Assignment 5

By: Andréas Kaytar-LeFrançois, 6467076

Dmitry Anglinov, 5958895

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

**Requirements outline:**

The goal in a game of Tetris 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 main interface will be simple. The user can choose to play alone or insert commands to set up a server or connect to a server in the chat window. If there are more than 2 users in a server’s lobby, two users are randomly selected to play against each other. The game ends when either player tops out or the remaining player is declared the winner. 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.

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.

**Use cases:**

**First Use Case**

**Use Case:** Setting up the server.

**Summary:** The user sets up a server and waits for clients to join.

**Related User Case:** Connecting to server

**Steps:**

|  |  |
| --- | --- |
| **Actor actions** | **System Response** |
| Type in the new server command followed by the port number in chat input field. | The server attempts to set itself up and confirms its state. |
| Type in username in chat input field. | Server acknowledges user with welcome message. Displays current lobby status. |
|  | Server attempts to place the user in a match |

**Second Use Case**

**Use Case:** Chatting on the server

**Summary:** Connected users can communicate in a chat box. Users enable chat by hitting the designated chat key “T” to talk. They type in messages and hit the “ENTER” key to send their messages.

**Steps:**

|  |  |
| --- | --- |
| **Actor actions** | **System Response** |
| Hit the key “T” | Chat input area is focused. A marker is drawn around the chat to indicate focus has changed. |
| Type in text in text field |  |
| Press “enter” | Display the text in the chat box prefixed with player’s username. |
| Click on board or hit the key “T” | Chat input area is unfocused. A marker is drawn around the chat to indicate focus has changed. |

**Third Use Case**

**Use Case:** Moving piece

**Summary:** The user uses the arrow keys to move and rotate the shape. The space key causes the shape to fall down faster.

**Steps:**

|  |  |
| --- | --- |
| **Actor actions** | **System Response** |
| Press up arrow key | Rotate shape clockwise |
| Press down arrow key | Lower shape by one |
| Press left arrow key | Translate shape left |
| Press right arrow key | Translate shape right |
| Press the space key | Drop shape down |

**Fourth Use Case**

**Use Case:** Host player closes the game

**Summary:** The user can exit the game by clicking the quit button

**Related Use Case**: Client player closes the game

**Steps:**

|  |  |
| --- | --- |
| **Actor actions** | **System Response** |
| Click on the “Exit” button or “X” | Purge clients |
|  | Terminate server |

**Architecture description:**

**Languages**

This project will be written entirely in Java. The Swing and AWT libraries will be imported in order to design and implement the user interface. The final project will also include image icons and sound files.

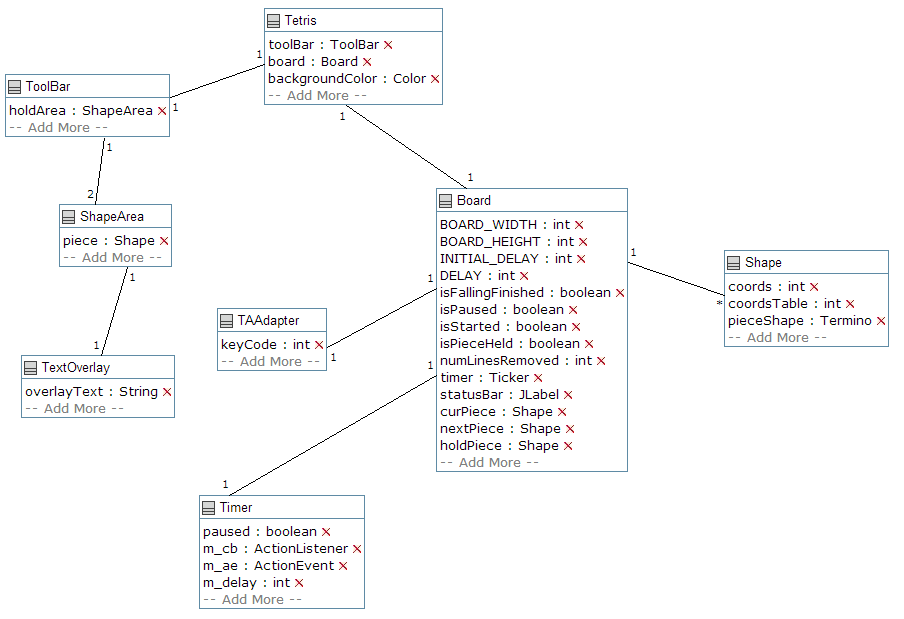
**Client:**

The OCSF framework will be used in order to create the client server architecture of this project. The system will be multi-client, thus multiple clients will be able to connect to a server. The abstract class AbstractClient will be extended to implement the client for the Tetris application. The abstract and hook methods of the AbstractClient class will be overwritten and new methods will be implemented as necessary to develop a Tetris multiplayer client.

**Server:**

The OCSF framework will be used in order to create the client server architecture of this project. The abstract class AbstractServer will be extended to implement the client for the Tetris application. The abstract and hook methods of the AbstractServer will be overwritten and new methods will be implement as necessary to develop a Tetris multiplayer server. The server will have a user who will act as the host and will be able to interact and participate in games like all the other clients.

**UML diagram:**



**Requirements and design of communications:**

**Problem:** The Tetris game must be able to constantly update both players on the events that are occurring in both boards during gameplay. Thus, both players and all the users connected to the server should be able to see the boards of both competing players synchronously and information should be passed between clients and server dynamically. All users connected to the server should also be able to communicate on a chat included in the Tetris game frame.

**Background Information:** The planned implementation of the client-server system for the Tetris game is presented below.

**Issue 1:** What are the objects that should be exchanged between clients and server?

**Decision:** The entire Board object should be exchanged. The reason behind exchanging the entire board is due to the fact that an instance of the board object will contain all necessary information to inform the users on the game’s progress.

**Issue 2:** When should the Board objects be exchanged among users?

**Decision:** The information will be exchanged every call to the ActionPerformed method in the Board class. The reason for this is because the Tetris board enters a new state every time the ActionPerformed method is called.

**Issue 3:** Which users should be constantly receiving and outputting Board objects from and to the network?

**Decision:** The Board objects are exchanged between the 2 competing players and will also be sent to all users currently connected to server. Users that are not playing the game do not send Board objects but they do receive them from both competing players in order to be able to observe the game.

**Issue 4**: Should a user be able to stop receiving messages?

**Decision:** No, as long as a user is connected to the server they must be able to receive all updates on the current Tetris game and must have the chat log updated.

**Issue 5:** Should all users (including players) be able to see the board GUI of both players?

**Decision:** Yes

**Issue 6:** If a player pauses the game, would the opposing player’s game be paused?

Decision: Yes, both Tetris games would be paused if a user pauses their game.

**Issue 7:** If a player pauses the game, would the other player be able to resume the game?

**Decision:** Yes, but users that are not players may not. Non-player users are also unable to pause the game.

**Issue 8:** If a player loses by topping out, should the game end for the opposing player?

**Decision:** Yes, if a player tops out then the game is over for both players.

**Issue 9:** What if a player disconnects from the server or terminates the game?

**Decision:** A user should be able to disconnect from the server at all times. If a player disconnects from the server, the opposing player is declared the winner and new players are selected. If a client that is not currently a player is disconnected, then the game proceeds like normal. If the server user is disconnected, then the game is terminated and all clients are disconnected from the server. The server is also terminated.

**Issue 10:** If a game is over, how are the new players selected?

**Decision:** The server holds an *ArrayList* of currently active users connected to the server. It randomly selects 2 players to play.

**Issue 11:** Can users communicate with each other?

**Decision:** Yes, there will be a simple chat client to allow all users to communicate with each other.

**Issue 12:** How are users identified during chat?

**Decision:** User comments would be prefixed with their usernames followed by a “>”.

**Issue 13:** How will the chat system function?

**Decision:** The chat will be hosted on the server. Clients will be able to send string messages to the server and the server will then output them on all user GUI.

**Issue 14:** How many messages will be allowed to be displayed the chat screen?

**Decision:** The chat will only display the last 15 entries in order to conserve memory. Further chat will cause previous messages to “leak out” and be deleted to make room for new messages.

**Environment and System Models:** The game should function normally on any operating system due to the Java Virtual Machine. The OCSF Framework is used in order to construct the client-server architecture. Classes are created to extend OCSF framework abstract classes in order to construct a distributed system. This requirements document describes the type of data transmitted on the network and the conditions on data transfer. The UML diagram of the Tetris game is provided in this report. Clients must connect to a server hosted by the server user. The hostname and port of the server must be known in order for clients to connect.

**Functional Requirements:** Inputs to the game are made by the keyboard using preset keycode controls. They are described below. Each of the actions below should be observable by all users currently connected to the server.

**Pause – ‘p ‘**

* A player pauses the game. Both players’ Tetris boards will enter the paused state.
* Pressing the ‘p’ key again will un-pause the boards for both players.
* Players should still be able to chat while paused.

**Restart – ‘r‘**

* A player restarts his single player game. The player’s Tetris boards will reset and the game will then be paused.

**Quit – ‘q’**

* A user quits the game if this command is pressed by a player.
* A player who quits the game does not disconnect from the server, but is declared the loser of the game.
* A new game is initiated and launched in the pause state with 2 new players.

**Moving shapes – ‘a’, ‘s’, ‘d,’ ‘left’, ’right’, ‘down’, ‘space’,**

* A player pressing these keys will cause a piece to translate left, be nudged down, translate right or be dropped to the bottom respectively.
* Any changes in state of the board should be outputted to the network to all users connected to the server in order for them to observe the change.

**Rotating shapes – ‘w’, ‘up’**

* A player pressing these keys will cause a piece to rotate clockwise
* Any changes in state of the board should be outputted to the network to all users connected to the server in order for them to observe the change.

All users should also be able to enter basic commands into the chat screen in order to be able to obtain information about the server. Chat commands are described below and should be proceeded by a ‘/’ symbol.

**Client-side:**

**/quit:** Disconnect from server gracefully. Alternatively, a button can also be pressed to disconnect from server. Makes sure connection to server terminated before program is exited.

**/logoff:** Causes client to disconnect from server, but not quit.

**/sethost <host>**: Calls the setHost method. Only allowed if client is logged off.

**/setport <port>**: Calls the setPort method. Only allowed if client is logged off.

**/gethost:** Displays current host name.

**/getport:** Displays the current port number.

**Server-side:**

**/quit:** Quit server user from server. Terminates the server gracefully and disconnects all clients.

**/stop**: Causes the server to stop listening for clients.

**/start:** Makes the server start listening for new clients.

**/close:** Causes server to stop listening for clients and to disconnect all connected clients.

**/setport <port>**: Calls the setPort method. Only allowed if server is stopped.

**/getport:** Displays the current port number.

**Other Requirements:**

There should be a delay of no more than 500 miliseconds for all operations, commands and data transfers across the network for a user with a stable internet connection.