Assignment 7 Report

By: Andréas Kaytar-LeFrançois, 6467076

Dmitry Anglinov, 5958895

**Problem statement:** Develop a multiplayer Tetris clone with client server architecture.

**Final Tetris Game Outline:**

**Original Design:**

The original idea behind a basic Tetris game remained the same. The goal is to fill and destroy as many rows as fast as possible. There are eight different shapes: S-shape, Z-shape, T-shape, L-shape, MirroredLShape, Line-shape, Square shape and the null shape. All shapes fall from the top of the board toward the bottom. The player is able to move and rotate the shapes as they fall at a certain speed and position them appropriately on the board. If a complete row is formed on the board, then that row is destroyed and the player’s score increases. If a player reaches the top of the board (tops out), the player loses the game. The basic Tetris game logic was imported from an open source website.

**Changes:**

Although the basic premise was unchanged, many new features were added to expand the barebones Tetris game that was imported from the internet. During development we added “hold piece” functionality that allows the user to reserve a piece that is currently falling down and use it later in the game. By pressing the “shift” key, the user can reserve a piece, which is removed from the board and clearly displayed on the top left of the user interface. Functionality was also added to allow the user to view the next piece that will appear on the board once the current piece has reached its desired position. The “next piece” window is clearly displayed on the top right of the user interface.

Media options were also added to the current version of the Tetris game. The user is granted control of the game by being able to pause, restart and mute the application. Appropriate icons are attached to all media buttons that change dynamically according to represent the current state of the system. By clicking the mute button or pressing the ‘m’ hotkey, the user is able to toggle the game soundtrack on and off. The mute button also effects whether the sound effects produced during the game will be performed. A user is only able to restart the game once the game is paused or if a player has won.

The user interface was developed with a “Nimbus” look and feel. The user interface was designed to be colourful and pleasing to look at. Controls on the user interface were appropriately spaced apart and important labels were highlighted in colour.

**Client-Server Functionality:**

**Original Design:**

The OCSF framework will be used in order to create the client server architecture of this project. During gameplay all connected users are able to communicate through a simple chat box application in the main UI. The chat box should hold the last 15 lines of messages and will identify users by prefixing their username to their message. During gameplay, either client can terminate the game by disconnecting or closing the server. At the end of a game, the scores of both players will be presented on the interface and a winner will be shown. The server is thus able to select 2 new players to play again.

**Changes:**

Various additional client server features were added during development of the system. The distributed system was built off the OCSF framework by extending the classes AbstractClient and AbstractServer. A standard client-server system was used in implementing the game. A server must be started and clients must connect to the server to play against each other. However, a client is not forced to connect to a server to play a game of Tetris and is able to play the game in singleplayer. This decision was made due to the fact it was considered highly important that a player is able to play the game as a single player game, since Tetris has single player roots.

Nested classes were used in order to construct the client-server architecture. The nested class “TetrisClient” extended the class AbstractServer and was embedded in the class “Tetris2P” since any player that opens up the game can be considered to be a client. The class TetrisServer was developed separately and extends AbstractServer. The game has a server user implemented by the class “ServerConsole” imported and modified from the OSCF assignment. The user interface informs the user whether the game is in single player or multiplayer mode through an information label positioned near the bottom right of the user interface. The label displays the current network status and connection information of the client.

If a client wishes to participate in a multiplayer game they must set the host and the port of the server they wish to connect to using commands from the chat box in the user interface. Once connected, players are able to chat with each other using a text input on the user interface. A “readytostart” flag is used to indicate if players are ready to begin the game. Once, both flags are set the players are able to play a game of multiplayer Tetris.

An algorithm was developed to match a newly connected client with a client that doesn’t currently have an opponent. Currently connected clients are stored in a LinkedList that contains a client’s username, connection ID and the connection ID of the opponent. If a client disconnects, the disconnected client’s opponent is informed that they no longer have an opponent and must wait for a new client to connect before being able to play another online game.

The chat system is capable of more than exchanging dialogue. The client and the server are able to issue commands to obtain information from the system. A few new commands were added since the first version of the Tetris game was submitted. The list of new commands can be seen below:

**Client:**

‘/connect’ – Attempts to connect to a server and convert a singleplayer game of Tetris to a multiplayer game.

‘/disconnect’ – Disconnects from a multiplayer game of Tetris to return to singleplayer mode.

**Server:**

‘/status’ – Displays the port number and hostname of the server.

**New UML**

