Белорусский государственный университет информатики и радиоэлектроники

Отчет по курсу КПП на тему:

«Bomb ‘n Shovel»

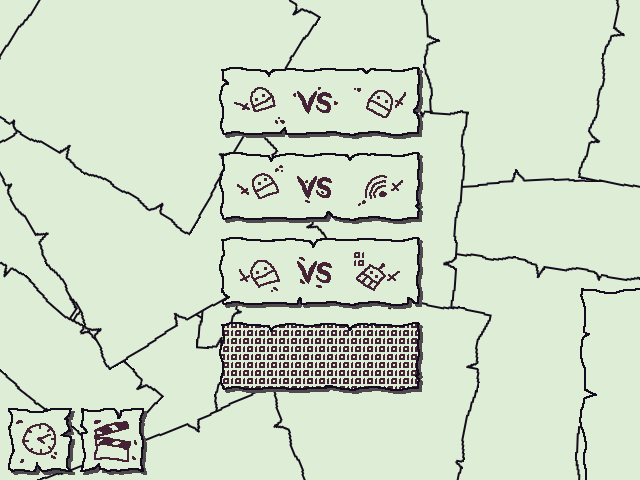
|  |  |
| --- | --- |
| Выполнил:  Савин Д. В. | Проверил:  Кухарчук И.В.  Лектор:  Искра Н.А. |

Минск

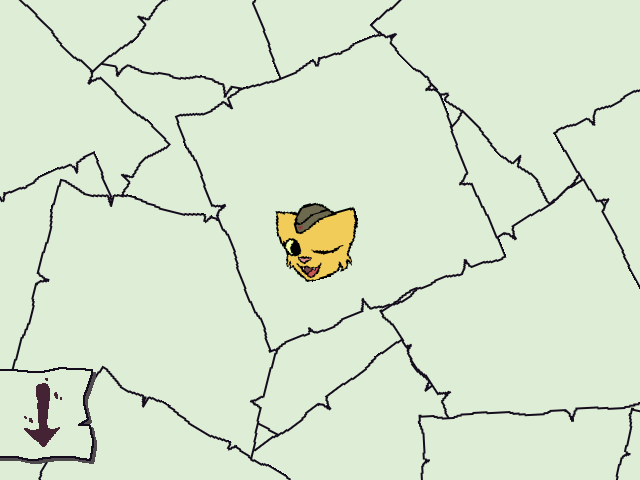
2017

**Лабораторная работа №1.   
Разработка кроссплатформенного интерфейса**

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. 1.1

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. 1.2

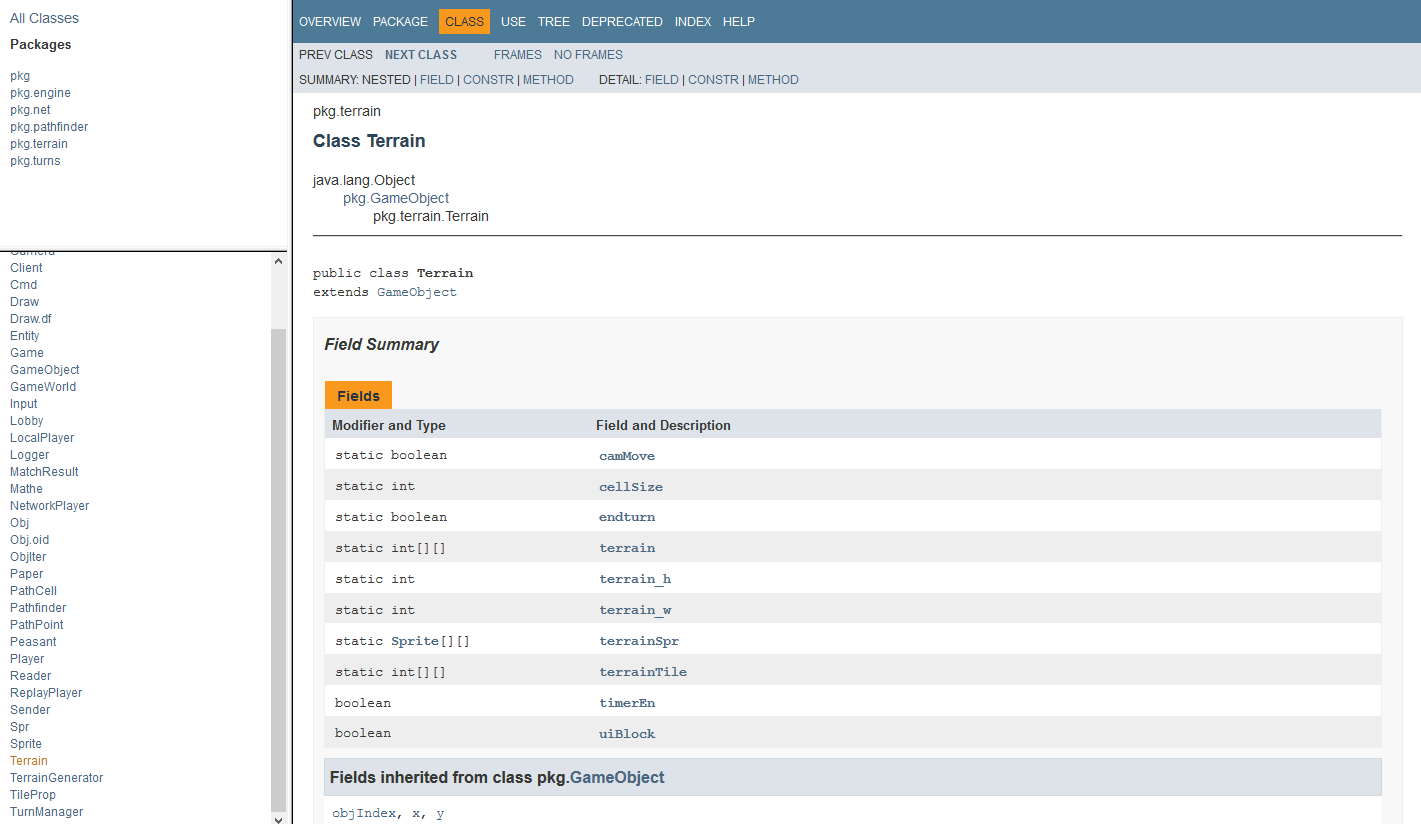
При соединении с другим игроком или выбрав один из других режимов игры, начинается собственно геймплей. В левом нижнем углу находится таймер, если выбран режим игры на время, в правом – кнопки обзора и завершения хода.

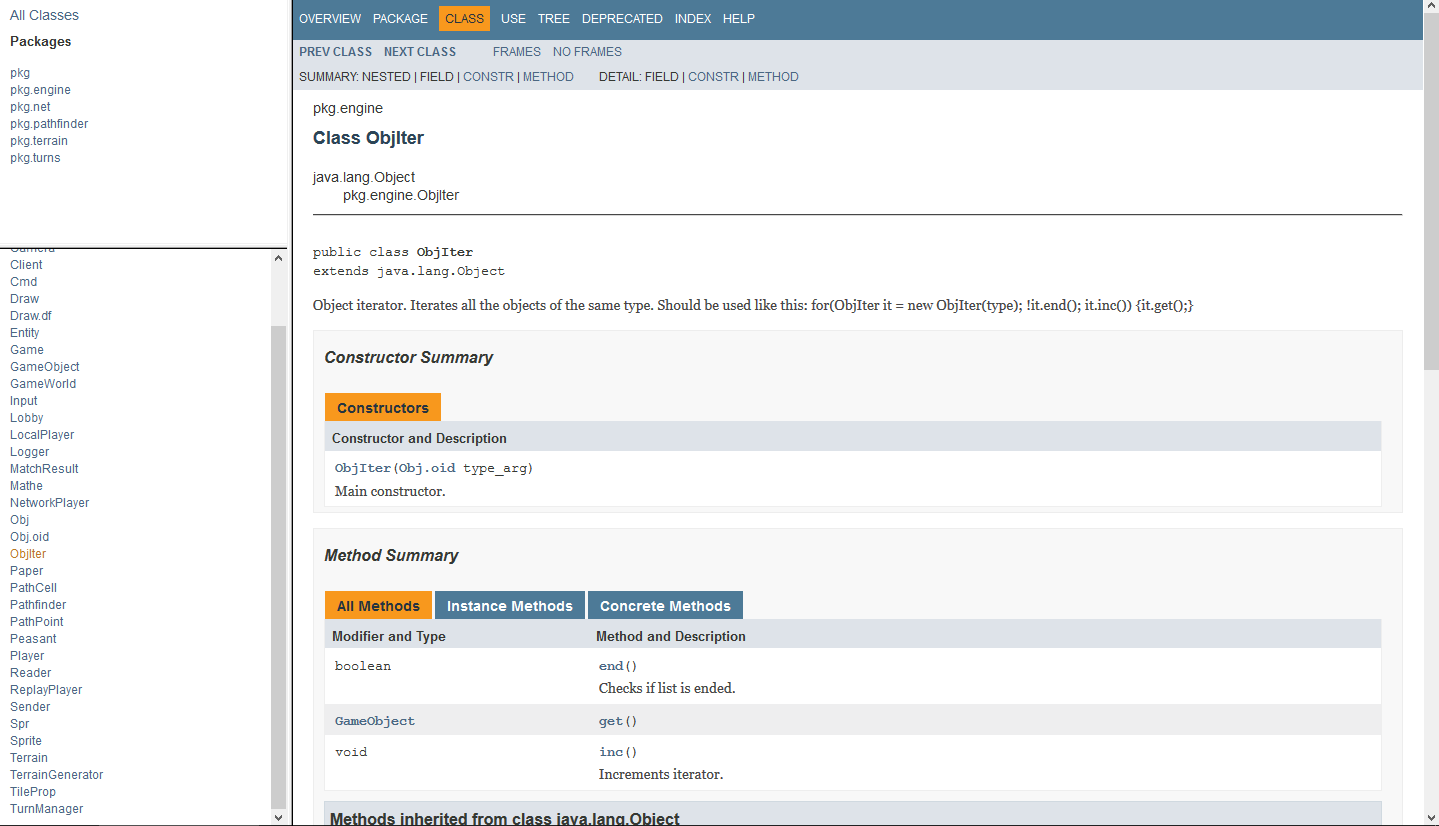
  
Рис. 1.3

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.

**Лабораторная работа №2.   
Программирование алгоритмов с использованием механизмов объектно-ориентированного программирования**

В ходе двух занимательнейших месяцев выполнения лабораторной работы был разработан прототип игры Bomb ‘n Shovel. Диаграмма классов находится в приложении А. Также были проведены тесты на приношение интерфейсом радости.   
 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.  
  
*Obj*  
Контроллер объектов. Хранит список экземпляров и каждый шаг выполняет их события. Бонусом имеет полезные функции вроде подсчёта и поиска объектов.   
 *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.  
  
*ObjIter*  
Итератор игровых объектов. Служит для пробега по всем экземплярам.  
  
*Draw*  
Работает с конвейером отрисовки, имеет множество функций рисования фигур и спрайтов.  
  
*Draw.df*  
Список флагов отрисовки.  
  
*Input*  
Следит за состоянием мыши.  
  
*Mathe*  
Сборник часто используемых пользовательских функций.   
  
*Sprite*  
Спрайт с поддержкой анимаций.  
  
*Spr*  
Список спрайтов.  
  
*Camera*  
Игровая камера. Поддерживает движение, скейл и вращение.  
  
  
Непосредственно радость приносят следующие классы:  
  
*Lobby*Главное меню.  *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. *TileProp*Хранит в себе свойства для различных типов тайлов террейна. *TurnManager*Игра пошаговая, так что ей нужен менеджер, чтобы игроки ждали друг друга и ходили по очереди. *Player*Класс-родитель для всех игроков. *LocalPlayer*Локальный игрок. Управляется напрямую.  *NetworkPlayer*Удалённый игрок. Получает и выполняет команды, приходящие с сервера.  *BotPlayer*Игрок под управлением довольно глупого бота. *ReplayPlayer*Игрок, повторяющий команды записанной ранее игры. *Logger*Записывает все действия юнитов в файл для возможности их воспроизведения. *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. *Peasant*Юнит под управлением игрока. Умеет рубить деревья и убивать врагов. *MatchResult*Выводит результат матча и завершает игровую сессию. *Pathfinder*Модифицированный поиск пути А\*. Используется для генератором, ИИ и юнитами. *PathCell*Служебный класс для алгоритма поиска пути. Хранит в себе отмеченную поиском клетку. *PathPoint*  
Односвязный список, хранящий путь.  
  
*Client*Обеспечивает взаимодействие с сервером.   
*Reader*Отдельный поток, читающий команды, приходящие с сервера. *Sender*Посылает объекты на сервер. *Cmd*Команды, которыми обмениваются клиент и сервер. Они же используются для управления юнитами. Состоит из строки-команды и массива аргументов.  
 Основная архитектура 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 ~~случается вся магия~~ 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.   
 Некоторые скриншоты Javadoc представлены на рис. 2.1 и 2.2. Полная версия в формате html находится в приложении Е.

 Рис. 2.1

Рис. 2.2

Блок-схемы, описывающие алгоритм генерации террейна, представлены в приложениях Б, В и Г.

**Лабораторная работа №3.   
Разработка программ с использованием модульного подхода**

Для реализации сохранения и загрузки была использована следующая нотация:

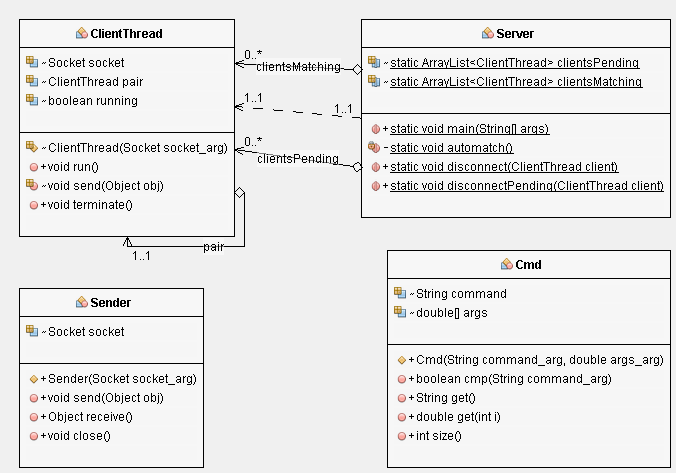
*Таблица 1.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Состояние | Интерпретация | Сохранение в файле | | |
| Клик по игровому полю. | Задание пути | “setpath” | x | y |
| Клик по пути. | Движение | “move” | - | - |
| Нажатие на кнопку завершения хода или истечение времени. | Завершение хода | “endturn” | - | - |

Такая нотация позволяет не только сохранять реплей матча, но и реализовать унифицированную систему управления юнитами, которая гораздо упрощает создание удалённого игрока и бота, а также передачу команд по сети.

**Лабораторная работа №4.   
Разработка многопоточных программ, использование стандартных примитивов синхронизации**

Для клиент-серверного взаимодействия выделено отдельное приложение-сервер. Его диаграмма классов представлена на Рис. 4.1.

  
Рис. 4.1

*Server*  
Главный класс сервера. Следит за подключившимися клиентами и автоматчит их. Поддерживает множество матчей сразу.  
  
*ClientThread*   
Каждому подключившемуся клиенту выделяется отдельный поток. Он принимает сообщения от клиента и переправляет их своей паре.  
  
*Sender*   
Копия одноимённого класса из приложения-клиента.  
  
*Cmd*  
Копия одноимённого класса из приложения-клиента.

Рис. 4.2



На рис. 4.2 представлено клиент-серверное взаимодействие. Подключение двух клиентов, пересылка сообщения и преждевременное отключение одного из клиентов.