Содержание

ВВЕДЕНИЕ

Компьютерные сети представляют собой одно из самых мощных и универсальных средств коммуникации.

Они позволяют создавать программы, связывающие пользователей, находящихся в различных точках земного шара для различных целей: обмена информацией, решения научных и производственных задач, развлечений.

Одной из самой интенсивной областей развития сетевого программирования является программирование сетевых компьютерных игр.

В данной работе обсуждается создание сетевой компьютерной версии классической игры "Монополия и приводится программа на C++, реализующая клиентскую и серверную части игры. Отчёт содержит также описание функций, инструкцию пользователю и тестовый пример.

1. СЕТЕВАЯ ИГРА МОНОПОЛИЯ

1.1. Содержательное описание задачи

Создать приложение, позволяющие провести партию игры "Монополия"между игроками за компьютерами, связанными в сеть TCP/IP.

1.2. Формальная постановка задачи

Необходимо создать клиент и сервер для компьютерной версии игры "Монополия позволяющую игрокам удобно создать и провести сеанс игры.

Краткое описание игры.

Монополия — это классическая игра, в которой вы можете покупать, арендовать и продавать свою собственность! В начале игры участники выставляют свои фишки на поле «Вперед», затем перемещают их по игровому полю в зависимости от выпавшего на кубиках количества очков.

Если вы попадаете на Участок Недвижимости, которая пока еще никому не принадлежит, то вы можете купить эту Недвижимость у Банка. Если вы решаете не покупать ее, она может быть продана на Аукционе другому игроку, предложившему за нее самую высокую цену. Игроки, имеющие Недвижимость, могут взимать арендную плату с игроков, которые попадают на их Участок. При строительстве Домов и Гостиниц арендная плата значительно возрастает, поэтому вам следует вести строительсто на как можно большем количестве Участков.

Если вы нуждаетесь в деньгах, вы можете заложить вашу Недвижимость.

В ходе игры вам следует всегда выполнять указания, написанные на карточках «Общественная казна» и «Шанс». Но не расслабляйтесь — в некоторых случаях вас могут посадить в Тюрьму.

Цель

Остаться единственным необанкротившимся игроком.

Начало игры

Фишки всех игроков выстраиваются на поле «Вперед», после чего поочередно каждый делает свой ход.

Ход игры

Когда подошла ваша очередь, бросте кубики. Ваша фишка передвинется по доске вперед в направлении по часовой стрелке. Поле, на котором вы остановитесь, определяет, что вам надо делать. На одном поле одновременно могут находиться несколько фишек. В зависимости от того на каком поле вы оказались, вам предстоит:

1) купить участки для строительства или другую недвижимость

2) заплатить арендную плату, если вы оказались на территории недвижимости, принадлежащей другим игрокам

3) уплатить налоги

4) вытащить карточку «Шансов» или «Общественной Казны»

5) оказаться в тюрьме

6) отдохнуть на «Бесплатной стоянке»

7) получить зарплату в размере 200$

Прохождение поля «Вперед»

Всякий раз, когда вы останавливаетесь или проходите поле «Вперед», двигаясь по часовой стрелке, Банк выплачивает вам зарплату 200 000. Эту сумму можно получить дважды за один и тот же ход, если, например, вы оказались на поле «Шанс» или «Общественная казна» сразу после поля «Вперед» и вытащили карточку с надписью «Перейдите на поле вперед».

Покупка недвижимости

Если вы остановились на поле, обозначающем незанятую другими Недвижимость (то есть на участке для строительства не занятом ни одним из игроков), у вас будет право первого покупателя на его покупку. Если вы решили купить недвижимость, заплатите Банку деньги в сумме, указанной на игровом поле. В обмен вы получите право собственности на эту недвижимость (игровое поле окрасится в цвет

вашей фишки).

Если вы решили не покупать Недвижимость, она немедленно выставляется на аукцион. В этом случае ее приобретает тот из игроков, кто предложит за нее наибольшую цену. Отказавшийся от покупки Недвижимости игрок не принимает участия в торгах.

Если в результате аукциона ни один из игроков не купил (или не смог купить) Недвижимость, то она остается свободной.

Владение недвижимостью

Владение недвижимостью дает вам право взимать арендную плату с любых арендаторов, которые остановились на поле, обозначающем ее. Очень выгодно владеть недвижимостью всей цветовой группы — иными словами, владеть монополией. Если вы владеете всей цветовой группой, вы можете строить дома на любом участке Недвижимости этого цвета.

Остановка на чужой недвижимости

Если вы останавливаетесь на чужой Недвижимости, которая была преобретена ранее другим игроком, с вас могут потребовать арендную плату за эту остановку. Сумма арендной плату недвижимости может изменяться в зависимости от построенных на поле этой недвижимост и домов и отелей. Если вся Недвижимость одной цветовой группы принадлежит одному игроку, арендная плата, взимаемая с вас за остановку на любом участке этой группы удваевается при условии, что на участках группы нет построек. Однако, если у владельца всей цветовой группы хотябы один участок Недвижимости этой группы заложен, он не может взимать с вас двойную арендную плату. Если на участках Недвижимости были построены Дома и Отели, арендная плата с этих участков увеличивается. За остановку на заложенной Недвижимости арендная плата не взимается.

Остановка на поле коммунального предприятия

Если вы остановились на одном из таких полей, вы можете купить это Коммунальное предприятие, если оно еще никем не куплено. Как и при покупке другой недвижимости в этом случае вам придется уплатить Банку сумму, указанную на этом

поле.

Если вы решили не покупать эту Недвижимость, Коммунальное предприятие выставляется на аукцион и продается игроку, предложившему за него наибольшую сумму. Вы не можете принять участие в аукционе.

Если в результате аукциона ни один из игроков не купил (или не смог купить) Коммунальное предприятие, то оно остается свободной.

Если это Коммунальное предприятие уже приобретено другим игроком, он может потребовать с Вас арендную плату. Арендная плата такого предприятия составит четырехкратное количество очков, выпавших на кубиках (вы снова кидаете кубики, чтобы определить сумму арендной платы). Если игрок владеет обоими Коммунальными предприятиями, вы должны будете заплатить ему сумму, равную десятикратному колличеству выпавших очков.

Остановка на вокзале

Если вы первым остановились на таком поле, у вас будет возможность купить этот вокзал. При вашем нежелании приобретать Вокзал, он уходит на аукцион и продается игроку, предложившему за него наибольшую сумму. Вы не можете принять участие в аукционе.

Если в результате аукциона ни один из игроков не купил (или не смог купить) Вокзал, то он остается свободным.

Если у Вокзала уже есть хозяин, оказавшийся на нем должен заплатить арендную плату. Эта плата зависит от количества вокзалов у игрока-владельца вокзала, на котором вы остановились. Чем больше вокзалов у хозяина, тем плата больше.

Остановка на поле «Шанс» и «Общественная казна»

Остановка на таком поле означает, что вам достается одна из карточек соответствующей группы. Эти карточки могут потребовать, чтобы вы:

1) передвинули вашу фишку

2) заплатили деньги, например, налоги

3) получили деньги

4) отправились в Тюрьму

Вы должны немедленно выполнить указания, написанные на карточке. Если вы взяли карточку, на которой написано «бесплатно освободитесь из тюрьмы», вы можете оставить ее у себя до тех пор, пока она вам не понадобится, или же вы можете продать ее другому игроку по договорной цене.

Примечание: На карточке может быть написано, что вы должны переместить фишку на другое поле. Если в процессе движения вы пересекаете по часовой стрелке поле «Вперед», то получите 200$. Если вас отправляют в Тюрьму, то поле «Вперед» вы не пересекаете.

Остановка на поле налогов

Если вы остановились на таком поле, вам просто нужно уплатить соответствующую сумму в банк.

Бесплатная стоянка

Если вы остановились на таком поле, то просто отдохните до следующего вашего хода. Вы находитесь здесь бесплатно и не подвергаетесь никаким штрафам.

Тюрьма

Вас отправляют в Тюрьму, если:

1) Вы остановились на поле «Отправляйтесь в Тюрьму», или

2) Вы взяли карточку «Шанса» или «Общественной Казны», на которой написано «Отправляйтесь в Тюрьму», или

Если вы попадаете в Тюрьму по карточке, зарплата в размере 200$ вам не выплачивается, где бы вы до того не находились.

Чтобы выйти из Тюрьмы вам надо заплатить штраф в размере 50$ и продолжить игру.

После того, как вы пропустили три хода, находясь в Тюрьме, вы должны выйти из нее и уплатить 50$, прежде чем вы сможете передвинуть вашу фишкуна выпавшее на кубиках число полей.

Находясь в Тюрьме вы имеете право взимать арендную плату за вашу Недвижимость, если она не заложена. Если вы не были отправлены в Тюрьму, а просто остановились на поле Тюрьма в ходе игры, вы не платите штраф, так как вы «просто

посетили» ее. Следующим ходом вы можете двигаться дальше, как обычно.

Дома

После того, как вы собрали все участки Недвижимости одной цветовой группы, вы можете покупать Дома, чтобы поставить их на любом из имеющихся у вас участков. Это увеличивает арендную плату, которую вы можете взимать с арендаторов, останавливающихся на вашей Недвижимости. Вы можете покупать дома во время вашего хода перед броском кубика. Стоимость дома варьируется в зависимости от линии, к которой принадлежат цветовые группы Недвижимости. За один ход вы можете построить не более одного дома на полях принадлежащих одной цветовой группе.

Максимальное количество домов на одном участке — четыре.

Так же при необходимости вы можете продавать дома обратно в банк. Стоимость дома в этом случае будет такой же, за какую вы его приобретали.

Нельзя строить дома, если хотя бы один участок данной цветовой группы заложен.

Отели

Прежде, чем вы сможете покупать отели, вам нужно иметь четыре дома на участке, на котором вы собираетесь построить отель. Отели покупаются также, как и дома, по той же цене. При воздвижении отеля, четыре дома с этого участка возвращаются в банк. На каждом участке можно построить только один отель.

Продажа недвижимости

Вы можете продать незастроенные участки, железнодорожные вокзалы и коммунальные предприятия любому игроку, заключив с ним частную сделку, на сумму согласованную между вами. Если на продоваемых вами участках имеются дома или отели, то продавать такую недвижимость нельзя. Сперва необходимо продать банку дома и отели стоящие на всех участках этой цветовой группы, а только после этого предлагать сделку другому игроку.

В сделке с обеих сторон для обмена могут быть предложены как участки Недвижимости, так и деньги, и карточки освобождения из тюрьмы. Комбинации обмена

могут быть самыми разнообразными на усмотрение игроков. Если игроку не интересна предложенная сделка, он может отказаться от нее.

Ни дома, ни отели нельзя продавать другим игрокам. Их можно продавать только банку.

При необходимости, для того, чтобы вы могли получить деньги, отели могут быть снова заменены домами. Для этого вам нужно продать отель в банк и получить взамен четыре дома, плюс стоимость самого отеля.

Залог

Если у вас не осталось денег, но могут возникнуть долги, вы можете получить деньги, заложив какую-либо Недвижимость или продать дома или отели. Для того, чтобы заложить недвижимость, необходимо сначала продать все дома и отели, построенные на участках закладываемой цветовой группы. При залоге, вы получаете из банка сумму, равную половине стоимости закладываемого участка. Если позднее вы захотите выкупить заложенную Недвижимость, вам придется выплатить банку ее полную стоимость, плюс 10

Если вы закладываете какую-либо Недвижимость, она попрежнему принадлежит вам. Ни один игрок не вправе выкупить ее вместо вас у банка.

С заложеной Недвижимости нельзя взимать арендную плату, хотя арендная плата попрежнему может поступать к вам за другие объекты Недвижимости той же цветовой группы.

Вы не можете продавать заложенную Недвижимость другим игрокам.

Возможность строить на участках дома появляется только после выкупа всех без исключения участков одной цветовой группы.

Банкротство

Если вы должны банку или другим игрокам больше денег, чем вы можете получить по вашим игровым активам, вас объявляют банкротом, и вы выбываете из игры.

Если вы должны банку, банк получает все ваши деньги и всю вашу Недвижимость. Вернувшаяся в банк недвижимость поступает в свободную продажу. Также в

банк возвращаются карточки освобождения от тюрьмы.

Если вы обанкротились из-за долгов другому игроку, все ваше имущество отправляется в банк. Вернувшаяся в банк недвижимость поступает в свободную продажу. А вашему должнику Банк выплачивает сумму долга.

Так же вы можете стать банкротом, если не успеете выполнить какое-либо игровое действие в отведенное на него время.

Победитель

Последний оставшийся в игре участник является победителем.

2. РАЗРАБОТКА АЛГОРИТМА

2.1. Разработка графического интерфейса пользователя

Для обеспечения минимального порога вхождения основной интерфейс должен более-менее точно копировать реальную доску игры "Монополия".

Основной экран обрамлен 40 стандартными полями игры - улицы, налоги, вокзалы и т.д. Рядом с каждым полем-улицей, ближе к центру экрана, находится прямоугольник цветовой группы, на котором будут располагаться дома и отели при покупке их игроком. Фишки, обозначающие положение игроков, находятся в нижних частях игровых полей. Если пользователь щелкает по полю, ему показывается информация по этому полю - рента для улиц, стоимость налога и т.д.

В оставшейся части экрана отображается список игроков, укрупненно показывается его фишка, имя, тип или состояние - локальный, удаленный, отключен и т.д., баланс. Кроме того, здесь располагаются 3 кнопки - завершения хода, отключения и выхода.

Помимо окна с информацией о полях, следует использовать окна для подключения, для присоединения игрока к игре, аукциона и торговли и различных сообщений.

Итак, внешний вид разработанного интерфейса представлен на рисунке 1. ы

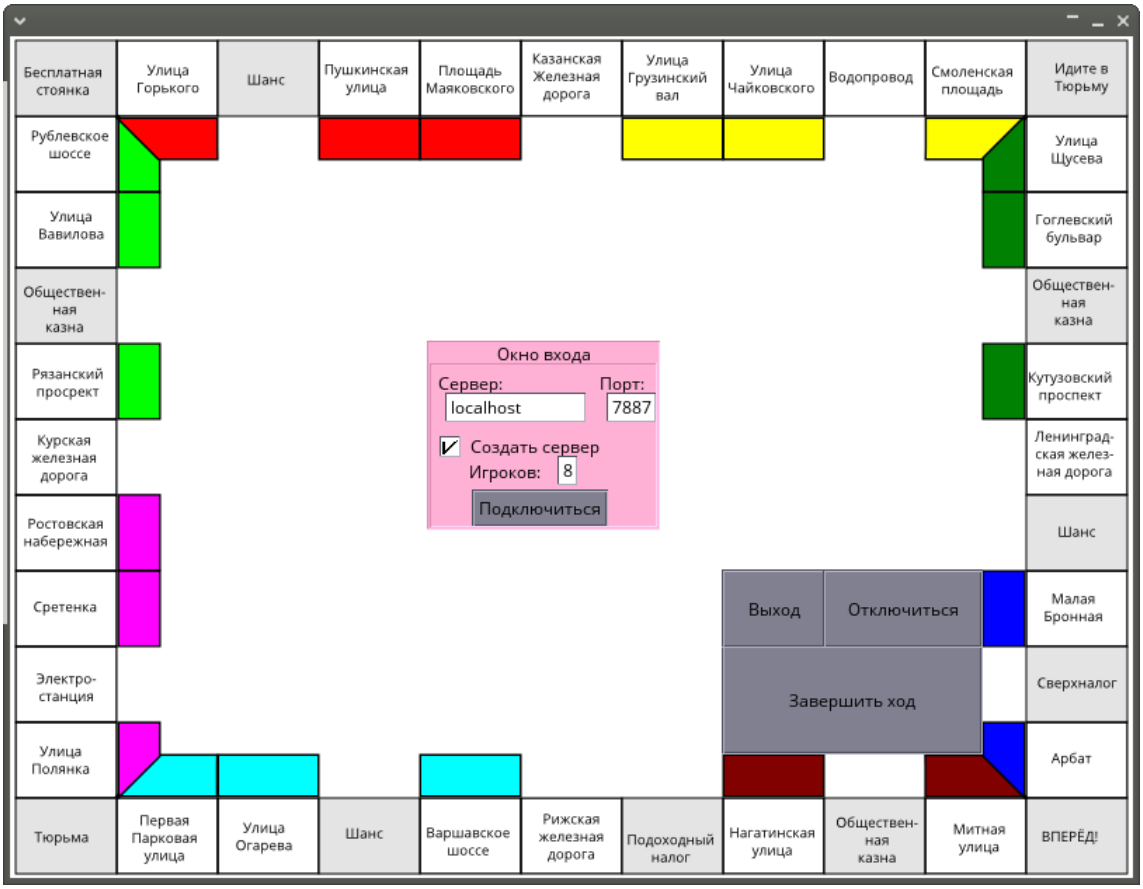

Рисунок 1 – Разработанный интерфейс программы

2.2. Разработка структур данных

Сервер и Клиент обмениваются сообщениями, структура которых имеет следующий формат:

type - тип сообщения(расшифровывается ниже);

names - массив имен игроков;

src,dst - обычно начальное и конечное поле события;

dice1,dice2 - выпавшие кости;

curr_player - текущий игрок;

pnum - номер пакета;

moneys - балансы игроков;

Типы сообщений:

EVENT - игровое событие - переход игрока на новое поле,

DISCONNECT - отключение сервера или клиента,

ACK - подтверждение соединения между клиентом и сервером,

SERVER_FULL - указывает, что сервер переполнен,

UPDATE_PLAYERS - указание клиентам обновить информацию об игроках,

UPDATE_FIELD - указание обновить информацию об игровом поле,

OWNED - говорит, что игрок покупает текущее поле,

NOT_OWNED - говорит, что игрок не покупает текущее поле,,

LOOSER - указание сервера на проигравшего игрока,

WINNER - указание сервера на выигравшего игрока,

AUCTION - сообщение об установке цены на аукционах и торгах,

BUY_HOUSE - запрос на покупку отеля,

BUY_HOTEL - запрос на продажу отеля,

BUY_FIELD - запрос на покупку поля,

SELL_FIELD - запрос на продажу поля.

2.3. Разработка структуры алгоритма

Программу можно разбить на следующие части:

1) Сервер - выполняет функции Банкира и Аукционера, а также кидает кости. Сам сервер может быть разбит на части: подпрограмма service_thread,которая управляет подключением игроков к серверу; подпрограмма main_server_thread, которая имитирует кидание костей и подсчет баланса игроков; подпрограмма server_thread, которая получает и обрабатывает сообщения конкретного игрока; подпрограммы auct_thread и trade_thread, которые позволяет проводить аукционы и торговлю между игроками.

2) Клиент client_thread - получает сообщения от сервера, обновляет игровые данные и запрашивает отрисовку.

3) Части приложения, которые отрисовывают данные, получаемые клиентом.

3. РАЗРАБОТКА ПРОГРАММЫ

3.1. Описание переменных и структур данных

Сервер и Клиент обмениваются сообщениями типа PacketData, структура которых имеет следующий формат:

PacketType type - тип сообщения(расшифровывается ниже);

char names[168] - массив имен игроков;

Sint32 src,dst - обычно начальное и конечное поле события;

Sint32 dice1,dice2 - выпавшие кости;

Sint32 curr_player - текущий игрок;

Uint32 pnum - номер пакета;

Uint32 moneys[8] - балансы игроков;

Типы сообщений:

EVENT - игровое событие - переход игрока на новое поле,

DISCONNECT - отключение сервера или клиента,

ACK - подтверждение соединения между клиентом и сервером,

SERVER_FULL - указывает, что сервер переполнен,

UPDATE_PLAYERS - указание клиентам обновить информацию об игроках,

UPDATE_FIELD - указание обновить информацию об игровом поле,

OWNED - говорит, что игрок покупает текущее поле,

NOT_OWNED - говорит, что игрок не покупает текущее поле,,

LOOSER - указание сервера на проигравшего игрока,

WINNER - указание сервера на выигравшего игрока,

AUCTION - сообщение об установке цены на аукционах и торгах,

BUY_HOUSE - запрос на покупку отеля,

BUY_HOTEL - запрос на продажу отеля,

BUY_FIELD - запрос на покупку поля,

SELL_FIELD - запрос на продажу поля.

3.2. Описание функций

1. main_server_thread(void *data)

Подпрограмма, которая имитирует кидание костей и подсчет баланса игроков;

2. server_thread(void *data)

Подпрограмма, которая получает и обрабатывает сообщения конкретного игрока;

3. service_thread(void *data)

Подпрограмма, которая управляет подключением игроков к серверу;

4. auct_thread(void *data)

Подпрограмма, которая позволяет проводить аукционы

5. trade_thread(void *data)

Подпрограмма, которая организует торговлю между игроками.

6. client_thread(void *data)

Подпрограмма, которая получает сообщения от сервера, обновляет игровые данные и запрашивает отрисовку.

7. redraw()

Подпрограмма, которая перерисовывает экран приложения.

8. redraw_board()

Подпрограмма, которая перерисовывает игровое поле.

4. ИНСТРУКЦИЯ ПОЛЬЗОВАТЕЛЮ

Данная программа позволяет сыграть партию игры "Монополия"с игроками за локальным компьютером или находящимися на удаленных компьютерах.

Для начала подключитесь к серверу, введя его символический или цифровой адрес и порт. Вы также можете создать сервер на указанном порту для указанного количества игроков.

После добавьте пользователей, сидящих за вашим компьютером к списку игроков сервера. Когда все места для игроков будут заняты, сервер начнет игру.

Вы можете руководствоваться стандартными правилами "Монополии". После того, как сервер передвигает вас на следующее поле, вы можете покупать улицы, воказалы, предприятия,дома и т.д. При проведении торгов и аукциона используется таймер на 9 секунд. После того, как первый игрок устанавливает цену, таймер активизируется. Когда таймер обнуляется, аукцион или торги заканчиваются. Каждое обновление цены игроками сбрасывает таймер обратно на 9 секунд.

Вы можете нажать Ctrl+F, чтобы переключаться в полноэкранный режим и обратно.

Для выхода закройте окно, нажмите соответствующую кнопку на экране или клавишу Esc на клавиатуре.

5. ТЕСТОВЫЙ ПРИМЕР

Ниже на рисунке 2 представлен пример работы программы-игры "Монополия".

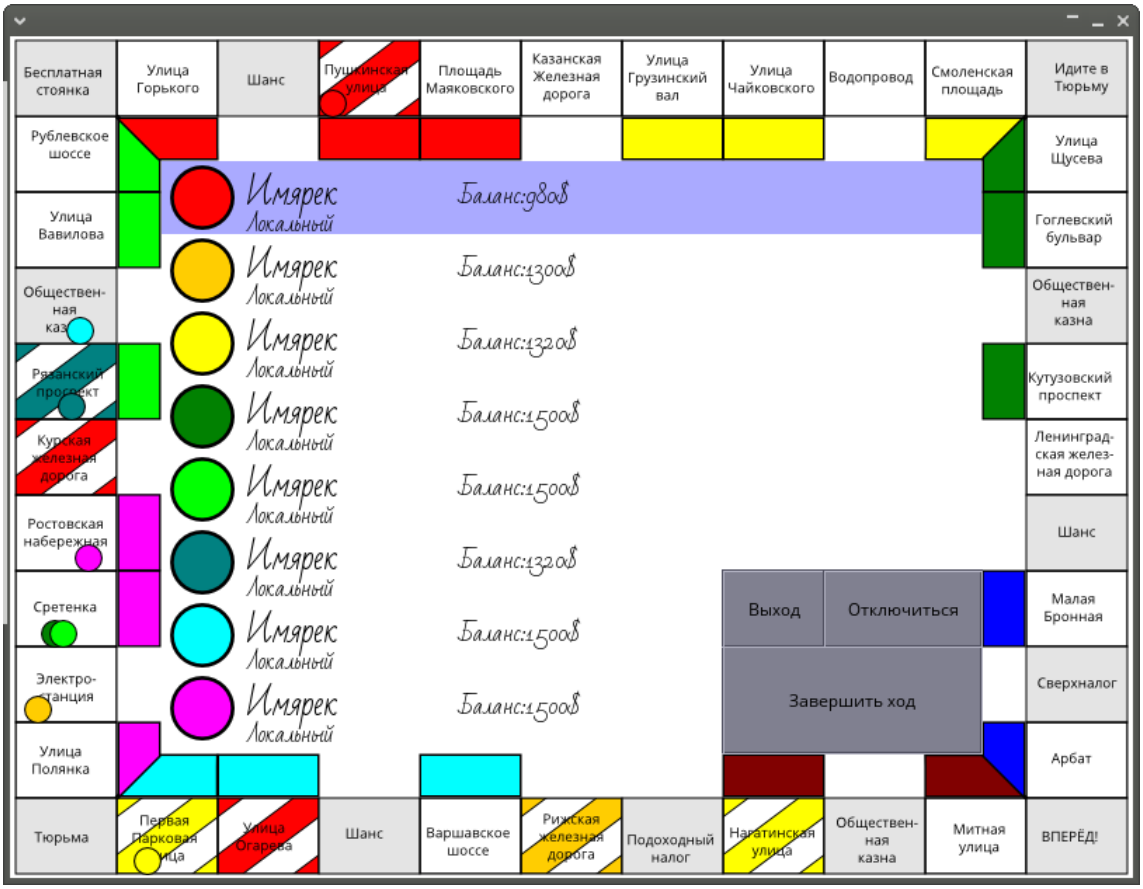

Рисунок 2 – Пример работы программы-игры "Монополия"

ЗАКЛЮЧЕНИЕ

Программирование сетевых приложений - одна из самых динамично развивающихся областей программирования.

Создание сетевой компьютерной версии игры "Монополия"позволяет понять общие принципы сетевого программирования, а также принципы создания компьютерных игр и сложных графических приложений.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1) http://www.libsdl.org

2) http://guichan.sourceforge.net

3) http://ru.wikipedia.org

4) http://en.wikipedia.org

ПРИЛОЖЕНИЕ

Далее приводится общая часть сетевой программы-игры "Монополия".

```
#ifndef CORE_HPP_INCLUDED

#define CORE_HPP_INCLUDED
#include <SDL.h>

namespace core{
    const int JAIL_LOCATION=10;
    enum FieldType {STREET,RAILWAY,PLANT,CCHEST,CHANCE,FREEPARKING,START,JAIL,
        TO_JAIL,FEE};
    enum JailState {FREE=0,JAIL_1,JAIL_2,JAIL_3};
    enum PlayerType {UNDEF,LOCAL,REMOTE,BANKROT};
    typedef struct {
        char name[21];
        Sint64 money;
        int location;
        int jailed;
        PlayerType type;
    } Player;
    typedef struct{
        FieldType type;
        int houses_num;
        int price;
        int rent[6];
        int mortgage_val;
        int house_cost;
        int hotel_cost;
        bool mortgaged;
        int owner;
    } Field;
    enum PacketType {EVENT,DISCONNECT,ACK,PLAYER_QUERY,SERVER_FULL,UPDATE_PLAYERS,
    UPDATE_FIELD,OWNED,NOT_OWNED,LOOSER,WINNER,AUCTION,BUY_HOUSE,BUY_HOTEL,BUY_FIELD,
    SELL_FIELD} ;
    typedef struct{

        PacketType type;
        char names[168];
        Sint32 src,dst;
        Sint32 dice1,dice2;
        Sint32 curr_player;
        Uint32 pnum;
        Sint64 moneys[8];
    } PacketData;
    //extern bool running;
    extern Field board[40];
    extern Player player[8];
```

```
    void init(Field board[40],Player players[8]);

    int valid_field(int num);

    void copy_name(const char *s,char name[20]);

}



#endif // CORE_HPP_INCLUDED

#include "core.hpp"
namespace core{
    bool running=true;
    //Field board[40];

    void copy_name(const char *s,char name[20]){
        int i;
        for (i=0;i<20;i++){
            if (!s[i]) break;
            name[i]=s[i];
        }
        name[i]='\0';
    }

    int valid_field(int num){
        return num%40;
    }

    void init(Field board[40],Player players[8]){
        int i;
        for (i=0;i<40;++i){
            switch (i){
                case 0: board[i].type=core::START; break;
                case 2: board[i].type=core::CCHEST; break;
                case 4: board[i].type=core::FEE; board[i].price = 200; break;
                case 5: board[i].type=core::RAILWAY; board[i].price = 200; break;
                case 7: board[i].type=core::CHANCE; break;
                case 10: board[i].type=core::JAIL; break;
                case 12: board[i].type=core::PLANT; board[i].price = 150; break;
                case 15: board[i].type=core::RAILWAY; board[i].price = 200; break;
                case 17: board[i].type=core::CCHEST; break;
                case 20: board[i].type=core::FREEPARKING; break;
                case 22: board[i].type=core::CHANCE; break;
                case 25: board[i].type=core::RAILWAY; board[i].price = 200; break;
                case 28: board[i].type=core::PLANT; board[i].price = 150; break;
                case 30: board[i].type=core::TO_JAIL; break;
                case 33: board[i].type=core::CCHEST; break;
                case 35: board[i].type=core::RAILWAY; board[i].price = 200; break;
                case 36: board[i].type=core::CHANCE; break;
                case 38: board[i].type=core::FEE; board[i].price = 100; break;
                default:
                    board[i].type=core::STREET;
            }
            board[i].owner=-1;
            if (board[i].type==core::STREET){
                printf("/%d/\n",i);
                board[i].mortgaged=false;
                board[i].houses_num=0;
                char filename[30];
                int l=sprintf(filename,"field%d.txt",i);
                filename[l]='\0';
```

```
                FILE* f=fopen(filename,"r");
                fscanf(f,"%d",&board[i].price);
                printf("price:%d\n",board[i].price);
                int j;
                for(j=0;j<6;j++){
                    fscanf(f,"%d",&board[i].rent[j]);
                    printf("rent%d:%d\n",j,board[i].rent[j]);
                }
                fscanf(f,"%d",&board[i].house_cost);
                printf("house_cost:%d\n",board[i].house_cost);
                fscanf(f,"%d",&board[i].hotel_cost);
                printf("hotel_cost:%d\n",board[i].hotel_cost);
                fscanf(f,"%d",&board[i].mortgage_val);
                printf("mortgage_val:%d\n",board[i].mortgage_val);
                fclose(f);

            }

        }
        for(i=0;i<8;++i){
            core::copy_name("Нет игрока\0",players[i].name);
            players[i].money=1500;
            players[i].type=UNDEF;
        }
    }
}
```

Далее приводится серверная часть сетевой программы-игры "Монополия".

```
#ifndef SERVER_HPP_INCLUDED

#define SERVER_HPP_INCLUDED
//#include "core.hpp"
#include <SDL_net.h>
#include <iostream>
#include "server.hpp"
#include "core.hpp"
namespace server{

    typedef struct {
        Uint16 port;
        int pCount;
    } ServerStartData;



    int send_broadcast(core::PacketData pdata);

    void send_to_watchers(core::PacketData pdata);

    int service_thread(void *data);

    int server_thread(void *data);

    int main_server_thread(void *data);


    extern bool running;
    extern core::Field board[40];
    extern core::Player players[9];

}
```

```
#endif // SERVER_HPP_INCLUDED

#include "server.hpp"
namespace server{
    Uint32 stime=1;
    bool running=true;
    core::Field board[40];
    core::Player players[9];

    TCPsocket psocks[9];
    TCPsocket watchers[16];

    int auct_timer;
    int auct_prices[8];

    SDL_cond *startgame=SDL_CreateCond(),*nextturn=SDL_CreateCond();

    SDL_mutex *startgamem=SDL_CreateMutex(),*nextturnm=SDL_CreateMutex();
    SDL_mutex *socketsm=SDL_CreateMutex(),*auctm=SDL_CreateMutex();

    SDLNet_SocketSet clients=SDLNet_AllocSocketSet(2*8);
    int clients_count=0;int curr_player=0;
    ServerStartData param;

    int send_broadcast(core::PacketData pdata){
    int i,failed=0;
    SDL_mutexP(socketsm);
    for (i=0;i<clients_count;i++){
            if(psocks[i]){
                if(SDLNet_TCP_Send(psocks[i],&pdata,sizeof(pdata))<sizeof(pdata))
                    {failed++;
                    printf("failed\n");}

            } else failed++;
        }
    SDL_mutexV(socketsm);
    return failed;
    }

    void send_to_watchers(core::PacketData pdata){
    int i;
    for (i=0;i<16;i++){
            if(watchers[i]){
                SDLNet_TCP_Send(watchers[i],&pdata,sizeof(pdata));
            }
    }
    }
    int auct_thread(void *data){
        core::PacketData pdata;
        int i;
        auct_timer=6;
        for(i=0;i<clients_count;i++){
            auct_prices[i]=0;
        }
        while (server::running){
            for (i=0;i<clients_count;i++){
                if(auct_prices[i]!=0){
                    SDL_mutexP(auctm);
                    auct_timer--;
                    SDL_mutexP(socketsm);
                    pdata.time=stime;
                    stime++;
```

```
                SDL_mutexV(socketsm);
                pdata.type=core::AUCTION;
                pdata.dst=auct_timer;
                pdata.curr_player=clients_count;
                for (i=0;i<clients_count;i++){
                    pdata.money[i]=auct_prices[i];
                }
                send_broadcast(pdata);
                printf("auction server%d\n",pdata.dst);
                SDL_mutexV(auctm);
                break;
            }
        }
        SDL_Delay(1000);

        if(auct_timer==0)
            break;
    }
    if(server::running){
        int max=0,maxi=0;
        for (i=0;i<clients_count;i++){
            if(auct_prices[i]>max){
                max=auct_prices[i];
                maxi=i;
            }
        }

        board[players[curr_player].location].owner=maxi;

        SDL_mutexP(socketsm);
        pdata.time=stime;
        stime++;
        SDL_mutexV(socketsm);
        pdata.type=core::OWNED;
        pdata.dst=players[curr_player].location;
        pdata.curr_player=maxi;
        send_broadcast(pdata);

        players[maxi].money-=auct_prices[i];

        SDL_mutexP(socketsm);
        pdata.time=stime;
        stime++;
        SDL_mutexV(socketsm);
        pdata.type=core::UPDATE_PLAYERS;
        pdata.dst=param.pCount;
        pdata.curr_player=clients_count;
        for (i=0;i<clients_count;i++){
            core::copy_name(players[i].name,pdata.name+21*i);
            pdata.money[i]=players[i].money;
        }
        send_broadcast(pdata);
    }
    return 0;
}
int service_thread(void *data){
    TCPsocket socket=*((TCPsocket*)data),newsocket;
    core::PacketData pdata;
    //int turn=1;
    int i;
    for (i=0;i<8;++i){
        psocks[i]=NULL;
    }
    for (i=0;i<16;++i){
```

```
        watchers[i]=NULL;
    }

    while (server::running){
        newsocket=SDLNet_TCP_Accept(socket);
        while (server::running&&!newsocket){
            SDL_Delay(500);
            newsocket=SDLNet_TCP_Accept(socket);
        }
        printf("%d\n",param.pCount);
        if(clients_count<param.pCount){
            if(server::running){
                for (i=0;i<16;++i){
                    if(!watchers[i]){
                        SDLNet_TCP_AddSocket(clients,newsocket);
                        watchers[i]=newsocket;
                        printf("New client!\n");

                        int j;
                        SDL_mutexP(socketsm);
                        pdata.time=stime;
                        stime++;
                        pdata.type=core::UPDATE_PLAYERS;
                        pdata.dst=param.pCount;
                        pdata.curr_player=clients_count;
                        for (j=0;j<clients_count;j++){
                            core::copy_name(players[j].name,pdata.name+21*j);
                            pdata.money[j]=players[j].money;
                        }
                        SDLNet_TCP_Send(newsocket,&pdata,sizeof(pdata));
                        SDL_mutexV(socketsm);
                        break;
                    }
                }
                if (i==16){
                     pdata.type=core::SERVER_FULL;
                     printf("Server full!\n");
                     SDLNet_TCP_Send(newsocket,&pdata,sizeof(pdata));
                     SDLNet_TCP_Close(newsocket);
                };
                 //++curr_player;
            }
            int i;
            if (SDLNet_CheckSockets(clients,500)!=-1){

                for (i=0;i<16;++i){
                    if (watchers[i]&&SDLNet_SocketReady(watchers[i])){

                        if(server::running&&SDLNet_TCP_Recv(watchers[i],
                            &pdata,sizeof(pdata))>0)
                        {
                            core::copy_name(pdata.name,
                                    players[clients_count].name);
                            printf("Accepted %s!\n",pdata.name);
                            pdata.curr_player=clients_count;
                            pdata.type=core::ACK;
                            psocks[clients_count]=watchers[i];
                            watchers[i]=NULL;
                            printf("1!\n");
                            SDLNet_TCP_Send(psocks[clients_count],
                                            &pdata,sizeof(pdata));

                            int* x=new int(clients_count);
```

```
                            SDL_CreateThread(server_thread,x);
                            clients_count++;
                            int j;
                            SDL_mutexP(socketsm);
                            pdata.time=stime;
                            stime++;
                            SDL_mutexV(socketsm);
                            pdata.type=core::UPDATE_PLAYERS;
                            pdata.dst=param.pCount;
                            pdata.curr_player=clients_count;
                            for (j=0;j<clients_count;j++){
                                core::copy_name(players[j].name,pdata.name+21*j);
                                pdata.money[j]=players[j].money;
                            }
                            send_broadcast(pdata);
                            send_to_watchers(pdata);
                            printf("2!%ld\n",sizeof(pdata));
                            if (clients_count==param.pCount){
                                SDL_CondSignal(startgame);
                            }
                        } else {
                            SDLNet_TCP_DelSocket(clients,watchers[i]);
                            SDLNet_TCP_Close(watchers[i]);
                            watchers[i]=NULL;
                        }

                    }
                }
            }

        } else {
            pdata.type=core::SERVER_FULL;
            for (i=0;i<16;++i){

                if (watchers[i]){
                    SDLNet_TCP_Send(watchers[i],&pdata,sizeof(pdata));
                    SDLNet_TCP_DelSocket(clients,watchers[i]);
                    SDLNet_TCP_Close(watchers[i]);
                    watchers[i]=NULL;
                }
            }
            if (newsocket){
                SDLNet_TCP_Send(newsocket,&pdata,sizeof(pdata));
                SDLNet_TCP_DelSocket(clients,newsocket);
                SDLNet_TCP_Close(newsocket);
                newsocket=NULL;
            }
        }
    }
    pdata.type=core::DISCONNECT;
    for (i=0;i<16;++i){
        if (watchers[i]){
            SDLNet_TCP_Send(watchers[i],&pdata,sizeof(pdata));
            SDLNet_TCP_DelSocket(clients,watchers[i]);
            SDLNet_TCP_Close(watchers[i]);
            watchers[i]=NULL;
        }
    }
    if (newsocket){
        SDLNet_TCP_Send(newsocket,&pdata,sizeof(pdata));
        SDLNet_TCP_DelSocket(clients,newsocket);
        SDLNet_TCP_Close(newsocket);
        newsocket=NULL;
    }
```

```
    SDL_CondSignal(startgame);
    SDL_CondSignal(nextturn);
    printf("Service thread terminated\n");
    return 0;
}
int server_thread(void *data){
    bool running=true;
    int player=*((int*)data);
    core::PacketData pdata;
    printf("Server thread %d started!\n",player);
    while (server::running&&running){
        while(server::running&&running&&
            (SDLNet_CheckSockets(clients,500)==-1||
                !SDLNet_SocketReady(psocks[player])));
            //SDL_Delay(500);
           // printf("Nope\n");
          //printf("Socket not ready\n");
        if (running) printf("P...\n");
        if(server::running&&running){
        if(SDLNet_TCP_Recv(psocks[player],&pdata,sizeof(pdata))==sizeof(pdata)){
            printf("%d\n",pdata.type);
            switch (pdata.type){
                case core::ACK:
                    if (player==curr_player){
                        printf("Acknowledged!\n");
                        SDL_CondSignal(nextturn);
                    }

                    break;
                case core::AUCTION:
                    auct_prices[player]=pdata.money[0];
                    SDL_mutexP(auctm);
                    auct_timer=10;
                    SDL_mutexV(auctm);
                    break;
                case core::OWNED:
                    if (player==curr_player&&
                       (board[players[player].location].type==core::STREET||
                       board[players[player].location].type==core::RAILWAY||
                       board[players[player].location].type==core::PLANT)
                        &&board[players[player].location].owner==-1){

                        printf("OWNED:%d\n",
                            server::board[players[player].location].price);
                        players[player].money-=
                            server::board[players[player].location].price;
                        server::board[players[player].location].owner=player;
                        SDL_mutexP(socketsm);
                        pdata.time=stime;
                        stime++;
                        SDL_mutexV(socketsm);
                        pdata.type=core::OWNED;
                        pdata.dst=players[player].location;
                        pdata.curr_player=player;
                        send_broadcast(pdata);

                        int j;
                        SDL_mutexP(socketsm);
                        pdata.time=stime;
                        stime++;
                        SDL_mutexV(socketsm);
                        pdata.type=core::UPDATE_PLAYERS;
                        pdata.dst=param.pCount;
                        pdata.curr_player=clients_count;
```

```
                        for (j=0;j<clients_count;j++){
                            core::copy_name(players[j].name,pdata.name+21*j);
                            pdata.money[j]=players[j].money;
                        }
                        send_broadcast(pdata);

                    }

                    break;
                case core::DISCONNECT:
                    printf("Client disconnected!\n");
                    SDL_mutexP(socketsm);
                    SDLNet_TCP_DelSocket(clients,psocks[player]);
                    SDLNet_TCP_Close(psocks[player]);
                    psocks[player]=NULL;
                    SDL_mutexV(socketsm);
                    running=false;
                    break;
                case core::NOT_OWNED:
                    SDL_CreateThread(auct_thread,NULL);
                    break;
                default:
                    printf("server: Unknown packet type!\n");
            }
        } else {
            printf("Error!");
            running=false;
        }
        }
    }
    if (psocks[player]){
        pdata.type=core::DISCONNECT;
        SDLNet_TCP_Send(psocks[player],&pdata,sizeof(pdata));
        SDL_mutexP(socketsm);
        SDLNet_TCP_DelSocket(clients,psocks[player]);
        SDLNet_TCP_Close(psocks[player]);
        psocks[player]=NULL;
        SDL_mutexV(socketsm);
    }
    SDL_CondSignal(startgame);
    SDL_CondSignal(nextturn);
    printf("Server thread terminated\n");
    return 0;
}
int main_server_thread(void *data){
    core::PacketData pdata;
    param=*((ServerStartData*)data);
    core::init(server::board,server::players);
    int i;
    SDL_mutexP(startgamem);
    SDL_mutexP(nextturnm);
    srand(time(NULL));
    IPaddress address;
    SDLNet_ResolveHost(&address,INADDR_ANY,param.port);
    TCPsocket socket=SDLNet_TCP_Open(&address);
    if (socket){
        printf("Server started!\n");
        SDL_CreateThread(service_thread,&socket);
        SDL_CondWait(startgame,startgamem);

        printf("Game started!\n");
        int players_failed=0;
        while(server::running){
            if (psocks[curr_player]){
```

```
SDL_mutexP(socketsm);
pdata.time=stime;
stime++;
SDL_mutexV(socketsm);
pdata.type=core::EVENT;

pdata.curr_player=curr_player;
if (players_failed==param.pCount-1){
    pdata.type=core::WINNER;
    printf("Winner is founded\n");
    send_broadcast(pdata);
    SDL_mutexP(socketsm);
    SDLNet_TCP_DelSocket(clients,psocks[curr_player]);
    SDLNet_TCP_Close(psocks[curr_player]);
    psocks[curr_player]=NULL;
    SDL_mutexV(socketsm);
    server::running=false;
    break;
}

printf("cp-%d\n",curr_player);
pdata.dice1=rand()%6+1;
pdata.dice2=rand()%6+1;
if (!players[curr_player].jailed){
    pdata.src=players[curr_player].location;
    pdata.dst=core::valid_field(pdata.src+pdata.dice1+
        pdata.dice2);
} else {
    pdata.src=core::JAIL_LOCATION;
    pdata.dst=core::JAIL_LOCATION;
    players[curr_player].jailed--;
}

printf("SERVER:%d\n",pdata.dst);
if (players[curr_player].location>pdata.dst){
    players[curr_player].money+=200;
}
players[curr_player].location=pdata.dst;
if (board[players[curr_player].location].type==core::TO_JAIL){
    players[curr_player].location=core::JAIL_LOCATION;
    players[curr_player].jailed=core::JAIL_3;
}
printf("%d\n",board[players[curr_player].location].owner);
if(board[players[curr_player].location].owner!=-1){
        players[curr_player].money-=server::board[players[
                curr_player].location].rent[board[players[
                curr_player].location].houses_num];
        players[board[players[curr_player].location].owner].money+=
        board[players[curr_player].location].rent[board[
            players[curr_player].location].houses_num];
        printf("Payed\n");
}
switch (board[players[curr_player].location].type){
    case core::FEE:
        players[curr_player].money-=server::board[players[
            curr_player].location].price;
        break;
    default:;
}
if(send_broadcast(pdata)==param.pCount){
    printf("All clients are dead!\n");
    server::running=false;
    break;
}
```

```
                    int j;
                    SDL_mutexP(socketsm);
                    pdata.time=stime;
                    stime++;
                    SDL_mutexV(socketsm);
                    pdata.type=core::UPDATE_PLAYERS;
                    pdata.dst=param.pCount;
                    pdata.curr_player=clients_count;
                    for (j=0;j<clients_count;j++){
                        core::copy_name(players[j].name,pdata.name+21*j);
                        pdata.money[j]=players[j].money;
                    }
                    if(send_broadcast(pdata)==param.pCount){
                        printf("All clients are dead!\n");
                        server::running=false;
                        break;
                    }
                    SDL_CondWait(nextturn,nextturnm);
                    printf("money:%ld\n",players[curr_player].money);
                    if (players[curr_player].money<0){
                        pdata.type=core::LOOSER;
                        send_broadcast(pdata);
                        SDL_mutexP(socketsm);
                        SDLNet_TCP_DelSocket(clients,psocks[curr_player]);
                        SDLNet_TCP_Close(psocks[curr_player]);
                        psocks[curr_player]=NULL;
                        SDL_mutexV(socketsm);
                        players_failed+=1;
                    } else players_failed=0;
                } else {players_failed+=1;}
                if (players_failed==param.pCount)
                    server::running=false;
                curr_player++;
                if(curr_player>=param.pCount)
                    curr_player%=param.pCount;
            };
        } else {
            printf("Server not created\n");
        }
        printf("Server exited!\n");
        return 0;
    }
}
```

Далее приводится клиентская часть сетевой программы-игры "Монополия".

```
#ifndef CLIENT_HPP_INCLUDED

#define CLIENT_HPP_INCLUDED

namespace client{
    int client_thread(void *data);
    typedef struct {
        const char *host;
        Uint16 port;
        TCPsocket *socket;
    } ClientStartData;
    extern core::Field board[40];
    extern core::Player players[8];
    extern TCPsocket psocks[8];
    extern int players_count,current_player;
    extern bool running;
}
```

```
#endif // CLIENT_HPP_INCLUDED

#include "sdl.hpp"
#include "client.hpp"
#include <SDL_net.h>
namespace client{
    bool running=true;
    core::Field board[40];
    core::Player players[8];
    TCPsocket psocks[8];
    SDL_mutex *drawlock=SDL_CreateMutex();
    int current_player=-1,players_count=0;
    Uint32 stime=0;

    void init(){

    }

    void move_player(int player,int src,int dst,int step){
        while(client::running&&src!=dst){
            src=core::valid_field(src+step);
            players[player].location=src;
            SDL_mutexP(sdl::lock);
            sdl::redraw();
            SDL_mutexV(sdl::lock);
            SDL_Delay(75);

        }
    }

    int client_thread(void *data){
        printf("%p\n",data);
        ClientStartData param=*((ClientStartData*)data);
         printf("%p\n",data);
        srand(time(NULL));
         printf("%p\n",data);
        //SDL_Delay(rand()%300+2500);
        IPaddress server;
        printf("host resolving:%s;%u\n",param.host,param.port);
        SDLNet_ResolveHost(&server,param.host,param.port);
        printf("host resolved:%ud;%ud\n",server.host,server.port);
        TCPsocket socket=SDLNet_TCP_Open(&server);
        bool running=true;
        while(client::running&&!socket){
            printf("Connection error\n");
            SDL_Delay(rand()%100+400);
            socket=SDLNet_TCP_Open(&server);
        }
        *param.socket=socket;
        if (!client::running)
            return 0;
        printf("Connected!\n");
        core::PacketData pdata;
        //SDLNet_TCP_Send(socket,&pdata,sizeof(pdata));
        int dst=0,src=0,player;
        SDLNet_SocketSet sock=SDLNet_AllocSocketSet(1);
        SDLNet_TCP_AddSocket(sock,socket);
        int i;
        while(client::running&&running){
            while(client::running&&running&&(SDLNet_CheckSockets(sock,500)==-1||
```

```
      !SDLNet_SocketReady(socket)));
    if(client::running&&running&&SDLNet_TCP_Recv(socket,&pdata,
                                                 sizeof(pdata))>=1){
        switch (pdata.type){
            case core::ACK:
                player=pdata.curr_player;
                players[player].type=core::LOCAL;
                printf("Socket saved for %d\n",player);
                psocks[player]=socket;
                break;
            case core::WINNER:
                if (player==pdata.curr_player){
                    printf("I'm winner!\n");
                    running=false;
                }
                break;
            case core::LOOSER:
                if (player==pdata.curr_player){
                    printf("I'm looser!\n");
                    running=false;
                }

                players[pdata.curr_player].type=core::BANKROT;
                break;
            case core::AUCTION:
                SDL_mutexP(drawlock);
                if (stime<pdata.time){
                    printf("update%d\n",pdata.dst);
                    stime=pdata.time;
                    SDL_mutexV(drawlock);
                    widgets::awindow->timer=pdata.dst;
                    for (i=0;i<pdata.curr_player;i++){
                        widgets::awindow->price[i]=pdata.money[i];
                    }
                    widgets::awindow->update();
                } else printf("not updated");

                break;

            case core::DISCONNECT:
            case core::SERVER_FULL:

                printf("Server disconnected!\n");
                running=false;
                break;
            case core::UPDATE_PLAYERS:

                printf("Players update!\n");
                SDL_mutexP(drawlock);
                printf("%d,%d\n",stime,pdata.time);
                if (stime<pdata.time){
                    stime=pdata.time;
                    printf("Updated!%ld\n",sizeof(pdata));
                    for (i=0;i<pdata.curr_player;i++){
                        core::copy_name(pdata.name+21*i,players[i].name);
                        players[i].money=pdata.money[i];
                        if (players[i].type==core::UNDEF){
                            if (psocks[i])
                                players[i].type=core::LOCAL;
                            else
                                players[i].type=core::REMOTE;
                        }
                    }
                    players_count=pdata.dst;
```

```
                SDL_mutexP(sdl::lock);
                if (pdata.curr_player==players_count)
                    widgets::pwindow->setVisible(false);
                sdl::redraw();
                SDL_mutexV(sdl::lock);

            } else {
                //SDL_mutexV(drawlock);
                //printf("Not updated!\n");
            }
            SDL_mutexV(drawlock);
            break;
        case core::OWNED:
            SDL_mutexP(sdl::lock);
            if (pdata.time>stime){
                stime=pdata.time;
                board[pdata.dst].owner=pdata.curr_player;
                sdl::redraw_board();
                sdl::redraw();
            }
            SDL_mutexV(sdl::lock);
            break;
        case core::EVENT:
            current_player=pdata.curr_player;
            //printf("Packet received player %d!\n",player);
            dst=pdata.dst;src=pdata.src;
            //printf("%d\n",SDL_SemValue(drawlock));
            SDL_mutexP(drawlock);
            if (pdata.time>stime){
                stime=pdata.time;
                printf("%d\n",pdata.time);
                SDL_mutexV(drawlock);
                //printf("%d\n",SDL_SemValue(drawlock));
                printf("I paint!%d\n",player);
                current_player=pdata.curr_player;
                printf("CLIENT:%d\n",pdata.dst);
                if (src==core::JAIL_LOCATION&&dst==core::JAIL_LOCATION)
                    players[current_player].jailed=
                            players[current_player].jailed-1;
                else{
                    move_player(current_player,src,dst,1);

                    if (board[dst].type==core::TO_JAIL){
                        int step;
                        players[current_player].jailed=core::JAIL_3;
                        if (core::JAIL_LOCATION<dst)
                            step=-1;
                        else step=1;
                        src=dst;
                        dst=core::JAIL_LOCATION;
                        move_player(current_player,src,dst,step);

                    }
                }
                //SDL_SemPost(drawlock);
                if (players[pdata.curr_player].type==core::LOCAL&&
                    (board[pdata.dst].type==core::STREET||
                     board[pdata.dst].type==core::RAILWAY||
                     board[pdata.dst].type==core::PLANT)
                    &&board[pdata.dst].owner==-1){
                        printf("location: %d\n",pdata.dst);
                    widgets::bwindow->show(players[
                                pdata.curr_player].location);
```

```
                            SDL_mutexP(sdl::lock);
                            sdl::redraw();
                            SDL_mutexV(sdl::lock);
                        }
                    } else SDL_mutexV(drawlock);

                    break;
                default:
                    printf("client: Unknown packet type!\n");
            }
        } else {
            running=false;
        }
    }
    if (socket){
        pdata.type=core::DISCONNECT;
        SDLNet_TCP_Send(socket,&pdata,sizeof(pdata));
        SDLNet_TCP_Close(socket);
    }
    printf("Client exited!\n");
    return 0;
  }
}
```

Далее приводится часть сетевой программы-игры "Монополия которая занимается отрисовкой экрана.

```
#ifndef SDL_HPP_INCLUDED

#define SDL_HPP_INCLUDED
/*

 * Code that sets up an SDL application with Guichan using the

 * Guichan SDL back end.

 */

#include "server.hpp"

#include <guichan.hpp>

#include <guichan/sdl.hpp>

#include <stdlib.h>
#include <time.h>
#include <iostream>
#include <SDL_net.h>
#include <stdio.h>
#include <stdlib.h>
#include <SDL_ttf.h>
#include <SDL_image.h>
#include <SDL_rwops.h>
#include "widgets.hpp"
namespace sdl{
    const int fieldw=72;
    const int fieldh=54;
    const int boardw=800;
    const int boardh=600;
    const int tokenw=19;
    const int tokenh=19;
    const int groupmarg=32;
    extern int marginl,marginr,margint,marginb;
```

```
    typedef struct {
        SDL_Surface *owned,*token,*tokenbig;
        TCPsocket socket;
        SDL_sem *drawing;
    } PlayerView;


    extern gcn::Gui* gui;
    void init();
    void halt();
    void run();
    int client_thread(void *data);


    void redraw();

    void redraw_board();
    //extern int current_player,players_count;
    extern SDL_Thread *server;
    extern SDL_mutex *lock;
    extern bool running;
    //extern SDL_mutex *drawlock;
}

#endif

#include  "sdl.hpp"
#include "client.hpp"
namespace sdl

{
    bool running=true;
    int marginl=4,marginr=4,margint=3,marginb=3;
    bool fullscreen=false;
    gcn::Gui* gui;
    SDL_Thread *server=NULL;
    PlayerView Players[8];
    //core::Field Board[40];

    SDL_Surface* screen,*noowned,*boardsurf;

    TTF_Font *hugeFont,*normFont;

    // All back ends contain objects to make Guichan work on a

    // specific target — in this case SDL — and they are a Graphics

    // object to make Guichan able to draw itself using SDL, an

    // input objec to make Guichan able to get user input using SDL

    // and an ImageLoader object to make Guichan able to load images

    // using SDL.

    gcn::SDLGraphics* graphics;

    gcn::SDLInput* input;

    gcn::SDLImageLoader* imageLoader;

    SDL_mutex *lock=SDL_CreateMutex();

    SDL_Rect get_field_rect(int num);
```

```
SDL_Rect get_token_rect(int,int);
SDL_Surface* bmp;

/**

 * Initialises the SDL application. This function creates the global

 * Gui object that can be populated by various examples.

 */

 SDL_Rect get_field_rect(int num){
    SDL_Rect rect;
    int line_num=num/10,field_num=num%10;
    rect.w=fieldw;rect.h=fieldh;
    switch(line_num){
        case 2:
           rect.x=fieldw*field_num+marginl;
           rect.y=margint;
           break;
        case 3:
           rect.y=fieldh*field_num+margint;
           rect.x=boardw-marginr-fieldw;
           break;
        case 0:
           rect.x=boardw-fieldw*(field_num+1)-marginr;
           rect.y=boardh-marginb-fieldh;
           break;
        case 1:
           rect.y=boardh-fieldh*(field_num+1)-marginb;
           rect.x=marginl;
           break;
    }
    return rect;
}
SDL_Rect get_token_rect(int field_num,int num){
    SDL_Rect field=get_field_rect(field_num);
    field.y+=fieldh-tokenh;
    field.x+=((fieldw-tokenw)/8)*num;
    return field;
}
SDL_Rect get_player_rect(int num){
    SDL_Rect player;
    player.x=marginl+fieldw+groupmarg;
    player.w=boardw-marginr-groupmarg-fieldw-player.x;
    player.y=margint+fieldh+groupmarg;
    player.h=(boardh-marginb-groupmarg-fieldh-player.y)/8;
    player.y+=player.h*num;
    return player;
}
void redraw_board();

void init()

{

    // We simply initialise SDL as we would do with any SDL application.

    SDL_Init(SDL_INIT_VIDEO);

    screen = SDL_SetVideoMode(boardw, boardh, 32, SDL_HWSURFACE|SDL_DOUBLEBUF);


    // We want unicode for the SDLInput object to function properly.
```

```
if(SDLNet_Init()==-1)
{
    std::cout<<"SDLNet_Init: "<<SDLNet_GetError()<<"\n";
    exit(2);
}
IMG_Init(IMG_INIT_PNG);
TTF_Init();
hugeFont=TTF_OpenFont("BadScript-Regular.ttf",23);
normFont=TTF_OpenFont("BadScript-Regular.ttf",15);
SDL_EnableUNICODE(1);

// We also want to enable key repeat.

SDL_EnableKeyRepeat(SDL_DEFAULT_REPEAT_DELAY, SDL_DEFAULT_REPEAT_INTERVAL);


// Now it's time to initialise the Guichan SDL back end.


imageLoader = new gcn::SDLImageLoader();

// The ImageLoader Guichan should use needs to be passed to the Image object

// using a static function.

gcn::Image::setImageLoader(imageLoader);

graphics = new gcn::SDLGraphics();

// The Graphics object needs a target to draw to, in this case it's the

// screen surface, but any surface will do, it doesn't have to be the screen.

graphics->setTarget(screen);

input = new gcn::SDLInput();


// Now we create the Gui object to be used with this SDL application.

gui = new gcn::Gui();

// The Gui object needs a Graphics to be able to draw itself and an Input

// object to be able to check for user input. In this case we provide the

// Gui object with SDL implementations of these objects hence making Guichan

// able to utilise SDL.

gui->setGraphics(graphics);

gui->setInput(input);
bmp=IMG_Load("board.png");
//SDL_SetAlpha(bmp,SDL_SRCALPHA,SDL_ALPHA_TRANSPARENT);
boardsurf=SDL_CreateRGBSurface(SDL_SRCALPHA,boardw,boardh,32,
                        screen->format->Rmask,screen->format->Gmask,
                        screen->format->Bmask,screen->format->Amask);
printf("%d,%d\n",boardsurf->clip_rect.w,boardsurf->clip_rect.h);
bmp=IMG_Load("board.png");
```

```
    int i; char name[50];
    std::cout<<"Loading players data!\n";
    for (i=0;i<8;++i){
        sprintf(name,"token%d.png",i);
        Players[i].token=IMG_Load(name);
        sprintf(name,"tokenbig%d.png",i);
        Players[i].tokenbig=IMG_Load(name);
        sprintf(name,"owned%d.png",i);
        Players[i].owned=IMG_Load(name);
        Players[i].socket=NULL;
        Players[i].drawing=SDL_CreateSemaphore(1);

        //Players[i].turn=0;
        printf(IMG_GetError());
        SDL_SetAlpha(Players[i].token,SDL_SRCALPHA,SDL_ALPHA_TRANSPARENT);
    }
    noowned=IMG_Load("noowned.png");
    printf(IMG_GetError());
    printf("Loading fields data!\n");
    redraw_board();

}



/**

 * Halts the SDL application.

 */

void halt()

{

    delete gui;



    delete imageLoader;

    delete input;

    delete graphics;



    SDL_Quit();

}
void redraw_board(){
    SDL_Rect dest;
    SDL_FillRect(boardsurf,NULL,SDL_MapRGB(boardsurf->format,0xFF,0xFF,0xFF));
    int i;
    for (i=0;i<40;++i){
        if (client::board[i].type==core::STREET or
            client::board[i].type==core::RAILWAY or
            client::board[i].type==core::PLANT){
            dest=get_field_rect(i);
            //printf("%d-%d\n",i,client::board[i].owner);
            if (client::board[i].owner==-1){
                if (SDL_BlitSurface(noowned, NULL, boardsurf, &dest))
                    printf(SDL_GetError());
                //printf("%d-%d\n",i,Board[i].owner);
```

```
                }
            else{
                if (SDL_BlitSurface(Players[client::board[i].owner].owned
                        , NULL, boardsurf, &dest))
                    printf(SDL_GetError());
                //printf("FAIL:%d—%d\n",i,Board[i].owner);
                }
        }
    }
    dest.x = 0;
    dest.y = 0;
    dest.w = bmp->w;
    dest.h = bmp->h;
    //SDL_LockSurface(board);

    if (SDL_BlitSurface(bmp, NULL, boardsurf, NULL))
        printf(SDL_GetError());
    //SDL_BlitSurface(board, NULL, screen, NULL);
    //SDL_UnlockSurface(board);
}
void redraw(){
    //SDL_FillRect(screen,NULL,SDL_MapRGB(screen->format,0xFF,0xFF,0xFF));
    int i;
    SDL_BlitSurface(boardsurf,NULL,screen,NULL);
    //printf("%d,%d\n",Players[0].token_rect.x,Players[0].token_rect.y);
    SDL_Rect dest;
    for (i=0;i<client::players_count;++i){
        dest=get_token_rect(client::players[i].location,i);
        SDL_BlitSurface(Players[i].token,NULL,screen,&dest);
        dest=get_player_rect(i);
        if (i==client::current_player){
            SDL_FillRect(screen,&dest,0xAAAAAAFF);
        }
        SDL_BlitSurface(Players[i].tokenbig,NULL,screen,&dest);
        dest.x+=60;
        SDL_Color black,red;
        black.b=00;black.g=00;black.r=0;
        red.b=00;red.g=00;red.r=0xFF;
        SDL_Surface *label=TTF_RenderUTF8_Blended(hugeFont,
                client::players[i].name,black);
        SDL_BlitSurface(label,NULL,screen,&dest);
        dest.y+=30;
        switch (client::players[i].type){
            case core::LOCAL:
                label=TTF_RenderUTF8_Blended(normFont,"Локальный\0",black);
                SDL_BlitSurface(label,NULL,screen,&dest);
                break;
            case core::REMOTE:
                label=TTF_RenderUTF8_Blended(normFont,"Удаленный\0",black);
                SDL_BlitSurface(label,NULL,screen,&dest);
                break;
            case core::BANKROT:
                label=TTF_RenderUTF8_Blended(normFont,"Банкрот\0",black);
                SDL_BlitSurface(label,NULL,screen,&dest);
                break;
            default:
                ;
        }
        dest.y-=20;
        dest.x+=150;
        char money[50];
        int size=sprintf(money,"Баланс:%ld$",client::players[i].money);
        money[size]='\0';
        if (client::players[i].money<0){
```

```
            label=TTF_RenderUTF8_Blended(normFont,money,red);
        }
        else {
            label=TTF_RenderUTF8_Blended(normFont,money,black);
        }
        SDL_BlitSurface(label,NULL,screen,&dest);
    }
    gui->logic();
    // Now we let the Gui object draw itself.

    gui->draw();

    // Finally we update the screen.

    SDL_Flip(screen);
}


/**

 * Runs the SDL application.

 */


void run()

{


    SDL_Event event;

    std::cout<<"Start complete\n";

    while(running&&SDL_WaitEvent(&event))

    {

    // Check user input

        if (event.type == SDL_KEYDOWN)

        {

            if (event.key.keysym.sym == SDLK_ESCAPE)

            {

                client::running=false;
                server::running=false;
                running = false;
                break;

            }

            if (event.key.keysym.sym == SDLK_f)

            {

                if (event.key.keysym.mod & KMOD_CTRL)

                {

                    // Works with X11 only
```

```
                SDL_mutexP(lock);
                fullscreen=!fullscreen;
                if (fullscreen)
                    screen = SDL_SetVideoMode(boardw, boardh, 32,
                    SDL_HWSURFACE|SDL_DOUBLEBUF|SDL_FULLSCREEN);
                else screen = SDL_SetVideoMode(boardw, boardh, 32,
                    SDL_HWSURFACE|SDL_DOUBLEBUF);
                SDL_mutexV(lock);
                //SDL_WM_ToggleFullScreen(screen);

            }

        }

    }

    else if(event.type == SDL_QUIT)

    {

        client::running=false;
        server::running=false;
        running = false;
        break;

    }



    // After we have manually checked user input with SDL for

    // any attempt by the user to halt the application we feed

    // the input to Guichan by pushing the input to the Input

    // object.

    input->pushInput(event);



    // Now we let the Gui object perform its logic.

    //gui->logic();

    SDL_mutexP(lock);
    redraw();
    SDL_mutexV(lock);

    // Finally we update the screen.



}
/*SDL_WaitThread(client0, NULL);
SDL_WaitThread(client1, NULL);
SDL_WaitThread(client2, NULL);
SDL_WaitThread(client3, NULL);

SDL_WaitThread(client4, NULL);
SDL_WaitThread(client5, NULL);
SDL_WaitThread(client6, NULL);
SDL_WaitThread(client7, NULL);
```

```
        */
        if (server)
            SDL_WaitThread(server,NULL);


    }

}
```

Далее приводится часть сетевой программы-игры "Монополия которая занимается организацией GUI.

```
#ifndef WIDGETS_HPP_INCLUDED

#define WIDGETS_HPP_INCLUDED
/**

 * Code to populate a global Gui object with all the widgets

 * of Guichan.

 */

#include "sdlutf8truetypefont.hpp"
#include "utf8textfield.hpp"
#include "core.hpp"
#include <string>
#include <SDL_net.h>


namespace widgets

{

    class LoginWindow: public gcn::Window
    {
        public:


        LoginWindow(const std::string& caption);
        ~LoginWindow();
        gcn::Button *play;
        gcn::CheckBox *create;
        gcn::TextField *serv;
        gcn::TextField *port;
        gcn::TextField *client_num;
        gcn::Label *servl;
        gcn::Label *portl;
        gcn::Label *client_numl;
    };

    class PlayersWindow: public gcn::Window
    {
        public:
            PlayersWindow(const std::string& caption);
            ~PlayersWindow();
            gcn::ListBox *players;
            gcn::UTF8TextField *newplayer;
            gcn::Label *newplayerl;
            gcn::Button *addplayer;
            TCPsocket socket;
    };
    class FieldWindow: public gcn::Window
```

```
    {
        public:
            FieldWindow(const std::string& caption);
            ~FieldWindow();
            gcn::Icon *image;
            gcn::Label *price,*rent[6],house_cost,hotel_cost,mortgage_val;
            gcn::Label *info;
            gcn::Button *yes;
            gcn::Button *no;
            void show(int field);

    };
    class AuctWindow:public gcn::Window
    {
        public:
            AuctWindow(const std::string& caption);
            gcn::Label * players[8];
            gcn::Label * lprice[8];
            gcn::Label * ltimer;
            gcn::TextField * tprice[8];
            gcn::Button * submit[8];
            Sint64 price[8];
            int timer;
            void show();
            void update();
    };

    extern LoginWindow *window;
    extern PlayersWindow *pwindow;
    extern FieldWindow *bwindow;
    extern AuctWindow *awindow;
    void init();
    void halt();
}
#endif

#include "widgets.hpp"
#include "sdl.hpp"
#include "client.hpp"
#include <sstream>
namespace widgets

{
    std::string intToString(int i)
    {
        std::stringstream ss;
        std::string s;
        ss << i;
        s = ss.str();

        return s;
    }
    void start_client();

    //gcn::ImageFont* font;

    gcn::SDLUTF8TrueTypeFont *font;
    gcn::Container* top;

    gcn::Button* button,*exitbutton,*discbutton;

    LoginWindow *window;
    PlayersWindow *pwindow;
    FieldWindow *bwindow;
    AuctWindow *awindow;
```

```
class ButtonListener: public gcn::ActionListener
{
    void action(const gcn::ActionEvent &actionEvent)
    {
        if (client::psocks[client::current_player]){
            bwindow->setVisible(false);
            core::PacketData pdata;
            pdata.type=core::ACK;
            printf("player:%d\n",client::current_player);
            SDLNet_TCP_Send(client::psocks[client::current_player],
                    &pdata,sizeof(pdata));
        }
    }

} buttonListener;

class CreateListener: public gcn::ActionListener
{
    void action(const gcn::ActionEvent &actionEvent)
    {
        window->client_num->setVisible(window->create->isSelected());
        window->client_numl->setVisible(window->create->isSelected());
        window->serv->setEnabled(!window->create->isSelected());
        if (window->create->isSelected()){
            window->serv->setText("localhost");
        }

    }
} *createListener;
class ConnectListener: public gcn::ActionListener
{
    void action(const gcn::ActionEvent &actionEvent)
    {
        if (window->create->isSelected()){
            server::ServerStartData *param=new server::ServerStartData;
            param->port=atoi(window->port->getText().c_str());
            param->pCount=atoi(window->client_num->getText().c_str());
            sdl::server=SDL_CreateThread(server::main_server_thread,param);
        };
        start_client();
        window->setVisible(false);
        pwindow->setVisible(true);


    }
} *connectListener;

class AddPlayerListener: public gcn::ActionListener{
    void action(const gcn::ActionEvent &actionEvent)
    {
        core::PacketData pdata;
        core::copy_name(pwindow->newplayer->getText().c_str(),pdata.name);
        printf("%p,%p,%lu\n",&pdata.time,&pdata.name[1],sizeof(pdata));
        pdata.type=core::ACK;
        SDLNet_TCP_Send(pwindow->socket,&pdata,sizeof(pdata));
        printf("transmitted!\n");
        start_client();

    }
} *addplayerListener;

class BuyYesListener: public gcn::ActionListener
{
    void action(const gcn::ActionEvent &actionEvent)
```

```
    {
        if (client::psocks[client::current_player]){
            core::PacketData pdata;
            pdata.type=core::OWNED;
            //printf("player:%d\n",sdl::current_player);
            SDLNet_TCP_Send(client::psocks[client::current_player],
                    &pdata,sizeof(pdata));
            bwindow->setVisible(false);
        }
    }
} *buyyesListener;

class BuyNoListener: public gcn::ActionListener
{
    void action(const gcn::ActionEvent &actionEvent)
    {
        if (client::psocks[client::current_player]){
            core::PacketData pdata;
            pdata.type=core::NOT_OWNED;
            SDLNet_TCP_Send(client::psocks[client::current_player],
                    &pdata,sizeof(pdata));
        }
        bwindow->setVisible(false);
        awindow->show();
    }
} *buynoListener;

class SubmitAuctListener: public gcn::ActionListener
{
    void action(const gcn::ActionEvent &actionEvent)
    {
        printf("action\n");
        int    player=atoi(actionEvent.getId().c_str());
        if (client::psocks[client::current_player]){
            core::PacketData pdata;
            pdata.type=core::AUCTION;
            pdata.money[0]=atoi(awindow->tprice[player]->getText().c_str());
            SDLNet_TCP_Send(client::psocks[player],&pdata,sizeof(pdata));
        }
    }
} *submitAuctListener;

class ListListener: public gcn::WidgetListener
{
    void widgetResized(const gcn::Event& event) {
        pwindow->newplayer->setY(10+pwindow->players->getHeight());
        pwindow->addplayer->setY(10+pwindow->players->getHeight());
        pwindow->resizeToContent();
        pwindow->setX((sdl::boardw-pwindow->getWidth())/2);
        pwindow->setY((sdl::boardh-pwindow->getHeight())/2);
        //this->add(addplayer,80,10+players->getHeight());
    }
} *listListener;

class PlayersList: public gcn::ListModel
{
    int getNumberOfElements(){
        return client::players_count;
    }

    std::string getElementAt(int i){
        return std::string(client::players[i].name);
    }
} *playersList;
```

```
/**
 * Initialises the widgets example by populating the global Gui
 * object.
 */
void start_client(){
    client::ClientStartData *param=new client::ClientStartData;
    param->host=window->serv->getText().c_str();
    param->port=atoi(window->port->getText().c_str());
    param->socket=&pwindow->socket;
    printf("starting client\n");
    SDL_CreateThread(client::client_thread,param);
}

void init()

{

    // We first create a container to be used as the top widget.

    // The top widget in Guichan can be any kind of widget, but

    // in order to make the Gui contain more than one widget we

    // make the top widget a container.

    top = new gcn::Container();
    top->setOpaque(false);

    // We set the dimension of the top container to match the screen.

    top->setDimension(gcn::Rectangle(0, 0, sdl::boardw, sdl::boardh));

    // Finally we pass the top widget to the Gui object.

    sdl::gui->setTop(top);



    // Widgets may have a global font so we don't need to pass the

    // font object to every created widget. The global font is static.

    font=new gcn::SDLUTF8TrueTypeFont("OpenSans-Regular.ttf",12);
    gcn::Widget::setGlobalFont(font);



    // Now we create the widgets



    button = new gcn::Button("Завершить ход");
    button->setWidth(3*sdl::fieldw-sdl::groupmarg);
    button->setHeight(2*sdl::fieldh-sdl::groupmarg);

    button->addActionListener(&buttonListener);
```

```
    exitbutton = new gcn::Button("Выход");
    exitbutton->setWidth(1*sdl::fieldw);
    exitbutton->setHeight(1*sdl::fieldh);

    discbutton = new gcn::Button("Отключиться");
    discbutton->setWidth(2*sdl::fieldw-sdl::groupmarg);
    discbutton->setHeight(1*sdl::fieldh);



    window = new LoginWindow("Окно входа");

    window->setBaseColor(gcn::Color(255, 150, 200, 190));


    pwindow=new PlayersWindow("Список игроков");
    pwindow->setBaseColor(gcn::Color(255, 150, 200, 190));

    pwindow->setVisible(false);

    bwindow=new FieldWindow("Купить поле");
    bwindow->setBaseColor(gcn::Color(255, 150, 200, 190));

    bwindow->setVisible(false);

    awindow=new AuctWindow("Аукцион");
    awindow->setBaseColor(gcn::Color(255, 150, 200, 190));

    awindow->setVisible(false);

    //window->resizeToContent();
    top->add(button, sdl::boardw-4*sdl::fieldw-sdl::marginl,
            sdl::boardh-3*sdl::fieldh-sdl::marginb);
    top->add(exitbutton, sdl::boardw-4*sdl::fieldw-sdl::marginl,
            sdl::boardh-4*sdl::fieldh-sdl::marginb);
    top->add(discbutton, sdl::boardw-3*sdl::fieldw-sdl::marginl,
            sdl::boardh-4*sdl::fieldh-sdl::marginb);
    top->add(window,(sdl::boardw-window->getWidth())/2,
            (sdl::boardh-window->getHeight())/2);
    top->add(pwindow,(sdl::boardw-pwindow->getWidth())/2,
            (sdl::boardh-pwindow->getHeight())/2);
    top->add(bwindow,(sdl::boardw-bwindow->getWidth())/2,
            (sdl::boardh-bwindow->getHeight())/2);
    top->add(awindow,(sdl::boardw-awindow->getWidth())/2,
            (sdl::boardh-awindow->getHeight())/2);

}


/**

 * Halts the widgets example.

 */

void halt()

{

    delete font;

    delete top;
```

```
    delete button;

    delete window;

}
LoginWindow::LoginWindow(const std::string& caption)
    :Window(caption)
{
    play=new gcn::Button("Подключиться");
    connectListener=new ConnectListener;
    play->addActionListener(connectListener);
    create=new gcn::CheckBox("Создать сервер",true);
    serv=new gcn::TextField("127.000.000.001");servl=new gcn::Label("Сервер: ");
    serv->setEnabled(false);
    port=new gcn::TextField("7887");portl=new gcn::Label("Порт: ");
    client_num=new gcn::TextField("8");client_numl=new gcn::Label("Игроков: ");
    createListener=new CreateListener;
    create->addActionListener(createListener);
    this->add(play,30,90);
    this->add(create,5,50);
    this->add(servl,5,5);this->add(serv,10,20);
    this->add(portl,120,5);this->add(port,125,20);
    this->add(client_numl,27,68);this->add(client_num,90,65);

    this->resizeToContent();
    serv->setText("localhost");


}
LoginWindow::~LoginWindow()
{
    delete play;
    delete create;
}
PlayersWindow::PlayersWindow(const std::string& caption)
    :Window(caption)
    {
    playersList=new PlayersList;
    players=new gcn::ListBox(playersList);
    addplayer=new gcn::Button("Добавить");
    newplayer=new gcn::UTF8TextField("Имярек   ");
    addplayerListener=new AddPlayerListener;
    addplayer->addActionListener(addplayerListener);
    listListener= new ListListener;
    players->addWidgetListener(listListener);
    this->add(players,5,5);
    this->add(newplayer,5,10+players->getHeight());
    this->add(addplayer,80,10+players->getHeight());

    this->resizeToContent();
    //this->setHeight(300);
}
PlayersWindow::~PlayersWindow()
{

}
FieldWindow::FieldWindow(const std::string& caption)
    :gcn::Window(caption)
    {
        image=new gcn::Icon(gcn::Image::load("groupinfo.bmp"));
        this->add(image,0,0);
        int i;
        price=new gcn::Label("asd");
        this->add(price,170,67);
```

```
        price->setWidth(20);
        price->setAlignment(gcn::Graphics::RIGHT);
        for(i=0;i<6;++i){
            rent[i]=new gcn::Label("asd");
            this->add(rent[i],170,87+14*i);
            rent[i]->setWidth(20);
            rent[i]->setAlignment(gcn::Graphics::RIGHT);
        }
        //this->add(info,0,0);
        yes=new gcn::Button("Да");
        no=new gcn::Button("Нет");
        info=new gcn::Label("Хотите купить?");

        this->add(info,5,5+image->getHeight());
        this->add(yes,20,20+image->getHeight());
        this->add(no,50,20+image->getHeight());
        buyyesListener=new BuyYesListener;
        buynoListener=new BuyNoListener;
        yes->addActionListener(buyyesListener);
        no->addActionListener(buynoListener);
        this->resizeToContent();
    }
FieldWindow::~FieldWindow(){
}
void FieldWindow::show(int field){
    int i;
    price->setCaption(intToString(client::board[field].price));
    for(i=0;i<6;++i){
        rent[i]->setCaption(intToString(client::board[field].rent[i]));
    }
    this->setVisible(true);
}
AuctWindow::AuctWindow(const std::string&caption)
    :gcn::Window(caption)
    {
        int i;
        for(i=0;i<8;i++){
            submit[i]=new gcn::Button("Принять");
            submit[i]->setActionEventId(intToString(i));
            lprice[i]=new gcn::Label("0");
            lprice[i]->setWidth(55);
            tprice[i]=new gcn::TextField("0");
            tprice[i]->setWidth(55);
            players[i]=new gcn::Label("");
            players[i]->setWidth(55);
            price[i]=0;
        }
        ltimer=new gcn::Label("5");
        submitAuctListener=new SubmitAuctListener;
        for(i=0;i<8;i++){
            price[i]=0;
            players[i]->setCaption(std::string(client::players[i].name));
            this->add(players[i],60*i+5,5);

            this->add(submit[i],60*i+5,45);
            submit[i]->addActionListener(submitAuctListener);
            this->add(tprice[i],60*i+5,20);
            this->add(lprice[i],60*i+5,20);
            submit[i]->setVisible(false);
            tprice[i]->setVisible(false);
            lprice[i]->setVisible(false);
            players[i]->setVisible(false);

        }
```

```
            timer=10;
            this->add(ltimer,100,35);
        }
    void AuctWindow::show(){
        int i;
        for(i=0;i<client::players_count;i++){
            players[i]->setCaption(std::string(client::players[i].name));
            players[i]->setVisible(true);
            if (client::players[i].type==core::LOCAL){
                tprice[i]->setText(intToString(price[i]));
                submit[i]->setVisible(true);
                tprice[i]->setVisible(true);
                lprice[i]->setVisible(false);
            } else {
                lprice[i]->setCaption(intToString(price[i]));
                submit[i]->setVisible(false);
                tprice[i]->setVisible(false);
                lprice[i]->setVisible(true);
            }
        }
        ltimer->setCaption(intToString(timer));

        this->setVisible(true);
        this->resizeToContent();
    }
    void AuctWindow::update(){
        int i;
        for(i=0;i<client::players_count;i++){
            if (client::players[i].type==core::REMOTE){
                lprice[i]->setCaption(intToString(price[i]));

            }
        }
        ltimer->setCaption(intToString(timer));
        SDL_mutexP(sdl::lock);
        sdl::redraw();
        SDL_mutexV(sdl::lock);
    }

}
```

Далее приводится основная часть сетевой программы-игры "Монополия".

```
/**
 * This is an example that shows of the widgets present in
 * Guichan. The example uses the SDL back end.
 */
#include <SDL_net.h>
#include <guichan.hpp>
#include <iostream>

// Here we store a global Gui object.  We make it global
// so it's easily accessable. Of course, global variables
// should normally be avioded when it comes to OOP, but
// this examples is not an example that shows how to make a
// good and clean C++ application but merely an example
// that shows how to use Guichan.
namespace globals
{
    gcn::Gui* gui;
}

// Include code to set up an SDL application with Guichan.
// The sdl.hpp file is responsible for creating and deleting
```

```
// the global Gui object.
#include "sdl.hpp"
// Include code to set up a Guichan GUI with all the widgets
// of Guichan. The code populates the global Gui object.
#include "widgets.hpp"
#include "server.hpp"
#include "client.hpp"
#include "core.hpp"

int main(int argc, char **argv)
{
    try
    {


        core::init(client::board,client::players);

        sdl::init();
        widgets::init();
        sdl::run();
        widgets::halt();
        sdl::halt();
    }
    // Catch all Guichan exceptions.

    catch (gcn::Exception e)
    {
        std::cerr << e.getMessage() << std::endl;
        return 1;
    }
    // Catch all Std exceptions.
    catch (std::exception e)
    {
        std::cerr << "Std exception: " << e.what() << std::endl;
        return 1;
    }
    // Catch all unknown exceptions.
    catch (...)
    {
        std::cerr << "Unknown exception" << std::endl;
        return 1;
    }

    return 0;
}
```