BATTLE BLOKZ

Fall 2014 CSCI-201 Final Project

Jerry Hsiung, Ichen Ko, Steven Mai, May Wu, Henry Lau

Table of Contents

Project Proposal **3**
High-Level Requirements **3**
Technical Specifications **4**
Detailed Design **6**
Test Cases **7**
Deployment **14**

Project Proposal

Our project idea is multiplayer Tetris Battle. The goal of the game is to be the last person/team remaining. Players and teams can target individual opponents and send permanent brick lines when a specific number of lines are cleared. Once a player has bricks that reach the top, the player will be knocked out of the game, and it will be up to the teammate to win the game. Standard rules of Tetris apply. There are no special combos or the ability to hold pieces. The controls will be the arrow keys on the keyboard to move and rotate pieces and spacebar to perform a hard drop.

Multi-threaded code will be used in the chatting function of the game. There will be a chat box in the game where the user can see messages from his team or messages from all players playing. The players will need to hit "enter" on the keyboard to access and chat with their respective team. Players can also talk to every player by hitting the "enter" key and typing "/a" [message].

Networking will be used in order to play together on separate applications. Each player will have their own game screen, but each player can see the number of lines left until knockout of other players in order to generate strategies.

High-Level Requirements

Overview:

A multiplayer game of Tetris where each player can pick one of the two teams and sends lines to the opponent teams. The goal is to be the last player/team alive.

Requirement:

When the game starts, players have 3 options: Play, Help, and Quit. Play lets players to start the game; Help contains instructions on how the game works and the shortcuts that can be used; and Quit lets players quit the game.

In the process of picking teams, players can choose which one of the two teams they want to join and quit if they regret joining one team. Players would be assigned a small icon as their characters for the game. When a player is ready, he

would click on the start button, and he won't be able to change his team at this point. When all players have clicked "Start," the game would start. The game won't start and an error message will show if the teams don't have the same amount of players.

The game panel would have a large panel on the left where players play Tetris. For every 10 lines he clears, he would send one line to the targeted opponent. On the right top corner, there will be a list of opponents, followed by his teammates. The number of lines each opponent player has left until he is sent a line will be next to each player's name. There is an arrow pointing to the targeted opponent player, and the player can choose to target another opponent by typing "c." On the left bottom of the screen, there's a chat box. To send a chat, player must type, "enter" and type by starting with either "/a" (message all players) or "/t" (message to teammates only). Pressing "enter" would bring the player back to the game mode, meaning the player wouldn't be able to control his blocks if he's typing in the chat box until he presses enter to send his message or clicks the game panel.

Players die when their highest block touches the top of the game panel. The game ends when there's only one player left in the game. This would bring them to another screen, which will show the amount of lines each player sends and which team wins. Players can then choose to quit or replay the game. Clicking "Replay" would bring the player back to the home screen.

Technical Specifications

When the game starts, the user is shown a Start window with "Username" and "Password" fields and "Log On", "Forgot Login" and "Create Account" buttons. If the player creates account, the player will be redirected to a new page that has "Username", "Password" and "Email" fields and a "Create" or "Cancel" button. Once the account has been created, user will be redirected to next page with three options. And new account information is sent to our database. Once the player logs on, he has 3 options: "Play", "Help", and "Quit". The "Play" button lets players to start

the game; the “Help” button contains instructions on how the game works and the shortcuts that can be used; and the “Quit” button lets players quit the game.

In the process of picking teams, players can choose which one of the two teams they want to join and quit if they regret joining one team. Players would be assigned a small icon as their characters for the game. When a player is ready, he would click on the start button, and he won’t be able to change his team at this point. When all players have clicked “Start,” the game would start. The game won’t start and an error message will show if the teams don’t have the same amount of players.

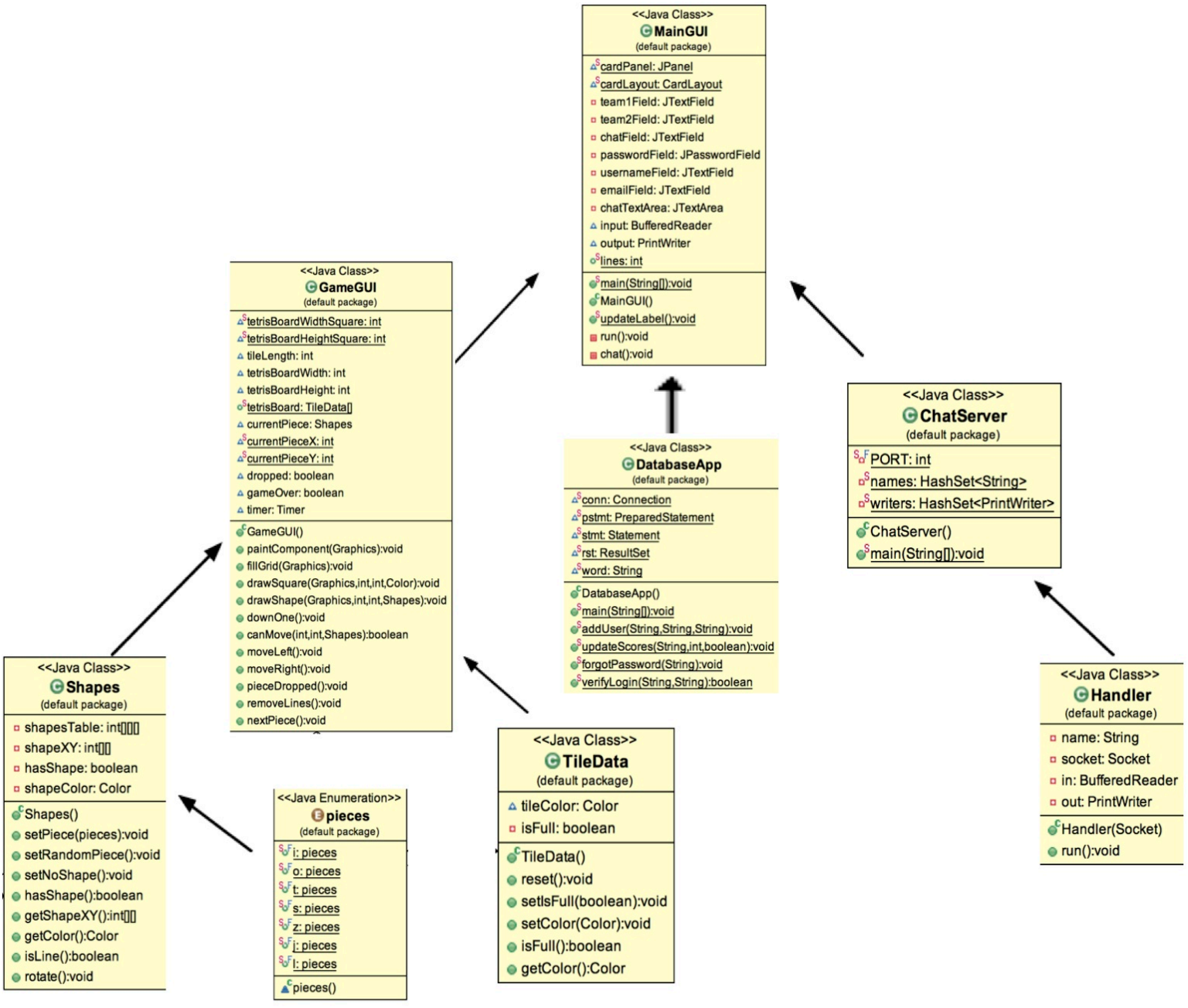
The game panel would have a large graphics panel on the left where players play Tetris. For every 5 lines he clears, he would send a line to all opponents. On the right top corner, there will be a list of opponents, followed by his teammates. On the left bottom of the screen, there’s a chat box. To send a chat, player must click on the chat box and type by starting with either “/a” (message all players), “/t” (message to teammates only), or “/username” (message specific user). Pressing “esc” would bring the player back to the game mode, meaning the player wouldn’t be able to control his blocks if he’s typing in the chat box until he presses enter to send his message.

Players die when their highest block touches the top of the game panel. The game ends when there’s only one player or one team left in the game. The statistics of the game, such as number of lines would then be exported and displayed into the next screen. Ranking will be based on who stayed alive the longest. The next screen would also shows which team wins. Players can then choose to quit or replay the game. Clicking the Replay button would bring the player back to the home screen, and the highest amount of lines sent of each user would be updated in the database; and clicking the Quit button would exit the program.

- Database Design
    - username (varchar[100])
    - password (varchar[100])
    - win (int[5])
    - total game played (int[5])

- number of lines sent (int[5])
- email (varchar[100])

Detailed Design



Test Cases

Black box testing

Black box testing tests for functionality without looking at the code and in the case of our game, we go through each sequence to make sure all buttons, text fields, and gameplay work as described. White box testing tests for internal structure while looking at the code and one of the easiest ways of doing this is to use print statements. For example, our game adds lines to other players when you complete five lines. We can test it by printing out where the line is sent from.

ex.

```
if (join_team1) {
        out.println("ADDLINE" + 1 + " " + temp_lineSent + " " + username);
        System.out.println("sent line from" + 1);
}
else {
        out.println("ADDLINE" + 2 + " " + temp_lineSent + " " + username);
        System.out.println("sent line from" + 2);
}
```

tileData Class

The purpose of this class is to represent the status of each tile on the current Tetris board. It does not provide any functionality so it only contains getter and setter methods for its data.

Shapes Class

This class provides the coordinates that the different shapes use to render on the Tetris board. This class has two main functions, setRandomPiece() and rotate().

setRandomPiece() – This method sets the current shape to a random shape using a random number generator to output a number between 0 and 7, each representing a different shape.

To check if the random number generator, output each number on to the screen and check if the number renders the correct shape on the Tetris board.

rotate() – This method attempts to rotate the current shape clockwise and does nothing if this is not possible.

```
for (each of the four squares of a piece) {
        // To check if X > 10 (out of bounds) or Y > 22 (out of bounds)
        System.out.println ( rotated X coordinate and rotated Y coordinate);

        // To check if the piece will land on an occupied location
        System.out.println ( tiles[rotated X coordinate][rotated Y coordinate] );
}
```

GameGui Class

This class contains the main functionalities of the Tetris game. This class has () main methods – drawSquare(), drawShape(), canMove(), removeLines(), and keyboardListener.

JUnit testing tests individual functionality of the code by writing customized programming. The following are JUnit tests.

drawSquare() – Paramters – Graphics g, int gridX, int gridY, and Color c
This is the method that is responsible for drawing the Tetris board and the tiles of the current Tetris piece. I will unit test it by attempting to draw a square on every row and column.

```
for( each column ) {
        // For each number of squares drawn on each row
        for( total row / total column) {
```

```
            drawSquare( Graphics g, currentColumn, currentRow, Color);
        }
}
```

drawShape() –
Paramters – Graphics g, int gridX, int gridY, and shape CurrentShape

This class provides the drawSquare class with coordinates to draw all 7 Tetris pieces. To unit test this method, I will render each of the 7 Tetris pieces on various locations of the Tetris board.

```
for ( each shape ) {
        // Provide a 4x4 square for each shape
        for( each tile of the current piece ) {
            drawSquare(g, i*4 + X coordinate for of piece, i*4 – Y coordinate of
            piece, color of current shape);
        }
}
```

canMove() –
Parameters – int newXafterMove, int newYafterMove, piece currentPiece

This class decides if the current piece can move down, left, and right and return true or false depending on the class' result.

```
for ( each of the 4 tiles of the current piece ){
        // Check for out of bounds on the bottom and sides
        System.out.println (newX + ", " + newY);
        // To check if the piece will land on an occupied location and see if the
        function outputs true when it's not occupied and false if it is.
```

```
        System.out.println (tile[newX][newY].isOccupied);
}
```

removeLines() –
No parameters

This class removes full rows on the Tetris board and increments the line sent label. After a line is removed, its data is replaced by the one above and the step repeats until it reaches to top. Therefore, I will check if the swap is accurate by outputting using if statement to determine if the tileDatas are the same before and after the swap.

```
for ( each row ) {
        for ( each column ) {
                // Check if current line is the one below the deleted line
                if( currentLine == deletedLine -1 ) {
                        // If it is the line directly below deleted line, check if current
                        line equals the line above the deleted
                        if( tile[originalX+1][originalY-1] == tile[currentX][currentY]) {
                                System.out.println("swap error on X : " + currentX + "
                                Y:" + currentY);
                        }
                        // If not the line directly below deleted line, check if equals line
                        above
                        else {
                        if( tile[originalX][originalY] == tile[currentX][currentY]) {
                                System.out.println("swap error on X : " + currentX + "
                                Y:" + currentY);
                        }
                }
        }
```

```
}
```

keyboardListener – This class provides the keyboardListener that allows the game to respond to keyboard commands. To test this, I will output text corresponding to keyboard inputs.

```
        switch (keycode) {
            case KeyEvent.VK_LEFT:
                System.out.println("Clicked Left");
                break;
            case KeyEvent.VK_RIGHT:
                System.out.println("Clicked Right");
                break;
            case KeyEvent.VK_DOWN:
                System.out.println("Clicked Down");
                break;
            case KeyEvent.VK_UP:
                System.out.println("Clicked Up");
                break;
            case KeyEvent.VK_SPACE:
                System.out.println("Clicked Space Bar");
                break;
        }
```

MainGui Class

The purpose of this class is to provide the general layout of our game. It includes all the buttons, text areas, labels, and layouts. Even though GUI components provide their own visible output, it is easy to forget to set GUI components as visible. One of the simplest forms of testing GUI is to print out statements to see where actions take the user.

```
    btnCancel.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
```

```
                cardLayout.show(cardPanel, “Start”);
                System.out.println(“Now in Team Select”);
        }
```

DataBase Class

This class stores information created by our game. It stores usernames, passwords, emails, and scores. An easy way to test the database is simply by calling functions and seeing if it updates in the database. For example, I’ll add a new user to the database by calling our addUser() function and seeing if it updates.

```
addUser(“bob”, “usc”, “tommy@usc.edu”);

public static void addUser(String username, String password, String email) {
        try {
                pstmt = conn.prepareStatement(“INSERT INTO players(username,
                password, email, wins, numgames, sentlines) VALUES (?,?,?,?,?,?)”);
                pstmt.setString(1, username);
                pstmt.setString(2, password);
                pstmt.setString(3, email);
                pstmt.setInt(4, 0);
                pstmt.setInt(5, 0);
                pstmt.setInt(6, 0);
                pstmt.execute();
        } catch (SQLException e) {
                e.printStackTrace();
}
```

SQLException catch:

SQLException: Error in SQL

Fixes: - Make sure connection with SQL is correct

- Ensure that the correct tables, columns and rows exist

ChatServer Class

This class handles all the chat functions in our game. A way to test that the chat works is by accepting messages from the client and broadcasting them and ignoring clients that cannot be broadcasted to.

```
public void run(){
   while (true) {
       String input = in.readLine();
       if (input == null) {
           return;
       }
      for (PrintWriter writer : writers) {
           writer.println("MESSAGE " + name + ": " + input);
       }
    }
}

private void run() throws IOException {
// Make connection and initialize streams
   String serverAddress = "localhost";
   Socket socket = new Socket(serverAddress, 9001);
   in = new BufferedReader(new InputStreamReader(
        socket.getInputStream()));
   out = new PrintWriter(socket.getOutputStream(), true);
}

private void chat() throws IOException {
   // Process all messages from server, according to the protocol.
   while (true) {
      String line = in.readLine();
          if (line.startsWith("MESSAGE")) {
```

```
                chatTextArea.append(line.substring(8) + "\n");
        }
    }
}
```

IOException catch:

fixes: Print out at intervals throughout the method to catch errors such as punctuation or incorrect character types.

Deployment

Steps for the Server

- Start the server

Steps for the Client:

- Connect to the internet
- Import project into Eclipse
    - Clone project from the Git repository or download the project from Blackboard.
- Fix any build path errors if they exist
- Start the program and enjoy!