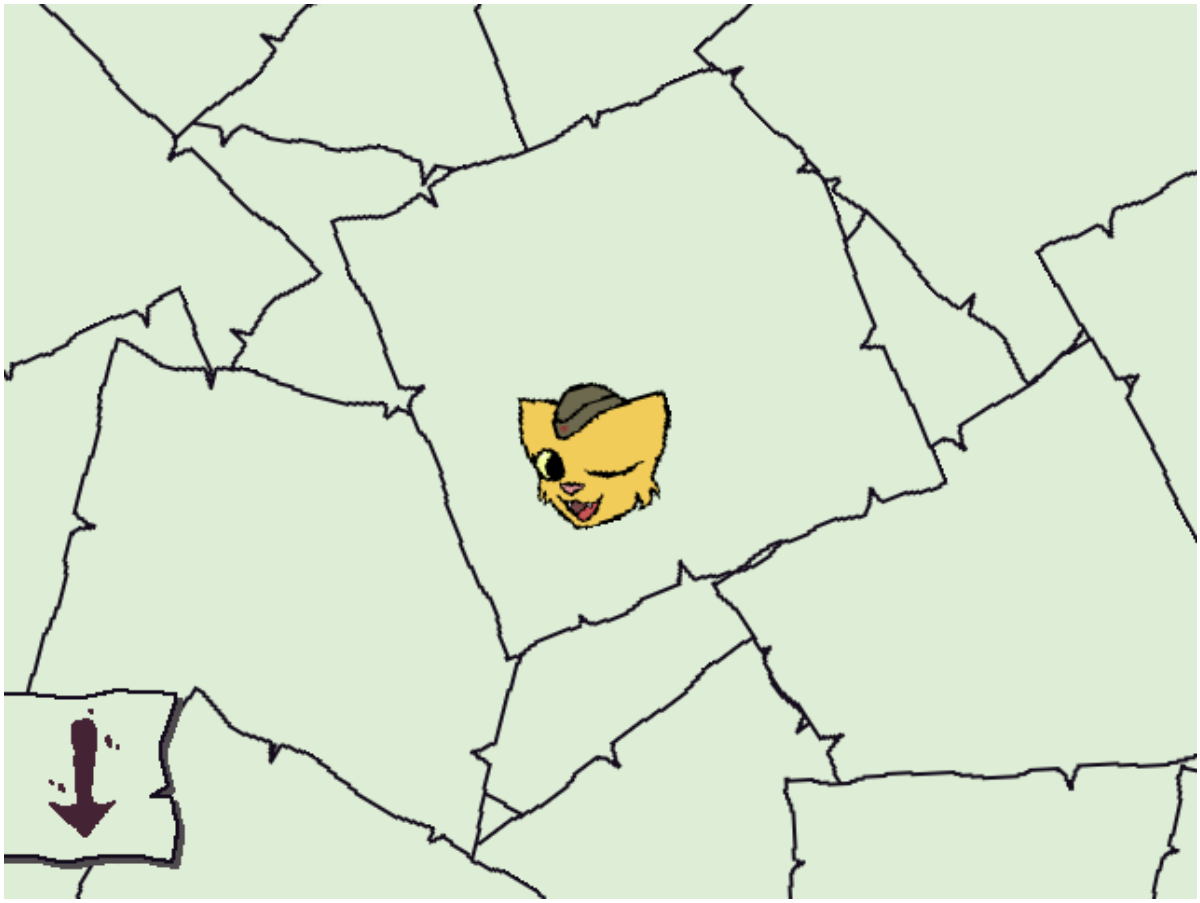

Рис. 3.2

При соединении с другим игроком или выборе одного из других режимов игры, начинается собственно геймплей. В левом нижнем углу находится таймер, если выбран режим игры на время, в правом – кнопки обзора и завершения хода.

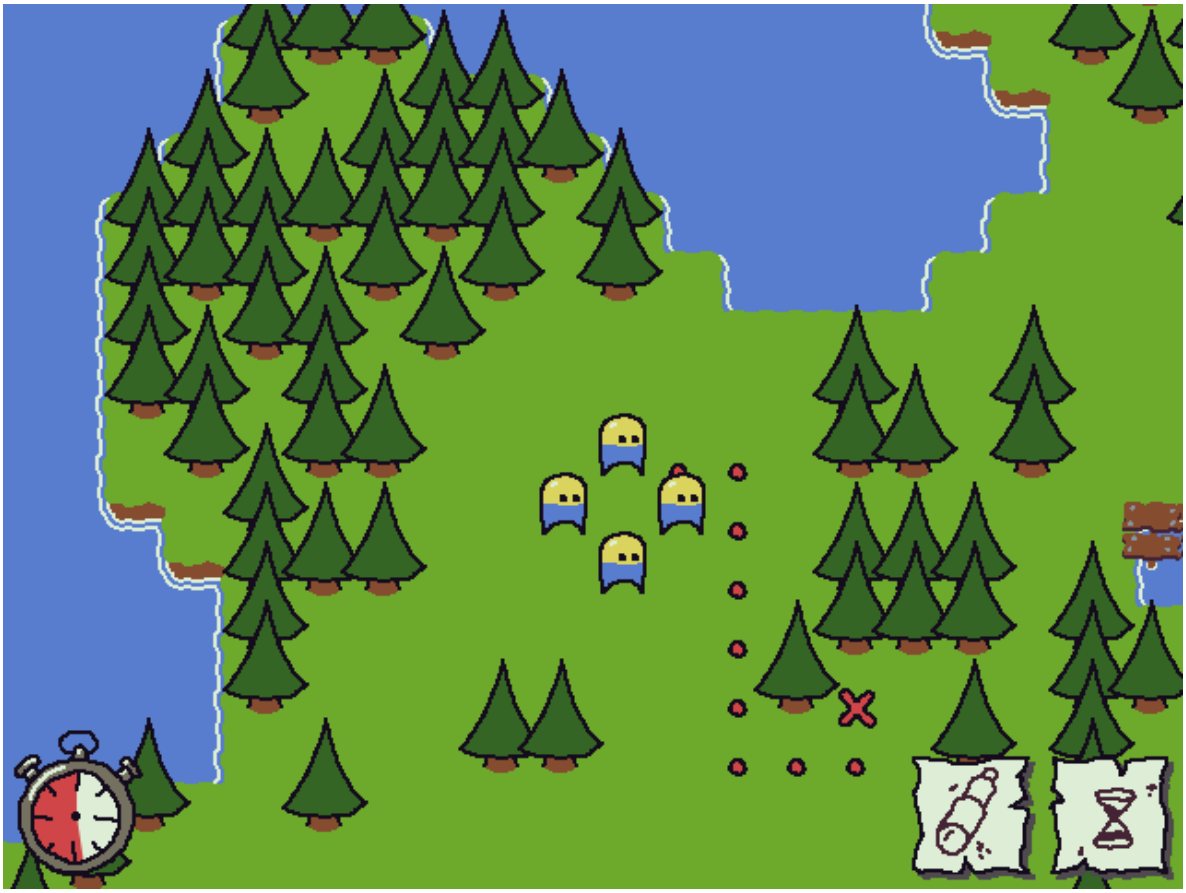

Рис. 3.3

В начале игры после рандомной генерации карты каждому игроку даётся четыре юнита. Юнит может ходить по траве и мостикам, рубить деревья и убивать вражеских юнитов. Цель игры – убить всех юнитов врага.

3.2 Android-порт

Как было сказано ранее, для портирования использовался JavaFXports. Приложение удалось запустить, однако производительность оказалось печальной. Очень печальной. Попытки исправить ситуацию особо ни к чему не привели.
Также на Рис. 3.4 можно заметить множество артефактов, которые тоже не пожелали исправляться.

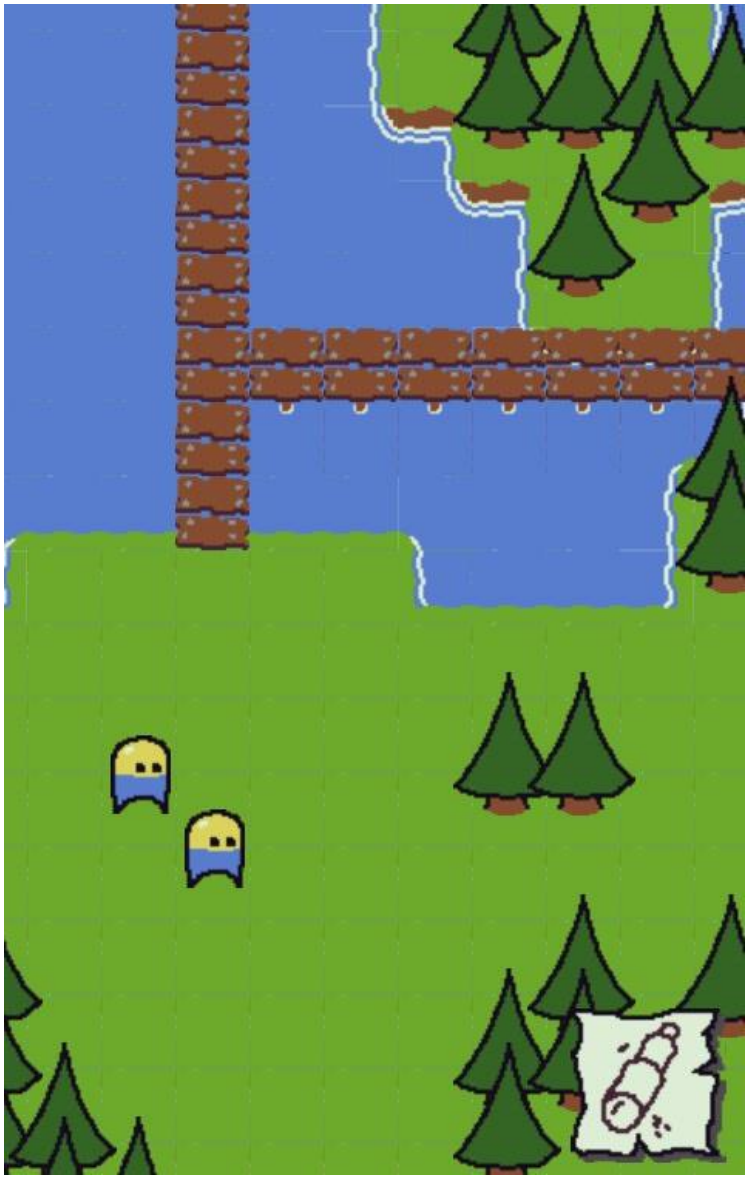

Рис. 3.4

Что же пошло не так? Во-первых, одна из проблем оказалась в баге самой основы игра – цикла фпс. Он считал время неправильно, и игра работала на максимальных для JavaFX 60 фпс вместо 30. Большая часть Android-устройств физически не способна вытянуть 60 фпс. Я знал об этом с самого начала, и заранее залочил игру на 30 фпс… не осознавая практически до конца разработки, что игра на самом деле работает на всех шестидесяти. Однако даже с учётом этого, большую часть времени игра на Андроиде не дотягивала даже до тридцати фпс, а десктопная версия с локом 30 фпс по какой-то странной причине выглядела просто ужасно. Так что я выбрал меньшее из двух зол, и оставил 60 фпс. Пускай игра хоть где-то выглядит хорошо.

Касательно артефактов у меня два предположения. Такое обычно случается, когда спрайт пытается нарисоваться в дробных координатах, однако все координаты у меня целочисленные, а отдельные тайлы на чанках

вообще рисуются только один раз на сюрфейс. Также подобное возникает при неправильной программной обрезке картинки. Для фикса пришлось бы рисовать спрайты особым образом и дописывать много кода. Однако я решил, что стоит оставить всё как есть. Всё равно фпс близок к неиграбельному, и, если игра и дойдёт до стадии релиза, то точно не на JavaFX.

4. Разработка программных модулей

Модули – это всегда хорошо. Легко достать из проекта, легко встроить в новый. Собственно, помимо модулей движка и игровой логики, описанных выше, можно выделить ещё и онлайн-модуль клиента.

4.1 Модуль взаимодействия с сервером

Модуль состоит из трёх классов: Client, Sender и Reader. Client – главный класс, который подключается к серверу и создаёт объекты чтения-записи. Reader – отдельный поток чтения. Асинхронно принимает объекты с сервера и сохраняет их в свой буфер. Sender заведует потоком записи.

4.2 Модуль сервера

Сервер вынесен в отдельное приложение. Его диаграмму классов можно посмотреть на Рис. 4.1.

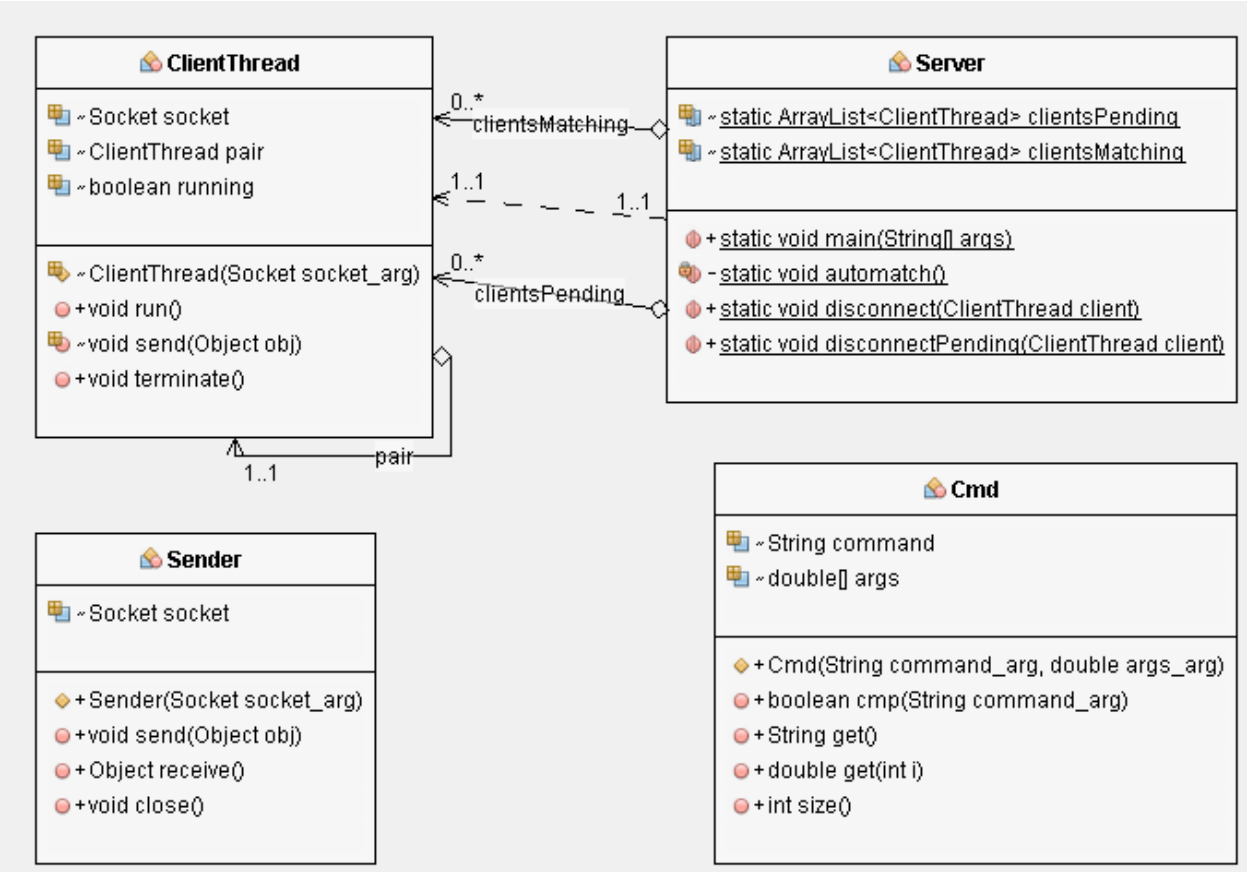

Рис. 4.1

Главный класс сервера Server принимает подключившихся клиентов и записывает их в список ClientsPending. Так как клиентов может быть не один и не два, каждому из них создаётся поток ClientThread. В режиме pending этот поток посылает своему клиенту команду ping, чтобы следить за его отключением. Когда на сервер приходит ещё один клиент, Server автоматически создаёт пару из двух клиентов и переводит оба их потока в режим matching. В этом режиме поток ожидает приходящих команд с клиента и при получении пересылает их своей паре.

Получается, что сервер служит эдаким ретранслятором и не имеет в себе никакой игровой логики.

При внезапном отключении одного из клиентов поток на сервере посылает своей паре команду disconnect, говорящую о том, что клиент отключился, и завершается.

Пересылка команд осуществляется по протоколу TCP с использованием сериализации объектов. Сериализация – перевод какой-то структуры данных, в нашем случае, объекта класса Cmd, в последовательность битов. Такой подход упрощает работу с командами. Не надо ничего интерпретировать, десериализовал – и пользуйся. Однако есть у этого подхода и большой минус – класс у клиента и у сервера должен быть стопроцентно идентичным. Совпадать должны не только поля, но и все методы, даже пакет должен быть один и тот же, иначе объект просто не десериализуется. Это создавало некоторый геморрой в процессе разработки.

Сам же класс Cmd содержит в себе строку-команду и массив даблов на случай, если нужны какие-то аргументы. Такой подход позволил создать единый интерфейс и для пересылки данных, и для управления юнитами – они получают и напрямую интерпретируют эти же команды.

Заключение

Джава всё-таки чертовски приятный язык. Сравнивая с опытом предыдущего семестра и попыткой написать игру на C++, процесс был гораздо плавнее. Уверен, с C++ я не успел бы и половины того, что я реализовал в этом курсовом. А уж о порте на другие платформы и речи бы не шло. Конечно, порт под Андроид вышел несколько… провальным, но тут уже ничего не поделать. Времени было очень мало, работы много, и инструмент был выбран немного не тот. Использовать JavaFX для игр оказалось несколько сомнительно. Всё же, если в будущем встанет подобная задача, стоит попробовать libGDX.

Также стоит отметить, что в этот раз откусил я кусок, который не способен проглотить. Всё-таки стратегия, даже такая минималистичная, пока что сложновата для моей геймдизайнерской мысли, особенно учитывая то, что я понятия не имел, какая корневая идея будет заложена в игре. В проекте упор сделан был в техническую часть как раз из-за этого. Да, есть генератор карт, есть онлайн, есть бегающие-дерущиеся болванчики… А вот геймплей вообще почти отсутствует.

Было много идей – добавить элементы строительства, миниинвентарь каждому юниту, какой-то крафт из материалов на карте, дать возможность развернуться хоть какой-то стратегии. А вот времени не было, как и главной киллер-фичи, за которую идеи бы цеплялись. В любом случае, ценный опыт был получен, я поработал со многими вещами, к которым особо не решался притрагиваться.

Литература

https://ru.wikibooks.org/wiki/TCP/IP

https://ru.wikipedia.org/wiki/NAT

http://www.softzenware.com/java/14.htm

https://xakep.ru/2014/09/10/java-gui/

К. Сьерра, Б. Бейтс - Изучаем Java (Мировой компьтерный бестселлер) - 2012

Б. Эккель - Философия Java