A Toogashada Game

Project repository: https://gitlab.fi.muni.cz/xradek1/pv264\_project.

# Description

Our game has client/server architecture. Clients communicate with server over TCP connection using Boost ASIO library. All the game state is maintained by server, which periodically sends all information needed to clients. Clients only (1) render the game state, and (2) sends informations about user’s action to server.

Physically, the source code of the project is organized into directories common, server and client. From those we build two separate targets for client and server. We decided to keep the server and client separate instead of building them to one executable, because the server does not need some libraries client needs (SDL).

The files client/client\_main.cpp and server/server\_main.cpp represents the entry points for client and server. They are responsible for handling command line arguments (not implemented yet) and terminal input (signals; currently implemented correctly only in server).

## Networking

To represent a connection to remote we use an interface named IConnection (see common/IConnection.h). Through the interface, simple one-way messages can be sent. Message consists of tag, which represents a message type, and variable-sized data (see common/Message.h). Concrete messages (common/Messages.h) are then represented as serializable/deseriazable structures. Since there are not many messages, the serialization is done manually; this is likely to get changed.

The interface is implemented in classes ConnectionToServer and ConnectionToClient; however, the common parts of the two are implemented in class Connection.

## Server

Server runs in two threads. One of them handles networking, while the other runs the main game loop. When a client connects, server creates new 2D game object and a player and sends appropriate information to clients.

## 

## Client

Client in client/Client.h is the main class.

ClientController implements client-side game logic.

ClientGui - screen drawing.

## Game model

Stores game objects (players, obstacles,...) and performs physic's computations.

## Other

* We have some geometric stuff implemented in common/GPoint.h, common/PolygonalShape.h and common/Angle.h. Nothing special/nice.
* Various game objects (2D objects, players etc) have their IDs. Templated class GenericManager (common/GenericManager.h) owns such objects and also manages the mapping from IDs to objects.

# What we have so far

* Clients can connect to and disconnect from server.
* With every client, there is an associated 2D object in the game area.
* User can move with ‘his’ object and all clients see it.

# What is missing

* No command-line control.
* We haven’t tried it on real network, so there may be some problems.
* No automated tests :-(.
* The code is not really nice.

# Next steps

* Refactor some parts of the source code. In server, move some responsibilities from class Server to GameModel. We will consider using futures Instead of IConnection:IHandler.
* Collision detection for game objects. Do not allow game objects (and players) to overlap.
* Shooting. Either simple ‘laser’ shots (easy to detect), or some fast moving balls…
* Game area. Players should be able to move only on an area of well-specified size. Also clients should respect it and create windows of appropriate size, or do some rescaling.
* Game logic. Create game, wait for players to connect, play, ...