COP4331 Object Oriented Design and Programming

3 Dimensional Tic-Tac-Toe Game
with Chat Feature

Group Number: 23

Professor: Ionut Cardei

Team Members:
Bryan Barreto, Abir Faisal, Jamahl Farrington, Ruth Jimenez

Deliverable 3
Due: 12/07

Table of Contents

Application Requirements

@Authors Abir Fasial, Byran Barreto
Scenario: User Application, the software is used by end users as a desktop GUI application.

Use Case 1 – Application – Single Player

Description: Application is launched, and a user plays against a computer algorithm.

Actors: User
Preconditions:

a. User is present
b. Graphical User Interface (GUI) is available
c. Application is running

Flow:

1. Application presents options to the User
2. User selects the ‘Single Player’ option
3. Application assigns a SinglePlayerController Class to the GUI
4. User plays against an algorithm implemented by the Class
5. Application checks if there is a winner/tie after each move.

Terminations:

a. User Wins
    1. Application informs User that it has won the game
    2. Application asks User if they would like to play again
b. Computer Wins
    1. Application informs User that it has lost the game
    2. Application asks User if they would like to play again
c. Game is tied
    1. Application informs User that the game is tied
    2. Application asks User if they would like to play again

@Authors Abir Fasial, Byran Barreto
Use Case 2 – Application – Single Client Host

Description: Application is launched and used as a host for a single client to join as player 2.

Actors:

a. User1
b. User2

Preconditions:

a. User1 is present
b. User2 is present
c. User1 is connected to a network
d. User2 is connected to a network
e. Graphical User Interface (GUI) is available
f. Application is running

Flow:

1. Application presents options to the User1
2. User1 selects the ‘Host Game’ option
3. Application assigns a SingleHostController Class to the GUI
4. Application displays connection information and waits for another user to join it
5. User2's instance of Application presents options to the User2
6. User2 selects the ‘Multiplayer’ option
7. User2 enters connection information related to User1's instance.
8. Application assigns a MultiPlayerClientController Class to the GUI using the connection information.
9. User1 plays against User2 over a computer network
10. Host Application (User1) checks if there is a winner/tie after each move.

Terminations:

d. User1 Wins

   1. Application informs User1 that it has won the game
   2. Application informs User2 that it has lost the game
   3. Application asks User1 and User2 if they would like to play again

e. User2 Wins

   1. Application informs User2 that it has won the game
   2. Application informs User1 that it has lost the game
   3. Application asks User1 and User2 if they would like to play again

f. Game is tied

   1. Application informs User1 and User2 that the game is tied
   2. Application asks User1 and User2 if they would like to play again

@Authors Abir Fasial, Byran Barreto
Use Case 3 – Application – Multiplayer Client

Description: Application is launched and used as a client and joins a host.

Actors: User, Server
Preconditions:

a. User is present
b. Server is present
c. Graphical User Interface (GUI) is available
d. Application is running

Flow:

1. Application presents options to the User
2. User selects the ‘Join Server’ option
3. The application allows the user to connect to the default server or a server or host of their choice.
4. Application connects to the server as a client.
5. Server matches User with another client (User2).
    a. If there are no other players on the server, the client will play against an algorithm implemented by the server.
    b. If there is an odd number of of players, the client will play against an algorithm implemented by the server.
6. Server tracks each set of players and the state of each game.
7. Server ingests actions from each client.
8. Server interprets the state of each game and informs clients of wins, losses, and ties.

Terminations:

a. User disconnects
    i. The server will attempt to match the User with a different User.
b. Server(host) disconnects
    i. The clients will attempt to recover the connection.
c. Server runtime error
    i. The server will restart.

@Authors Abir Fasial, Byran Barreto
Scenario: Server. The software is used in a server environment to host players as clients.

Use Case 4 — Server — Multiplayer Host

Description: Program is run as a game server and used as a host.

Actors: Admin

Preconditions:

a. Command line is available

Flow:

1. Program is launched in server mode using the --server command line option.
2. Program presents itself as a server on the network
3. A number of players connect to the server
4. Server matches players
   a. If a player cannot be matched it will play against an algorithm implemented by the program.
5. Server tracks each set of players and the state of each game
6. Server ingests actions from each client and informs corresponding clients of said actions clients.
7. Server interprets the state of each game and informs clients of wins, losses, and ties.

CRC Cards

@Authors Abir Fasial

class, responsibilities, collaborators

Main	
load settings launch App or Server	App Server SettingsManager

package app/

App	
setup the main window initialize the MVC	Model View Controller

abstract class Model	
manage data for a view notify view of changes	View

abstract class View	
user input notify controller of input	Controller

abstract class Controller	
update the model	Model

package startscreen/

StartScreenModel extends Model	
define data keys store data values store server info	View

StartScreenView extends View	
show the start screen user selects game type user inputs server if needed	StartScreenController

StartScreenController extends Controller	
tells App what type of game to launch handle user interaction	App

package game/

GameModel	
define data keys store data values store gamestats	GameView

GameView	
define UI elements	GameControllers

enum GameType	
Provide strategy pattern Single,multi, or host game	App

SinglePlayerGameController	
run game internally handle game logic handle game input	GameView GameModel

MultiPlayerClientController	
handle game input send input to server receive from server	GameView GameModel

MultiPlayerHostController	
handle game input provide server for other player handle game logic	GameView GameModel

package util/

Solver	
provide a solver for the game	GameControllers Server

SettingsManager	
Load settings from file Save settings to file	Everything

package server/

Server	
provide a way for clients to connect and communicate	Clients

package chat/

ChatModel	
hold chat messages	

ChatView	
Display chat messages input new messages	

ChatController	
Handle the Chat window	ChatBotController ChatClientController

ChatBotController	
respond to chat messages in single player mode	

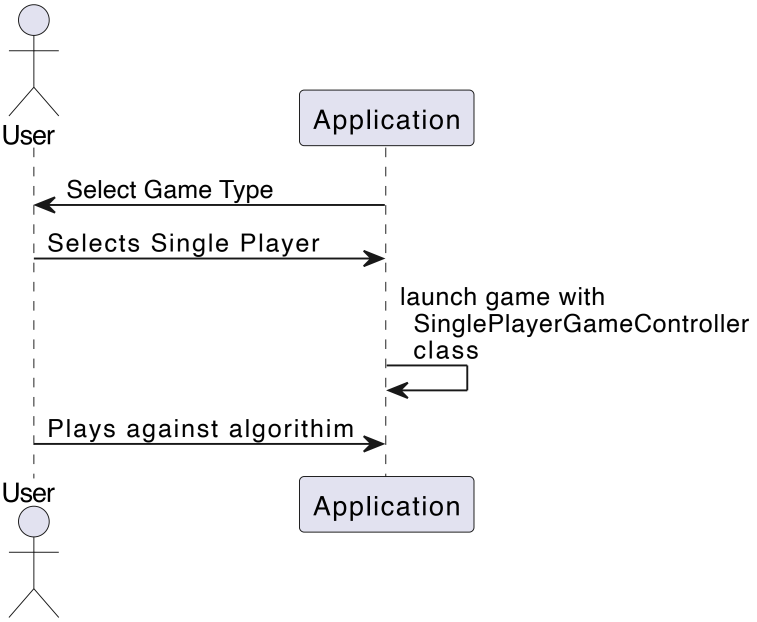
ChatClientController	
send receive messages from server	

Sequence Diagrams for scenarios

Description: Sequence diagrams for our use case scenarios

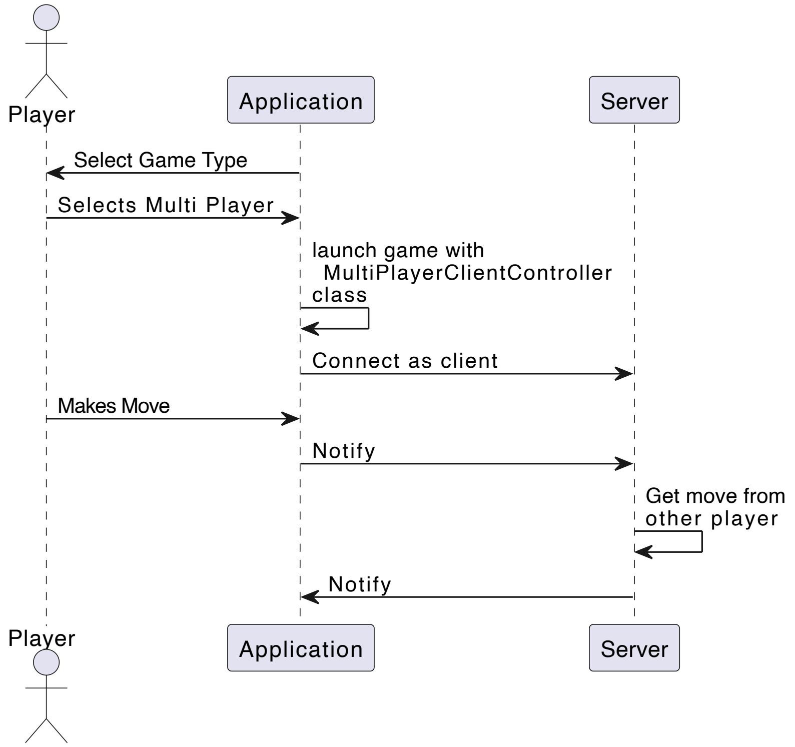
Author: Abir Faisal

Use Case 1 - Application - Single Player



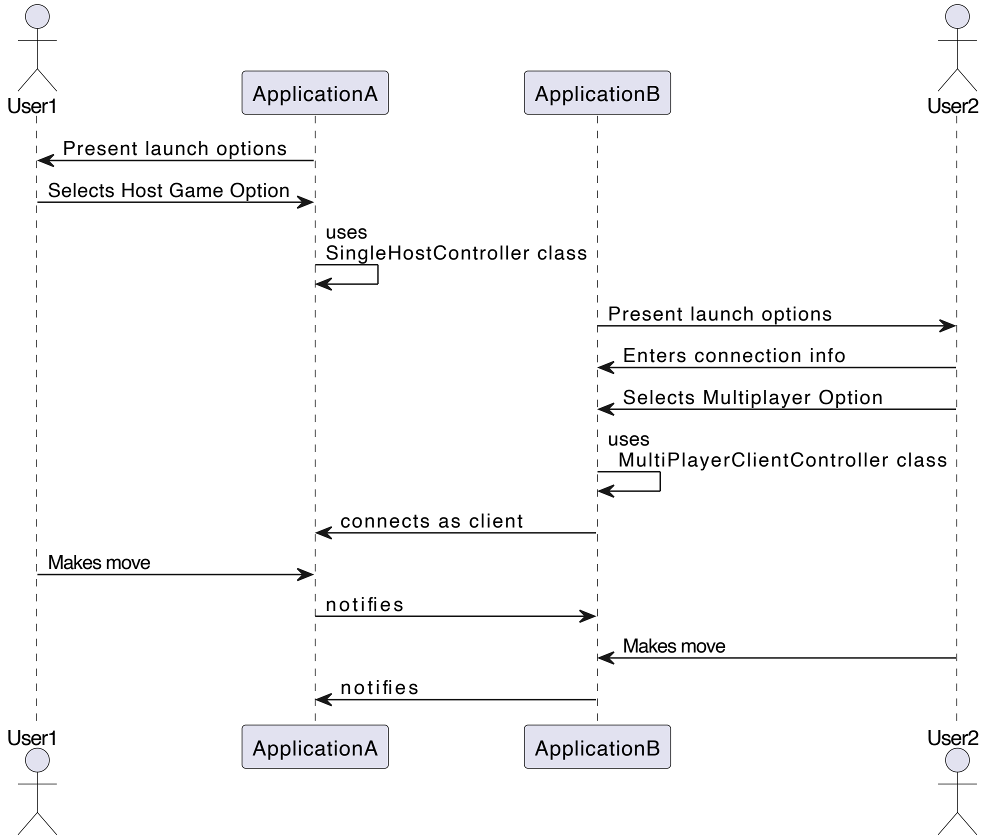
Author: Abir Faisal

Use Case 3 - Application - Multiplayer Client



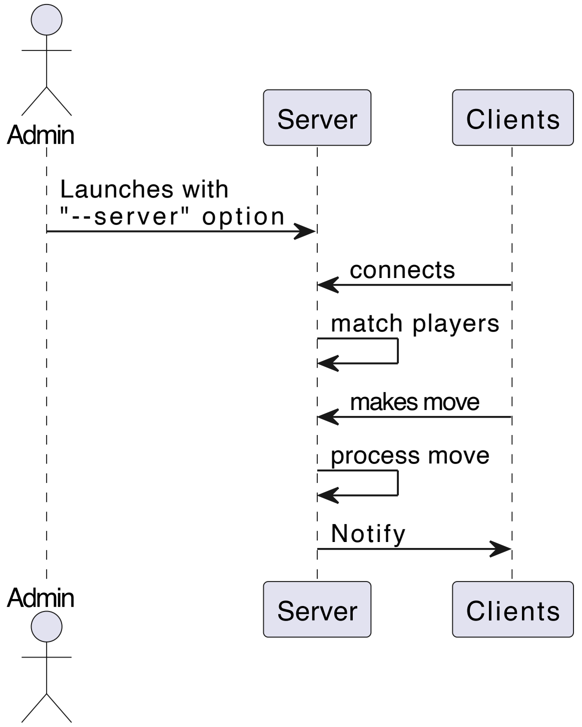
Author: Abir Faisal

Use Case 2 - Application - Single Client Host



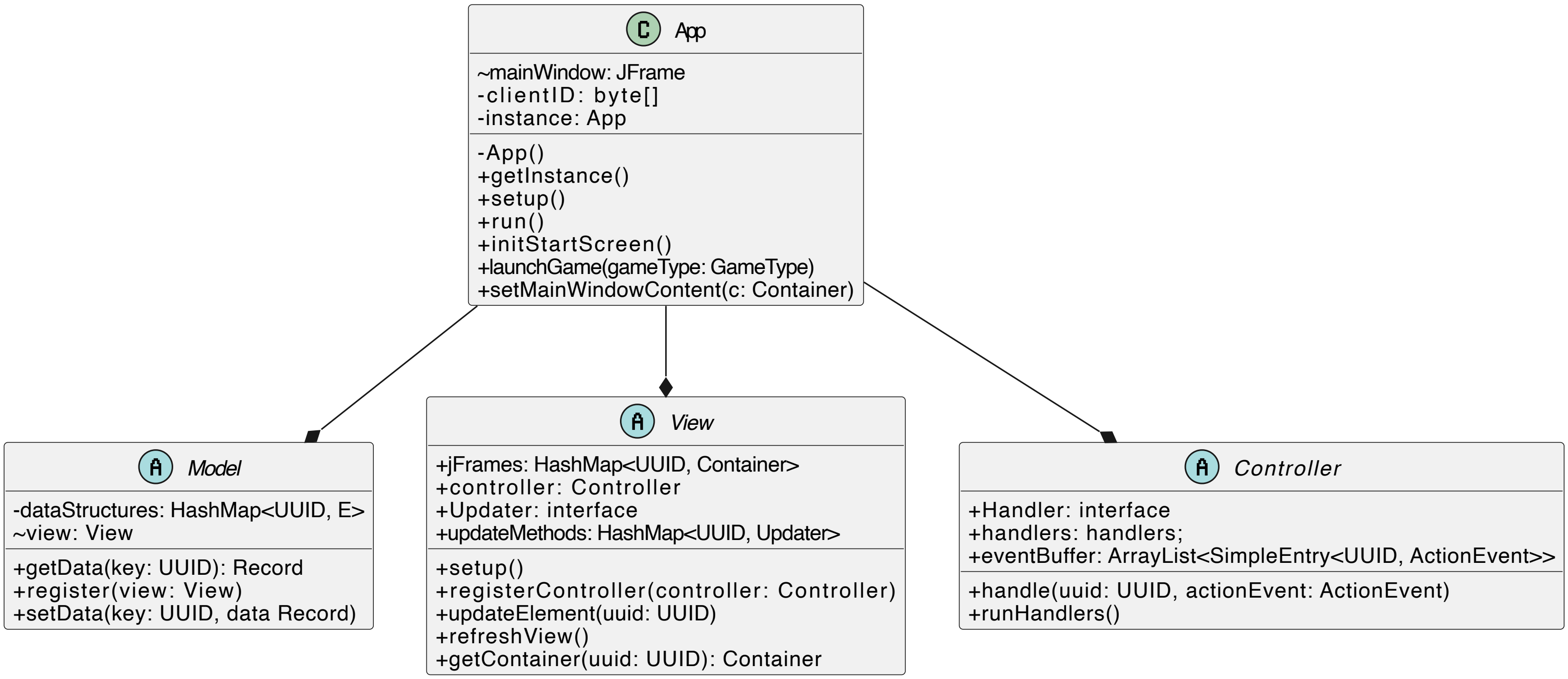
Author: Abir Faisal

Use Case 4 - Server - Multiplayer Host



UML Diagrams

Author: Abir Faisal

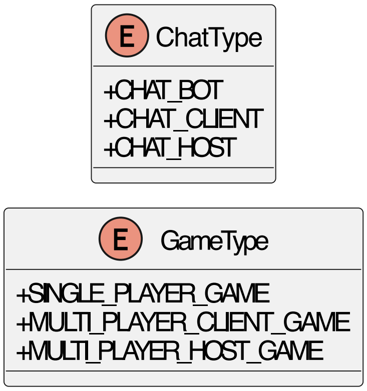

C App
~mainWindow: JFrame
-clientID: byte[]
-instance: App
-App()
+getInstance()
+setup()
+run()
+initStartScreen()
+launchGame(gameType: GameType)
+setMainWindowContent(c: Container)
A Model
-dataStructures: HashMap<UUID, E>
~view: View
+getData(key: UUID): Record
+register(view: View)
+setData(key: UUID, data Record)
A View
+jFrames: HashMap<UUID, Container>
+controller: Controller
+Updater: interface
+updateMethods: HashMap<UUID, Updater>
+setup()
+registerController(controller: Controller)
+updateElement(uuid: UUID)
+refreshView()
+getContainer(uuid: UUID): Container
A Controller
+Handler: interface
+handlers: handlers;
+eventBuffer: ArrayList<SimpleEntry<UUID, ActionEvent>>
+handle(uuid: UUID, actionEvent: ActionEvent)
+runHandlers()

Description: The App class is implimented as a singleton pattern

It declares a Model, View, and Controller.

When the application is launched it initializes an empty window and puts whatever View type the programmer specifies into the mainWindow JFrame

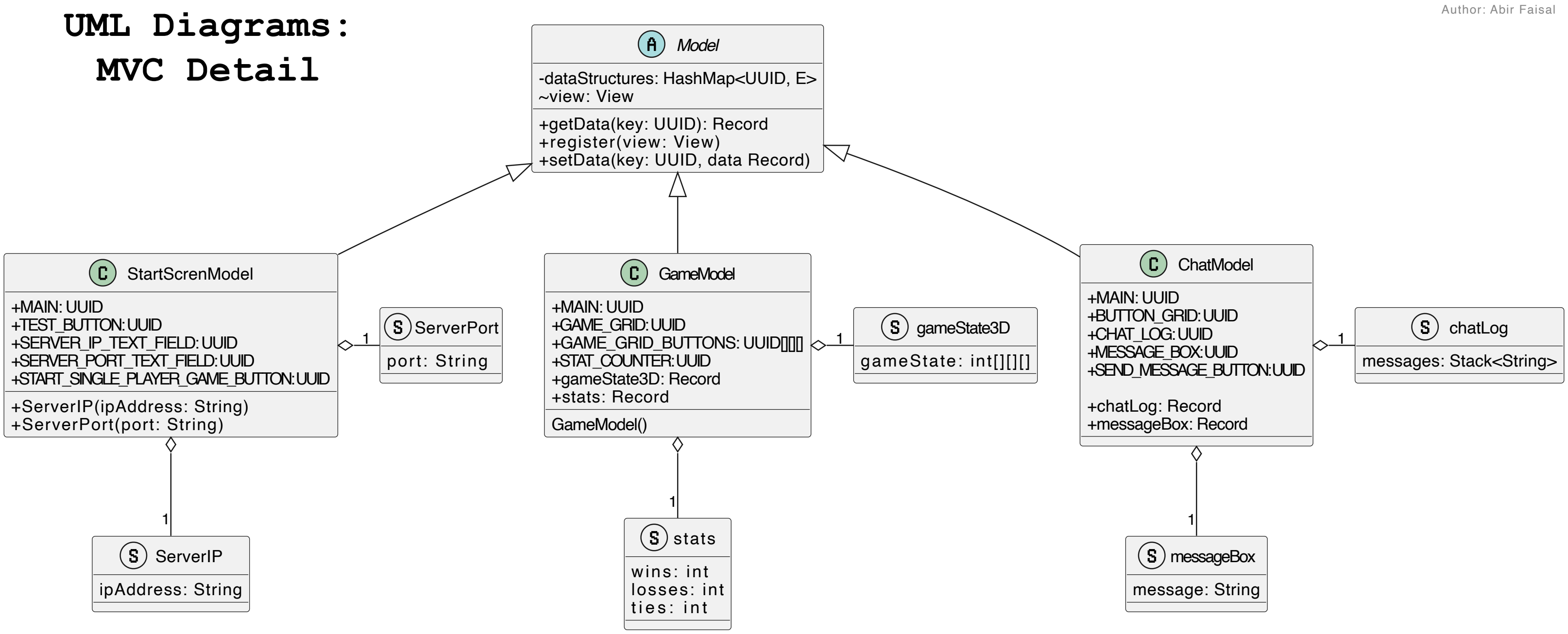
More detail on next page



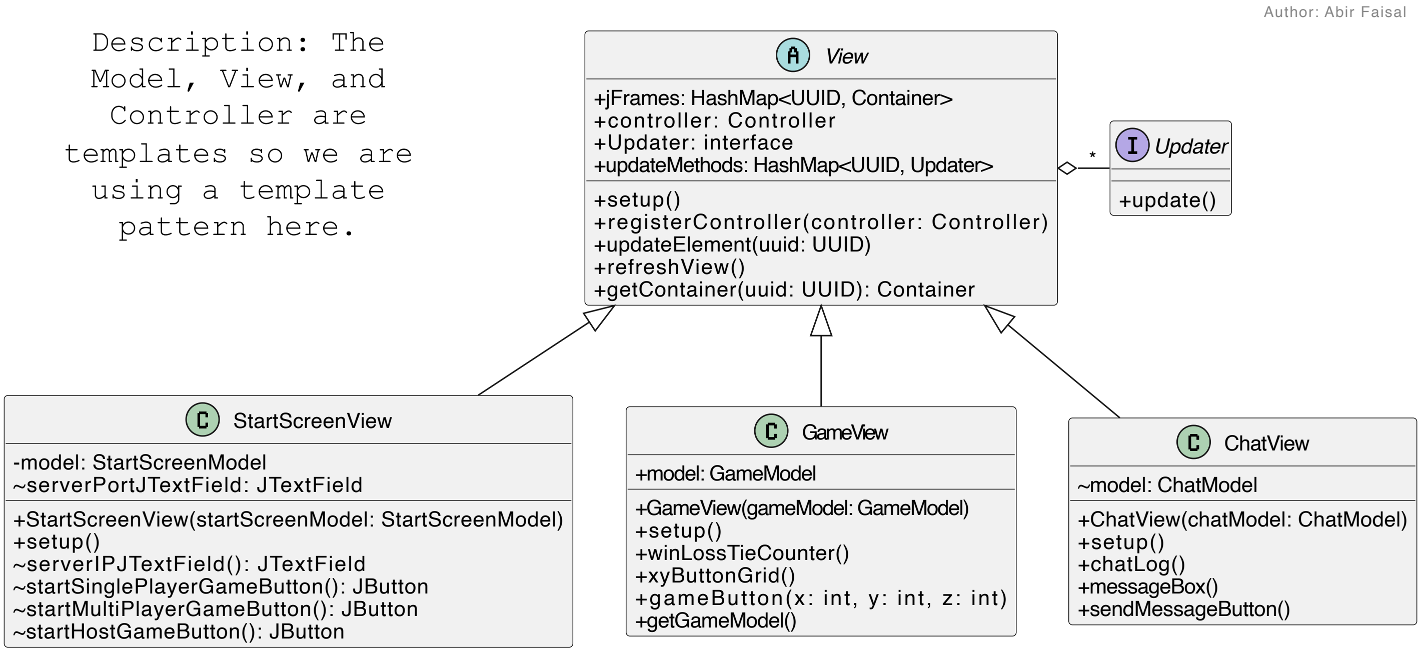
Description: These Enums are used by the launch controller to tell the App what type of Game and Chat to launch.

This is a strategy pattern.

UML Diagrams:
MVC Detail



Description: The Model, View, and Controller are templates so we are using a template pattern here.

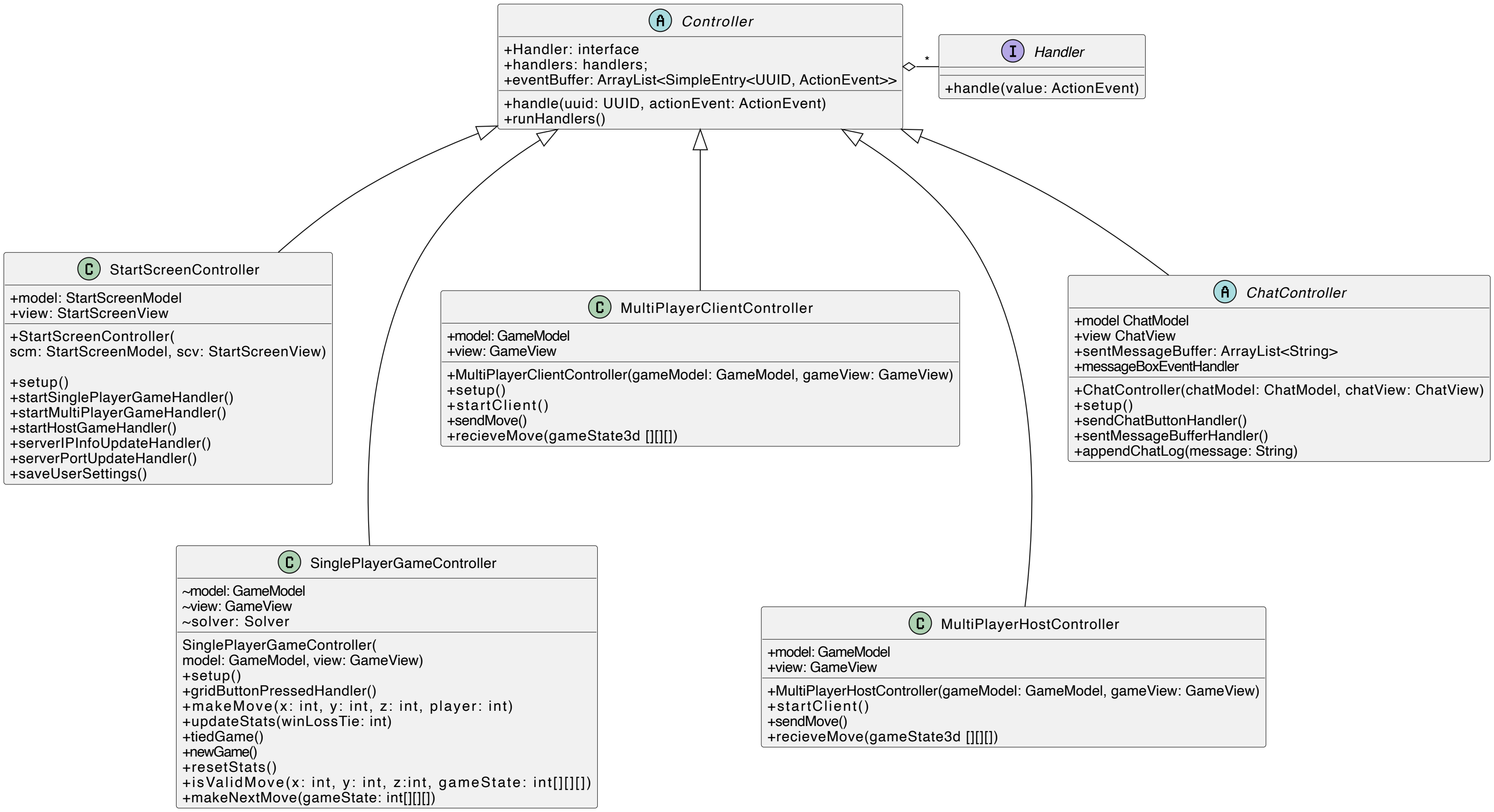

Author: Abir Faisal
A View
+jFrames: HashMap<UUID, Container>
+controller: Controller
+Updater: interface
+updateMethods: HashMap<UUID, Updater>
+setup()
+registerController(controller: Controller)
+updateElement(uuid: UUID)
+refreshView()
+getContainer(uuid: UUID): Container
*
I Updater
+update()
C StartScreenView
-model: StartScreenModel
~serverPortJTextField: JTextField
+StartScreenView(startScreenModel: StartScreenModel)
+setup()
~serverIPJTextField(): JTextField
~startSinglePlayerGameButton(): JButton
~startMultiPlayerGameButton(): JButton
~startHostGameButton(): JButton
C GameView
+model: GameModel
+GameView(gameModel: GameModel)
+setup()
+winLossTieCounter()
+xyButtonGrid()
+gameButton(x: int, y: int, z: int)
+getGameModel()
C ChatView
~model: ChatModel
+ChatView(chatModel: ChatModel)
+setup()
+chatLog()
+messageBox()
+sendMessageButton()

Controller Detail on next page

UML Diagrams: MVC Detail

Description: The Model, View, and Controller details continued

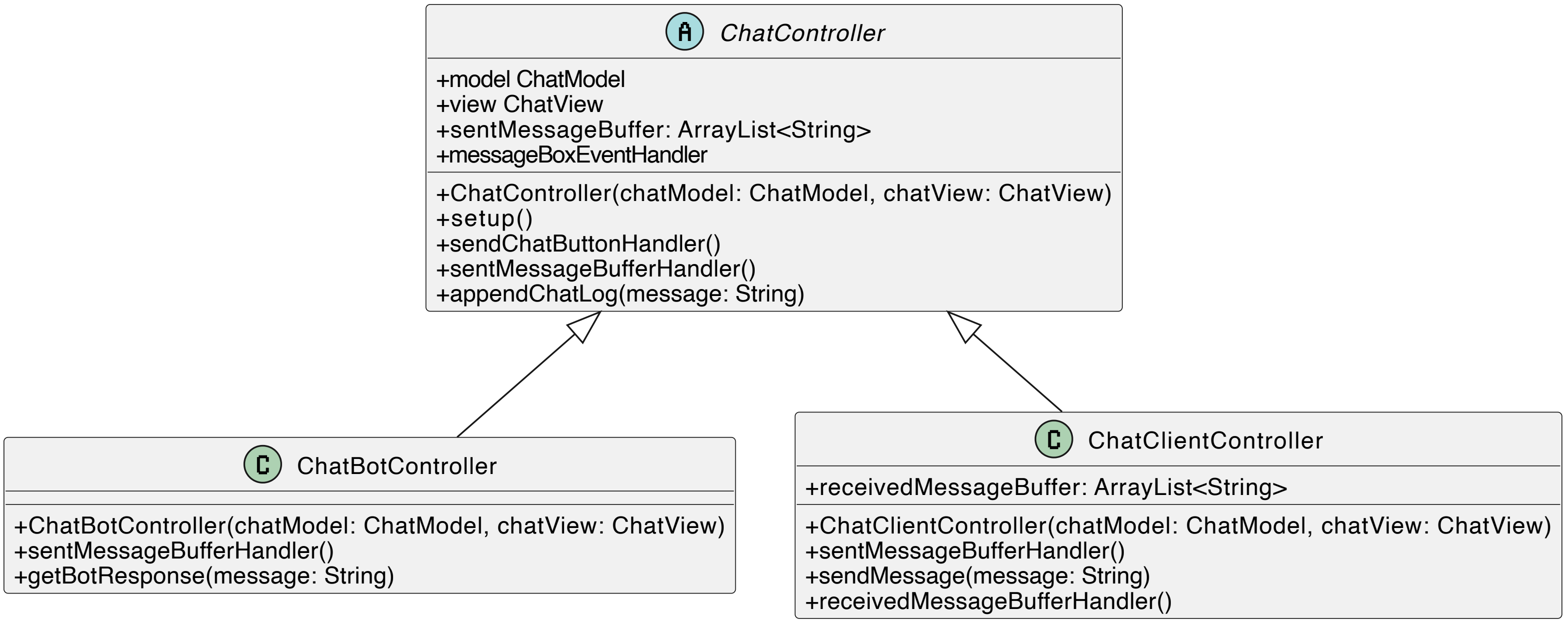
Author: Abir Faisal


A Controller
+Handler: interface
+handlers: handlers;
+eventBuffer: ArrayList<SimpleEntry<UUID, ActionEvent>>
+handle(uuid: UUID, actionEvent: ActionEvent)
+runHandlers()
*
I Handler
+handle(value: ActionEvent)
C StartScreenController
+model: StartScreenModel
+view: StartScreenView
+StartScreenController(
scm: StartScreenModel, scv: StartScreenView)
+setup()
+startSinglePlayerGameHandler()
+startMultiPlayerGameHandler()
+startHostGameHandler()
+serverIPInfoUpdateHandler()
+serverPortUpdateHandler()
+saveUserSettings()
C MultiPlayerClientController
+model: GameModel
+view: GameView
+MultiPlayerClientController(gameModel: GameModel, gameView: GameView)
+setup()
+startClient()
+sendMove()
+recieveMove(gameState3d [][][])
A ChatController
+model ChatModel
+view ChatView
+sentMessageBuffer: ArrayList<String>
+messageBoxEventHandler
+ChatController(chatModel: ChatModel, chatView: ChatView)
+setup()
+sendChatButtonHandler()
+sentMessageBufferHandler()
+appendChatLog(message: String)
C SinglePlayerGameController
~model: GameModel
~view: GameView
~solver: Solver
SinglePlayerGameController(
model: GameModel, view: GameView)
+setup()
+gridButtonPressedHandler()
+makeMove(x: int, y: int, z: int, player: int)
+updateStats(winLossTie: int)
+tiedGame()
+newGame()
+resetStats()
+isValidMove(x: int, y: int, z:int, gameState: int[][][])
+makeNextMove(gameState: int[][][])
C MultiPlayerHostController
+model: GameModel
+view: GameView
+MultiPlayerHostController(gameModel: GameModel, gameView: GameView)
+startClient()
+sendMove()
+recieveMove(gameState3d [][][])

UML Diagrams: MVC Detail
Chat Feature

Description: The chat feature was not a part of our original design, but we thought it would show how our MVC architecture is extensible, you can just extend the MVC classes and make your own thing. It doesn't have to be a game it can be anything.

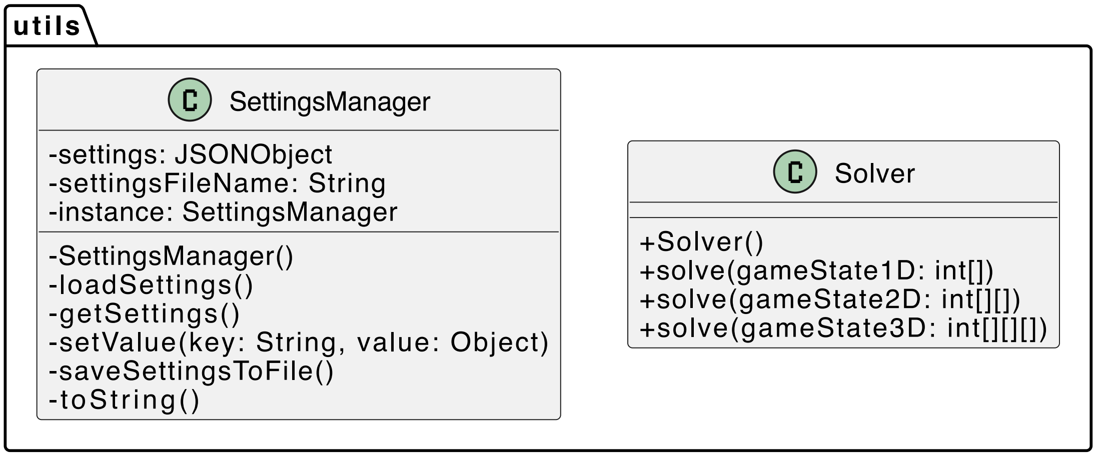
Author: Abir Faisal



A ChatController
+model ChatModel
+view ChatView
+sentMessageBuffer: ArrayList<String>
+messageBoxEventHandler
+ChatController(chatModel: ChatModel, chatView: ChatView)
+setup()
+sendChatButtonHandler()
+sentMessageBufferHandler()
+appendChatLog(message: String)
C ChatBotController
+ChatBotController(chatModel: ChatModel, chatView: ChatView)
+sentMessageBufferHandler()
+getBotResponse(message: String)
C ChatClientController
+receivedMessageBuffer: ArrayList<String>
+ChatClientController(chatModel: ChatModel, chatView: ChatView)
+sentMessageBufferHandler()
+sendMessage(message: String)
+receivedMessageBufferHandler()

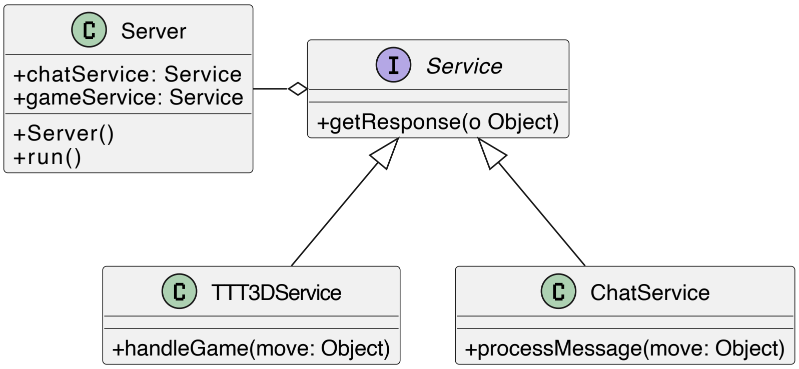
UML Diagrams: Other

Authors: Abir Faisal



utils
C SettingsManager
-settings: JSONObject
-settingsFileName: String
-instance: SettingsManager
-SettingsManager()
-loadSettings()
-getSettings()
-setValue(key: String, value: Object)
-saveSettingsToFile()
-toString()
C Solver
+Solver()
+solve(gameState1D: int[])
+solve(gameState2D: int[][])
+solve(gameState3D: int[][][])

Description: These are utilities that various parts of the program can use as needed.
The solver can check the game for a winner, and the SettingsManager loads and saves program settings from file.



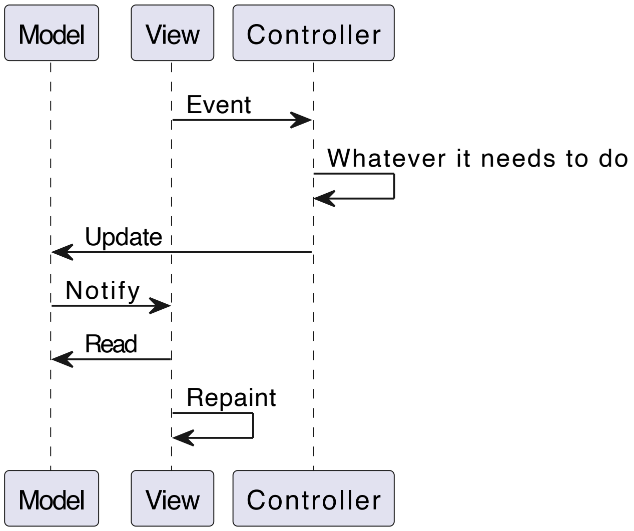
Description: The server is suppose to provide services for clients.

We didn't get to it but basically the idea was that the server could provide any service to any client as long as there exists a service handler.

Sequence Diagrams for program

Author: Abir Faisal

Model View Controller

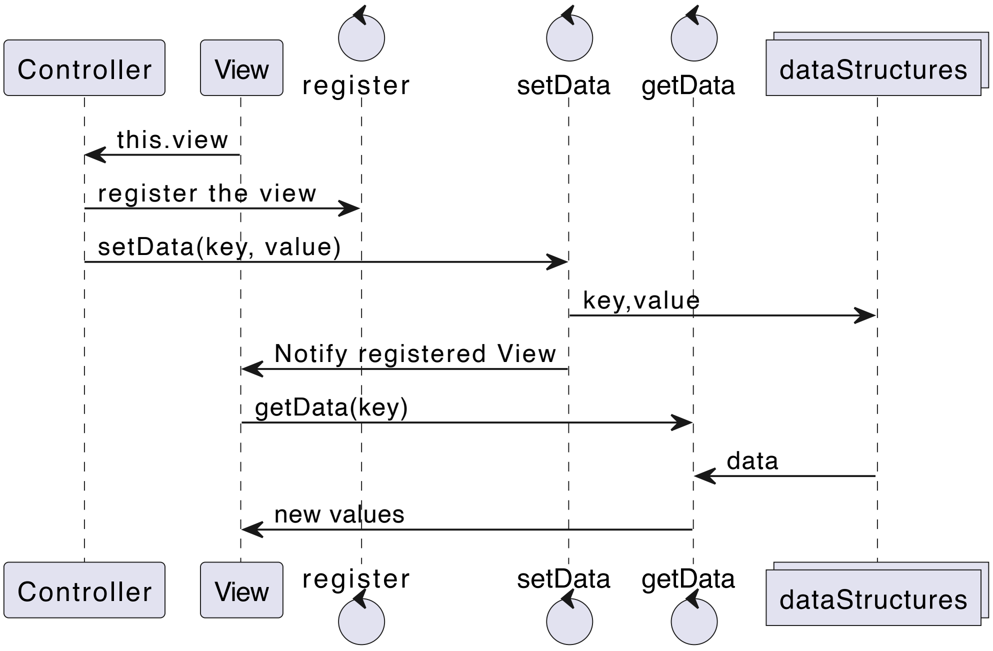


Description: The MVC components interact with each other using UUIDs that are defined in models that extend the abstract Model. (Template pattern)

The templates contain everything needed forthe MVC to work. You just have to define the UI components in your View subclass, event handlers in your Controller subclass, and UUID constants in the Model as well as data strctures in the form of record classes.

Author: Abir Faisal

How Controller and View interacts with Model

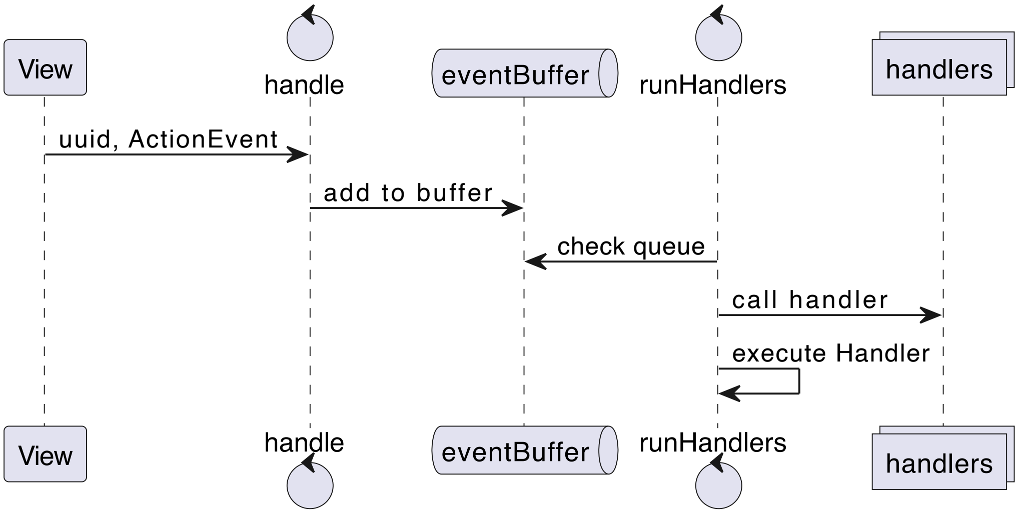


Description: A View is registered to a Model. When the model is updated by the Controller the view is notified that the model has changed. From there the view will update itself from the model.

Sequence Diagrams for program continued

Author: Abir Faisal

How View interacts with Controller

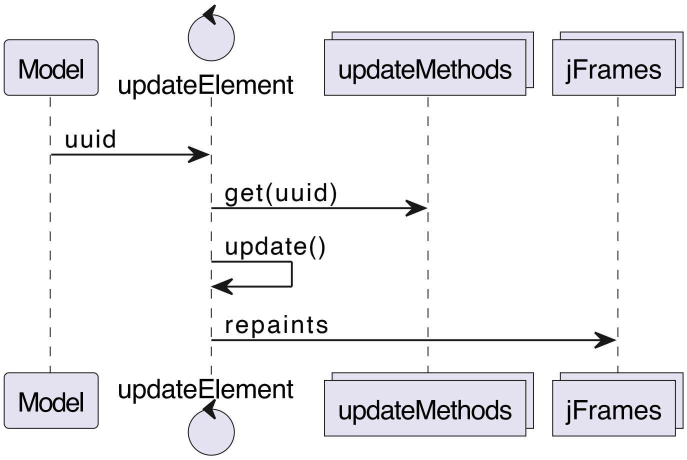


Description: When an event occurs in the view the Controller checks the handlers HashMap to see if it contains a corresponding Handler.

If so then the handler is executed.

Author: Abir Faisal

How View updates UI elements from Model

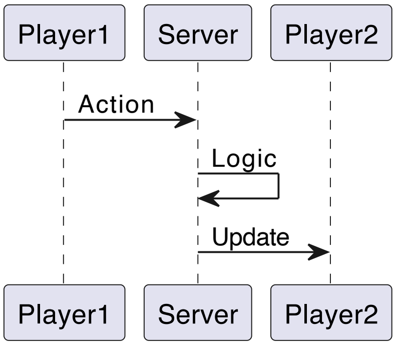


Description: Each Swing component has an by declaring an new Updater with the Updater interface and putting it into the updateMethods hashmap.

When the model notifies the view the View calls the corresponding update method executes

Author: Abir Fasial

Server



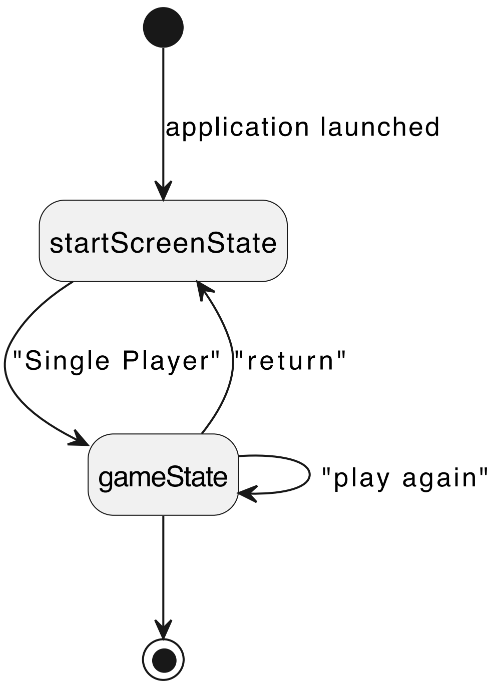
Description: Server We didn't get to this but basically it's a server that would have players connect to it and it would mediate between them.

State Diagrams:

Description: General Application State Diagrams

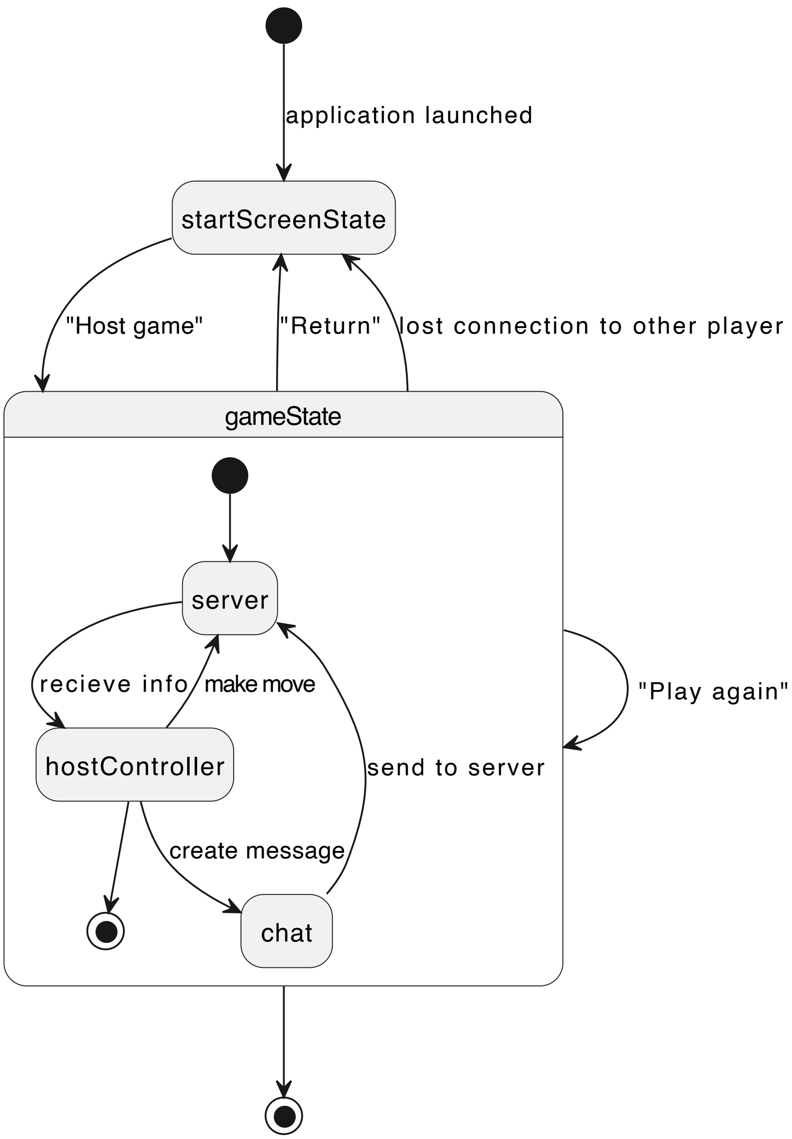
Author: Bryan Barreto

Single Player



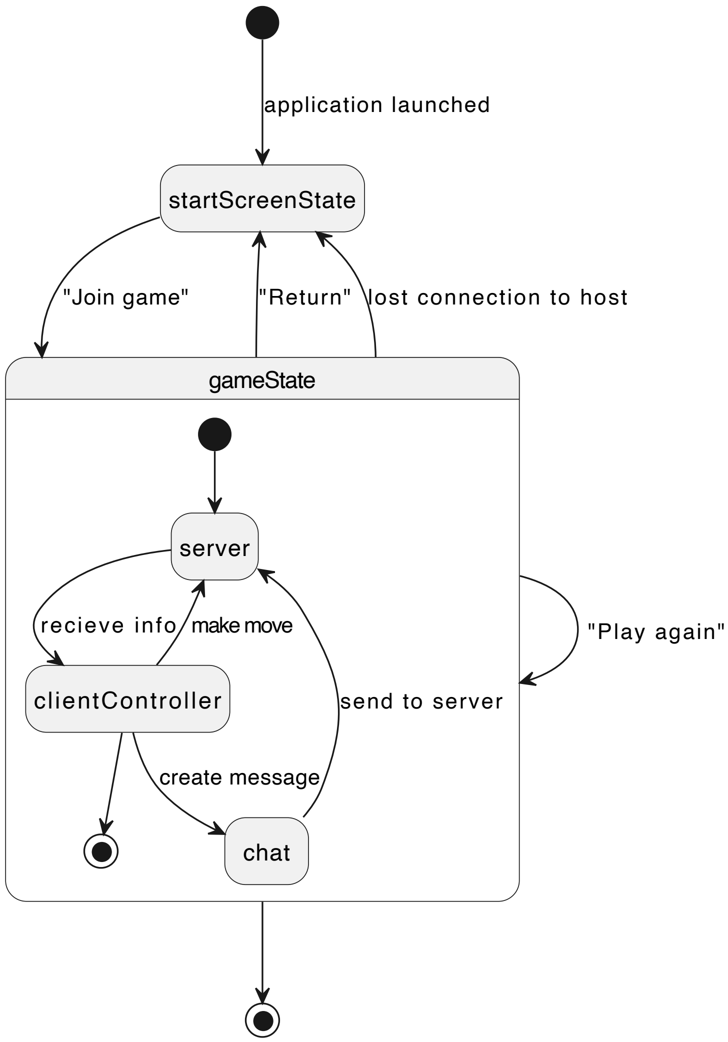
Author: Bryan Barreto

Multi Player Host



Author: Bryan Barreto

Multi Player Join



```java
package edu.fau.eng.cop4331.ttt3d;

import edu.fau.eng.cop4331.ttt3d.app.App;
import edu.fau.eng.cop4331.ttt3d.server.Server;
import edu.fau.eng.cop4331.ttt3d.util.SettingsManager;

import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

public class Main {
    public static void main(String[] args) throws IOException {
        Map<String, Integer> argmap = new HashMap<>();

        //interpret command line arguments
        //For example --f0 1234 --f1 5678
        for (int i = 0; i < args.length; i++) {
            String argument = args[i];

            if (argument.startsWith("--")) {
                String key = argument.substring(2); //remove -- from key
                String value = args[i+1]; //get value of key
                argmap.put(key, Integer.parseInt(value)); //put key and value into map for us
                System.out.println(key + "=" + value);//TODO remove when no longer needed.
            }
        }


        //load settings
        SettingsManager settingsManager = SettingsManager.getInstance();
        settingsManager.loadSettings();

        //if --server then launch game server instead of user application
        if (argmap.get("--server") != null) {
            System.out.println("Running Server");
            //port for server

            Server server = new Server();
            server.run();

            //TODO ip and port for load balance and failover

            //TODO Server server = new Server(port, secondaryServerIP, );
            //server.run();
        } else {
            System.out.println("Launch Start Screen");
            App instance = App.getInstance();
            instance.setup();
            instance.run();
        }
    }
}
```

```java
package edu.fau.eng.cop4331.ttt3d.app;

import edu.fau.eng.cop4331.ttt3d.app.chat.*;
import edu.fau.eng.cop4331.ttt3d.app.game.*;
import edu.fau.eng.cop4331.ttt3d.app.startscreen.StartScreenController;
import edu.fau.eng.cop4331.ttt3d.app.startscreen.StartScreenModel;
import edu.fau.eng.cop4331.ttt3d.app.startscreen.StartScreenView;
import edu.fau.eng.cop4331.ttt3d.util.SettingsManager;
import org.json.JSONArray;

import javax.swing.*;
import java.awt.*;
import java.util.Random;


public class App {
    JFrame mainWindow;
    private byte[] clientID; //128 bit client id

    //Singleton Pattern
    private static App instance;
    private App() {
        this.mainWindow = new JFrame("TTT3D");
        this.clientID = getClientID();
    }
    public static synchronized App getInstance() {
        if (instance == null) instance = new App();
        return instance;
    }

    /**
     * Set up the components of the main window and/or application
     * @author Abir Faisal
     */
    public void setup() {
        initStartScreen();
        this.mainWindow.setSize(800,600);//400 width and 500 height
        this.mainWindow.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        ///TODO this.mainWindow.setLayout();
        this.mainWindow.setVisible(true);
    }

    /**
     * run the application
     *
     * @author Abir Faisal
     */
    public void run() {
        //TODO this might be useless...
    }

    /**
     * Setup the initial MVC you want to show the user
     *
     * @author Abir Faisal
     */
    public void initStartScreen(){
        StartScreenModel startScreenModel = new StartScreenModel();
        StartScreenView startScreenView = new StartScreenView(startScreenModel);
        StartScreenController startScreenController = new StartScreenController(startScreenMo
        setMainWindowContent(startScreenView.getContainer(startScreenModel.MAIN));
    }

    /**
     *  Launch the application specified by the initial screen
     *
     * @author Abir Faisal
     * @param gameType the type of game you want to launch
     */
    public void launchGame(GameType gameType) {
        GameModel gameModel = new GameModel();
        GameView gameView = new GameView(gameModel);

        ChatModel chatModel = new ChatModel();
        ChatView chatView = new ChatView(chatModel);

        switch (gameType) {
            case SINGLE_PLAYER_GAME -> {
                SinglePlayerGameController gameController = new SinglePlayerGameController(ga
                ChatController chatController = new ChatBotController(chatModel, chatView);

                //show game and chat side by side
                JSplitPane jSplitPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT);
                jSplitPane.add(gameView.getContainer(gameModel.MAIN));
                jSplitPane.add(chatView.getContainer(chatModel.MAIN));

                setMainWindowContent(jSplitPane);
            }
            case MULTI_PLAYER_CLIENT_GAME -> {
                MultiPlayerClientController gameController = new MultiPlayerClientController
                ChatController chatController = new ChatClientController(chatModel, chatView

                //TODO ChatClientController

                //show game and chat side by side
                JSplitPane jSplitPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT);
                jSplitPane.add(gameView.getContainer(gameModel.MAIN));
                jSplitPane.add(chatView.getContainer(chatModel.MAIN));

                setMainWindowContent(jSplitPane);            }
            case MULTI_PLAYER_HOST_GAME -> {}
        }

        System.out.println("Launching Game " + gameType);
    }


    /**
     * Generate a client ID or try to load from settings
     *
     * @author Abir Faisal
     * @return 128bit Client ID as byte[16], 16 * 8bit = 128bits
     */
    public byte[] getClientID() {
        if (this.clientID == null) {
            this.clientID = new byte[16];
            SettingsManager sm = SettingsManager.getInstance();

            //if no clientID in settings.json then generate and save
            //else load from configureation
            if (sm.getSettings().opt("clientID") == null) {
                Random r = new Random();
                r.nextBytes(this.clientID);
                //save to settings
                sm.setValue("clientID", this.clientID);
            } else {
                //load from settings
                JSONArray clientIDJSONArray = sm.getSettings().getJSONArray("clientID");

                for (int i = 0; i < clientIDJSONArray.length(); i++) {
                    this.clientID[i] = (byte) clientIDJSONArray.getInt(i);
                }
            }
        }
        return clientID;
    }

    /**
     * set the content of the main window, replace existing content
     *
     * @author Abir Faisal
     * @param c a JPanel that contains the contents you want to display
     */
    public void setMainWindowContent(Container c) {
        this.mainWindow.getContentPane().removeAll();
        this.mainWindow.setContentPane(c);
        this.mainWindow.revalidate();
    }

    /**
     * add the content to the main window
     *
     * @author Abir Faisal
     * @param c a JPanel that contains the contents you want to display
     */
    public void addMainWindowContent(Container c) {
        this.mainWindow.add(c);
        this.mainWindow.revalidate();
    }

}


```

```java
package edu.fau.eng.cop4331.ttt3d.app;

import java.awt.*;
import java.util.HashMap;
import java.util.UUID;
import java.util.function.BiConsumer;

public abstract class View {


    //Objects of the view
    public HashMap<UUID, Container> jFrames = new HashMap<>();
    public Controller controller;


    //methods that are called when update is called on a UUID mapped to jFrames
    public HashMap<UUID, Updater> updateMethods = new HashMap<>();

    //TODO remove, it seems like its not used
//    public Model model;
//    public View(Model model) {
//        this.model = model;
//        this.model.register(this);
//    }

    /**
     * Used to setup the view, setup the main view and add elements to it
     * This should be called in the constructor
     *
     * @author Abir Faisal
     */
    public abstract void setup();


    //register a controller for the view

    /**
     * Registers a controller with the view so that the view
     * is aware of where it needs to send actions and events.
     * The view will call it's handle(UUID) method when soemthing happens.
     *
     * @author Abir Faisal
     * @param controller A subclass that extends the abstract Controller
     */
    public void registerController(Controller controller){
        this.controller = controller;
    }


    /**
     * Updates an element of the view given its corresponding UUID
     *
     * @author Abir Faisal
     * @param uuid UUID as defined in the model of the view
     */
    public void updateElement(UUID uuid) {
        if (this.updateMethods.get(uuid) != null)
            this.updateMethods.get(uuid).update();
    }

    /**
     * Refresh/Update the whole view.
     *
     * @author Abir Faisal
     */
    public void refreshView(){
        BiConsumer<? super UUID, ? super Updater> biConsumer = (uuid, updater) -> updater.up
        updateMethods.forEach(biConsumer);
    }

    /**
     * Get a component of the view
     *
     * @author Abir Faisal
     * @param uuid UUID as defined in the model of the view
     */
    public Container getContainer(UUID uuid){
        return this.jFrames.get(uuid);
    }

}

```

```java
package edu.fau.eng.cop4331.ttt3d.app;

import java.util.HashMap;
import java.util.UUID;

public abstract class  Model <E> {

    // Contains data structures that will be
    // updated by the controller or read by the view
    HashMap<UUID, E> dataStructures = new HashMap<>();

    //Just a refrence to the view that should be notified
    //when data is updated in of this model
    View view;

    /**
     * The view will request a dataStructure from the Model
     *
     * @author Abir Faisal
     * @param key UUID as defined in a subclass of this Model
     */
    public Record getData(UUID key) {
        return (Record) dataStructures.get(key);
    }

    /**
     * Register a view with the model so that setData()
     * can call its notify method after updating a value
     *
     * @author Abir Faisal
     * @param view the view that should be notified of changes to this model
     */
    public void register(View view) {
        this.view = view;
    }


    /**
     * Allows the controller to set/update a dataStructure
     * and the model to notify the view
     *
     * @author Abir Faisal
     * @param key UUID as defined in a subclass of this Model
     * @param data record object as defined a subclass of this Model
     */
    public synchronized void setData(UUID key, Record data) {
        if (dataStructures.containsKey(key)){
            //replace the object
            dataStructures.replace(key, (E) data);
            //notify the view that data has changed
            this.view.updateElement(key);
        } else {
            //add the object
            this.dataStructures.put(key, (E) data);

            //notify the view that data has changed
            this.view.updateElement(key);
        }
    }

}

```

```java
package edu.fau.eng.cop4331.ttt3d.app;

import java.awt.event.ActionEvent;

public interface Handler {
    void handle(ActionEvent value);
}

```

```java
package edu.fau.eng.cop4331.ttt3d.app;

/**
 * Updater interface.
 * the method update() is called when something in the view needs to be updated.
 *
 * @author Abir Faisal
 */
public interface Updater {
    void update();
}

```

```java
package edu.fau.eng.cop4331.ttt3d.app;

import java.awt.event.ActionEvent;
import java.util.AbstractMap.SimpleEntry;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.UUID;

import static java.lang.Thread.sleep;

public abstract class Controller {

    //Contains a set of UUID and handlers implimenting the Handler interface
    public HashMap<UUID, Handler> handlers = new HashMap<>();

    /**
     * When the user interacts with the View,
     * the View will notify the Controller that a (UUID, actionEvent) has occurred,
     * then the (UUID, ActionEvent) will go into a handlerBuffer
     * later it will be handled by a Thread launched by runHandlers().
     *
     * @author Abir Faisal
     *
     */

    public ArrayList<SimpleEntry<UUID, ActionEvent>> eventBuffer = new ArrayList<>();

    /**
     * passes events from the UI into the event buffer.
     * It is handled when the runHandlers thread checks it.
     *
     * @author Abir Faisal
     * @param uuid
     * @param actionEvent
     */
    public void handle(UUID uuid, ActionEvent actionEvent) {
        SimpleEntry<UUID, ActionEvent> tuple = new SimpleEntry<>(uuid, actionEvent);
        eventBuffer.add(tuple);
    }

    /**
     * This will monitor the event buffer and handle any events
     *
     * @author Abir Faisal
     */
    //TODO convert to iterator pattern
    public void runHandlers() {
        new Thread(() -> {
            while (true) {
                int i = 0;
                try {
                    for (i = 0; i < eventBuffer.size(); i++) {
                        //get the UUID and ActionEvent
                        SimpleEntry<UUID, ActionEvent> simpleEntry = eventBuffer.get(i);

                        //Handle the event
                        UUID uuid = simpleEntry.getKey();
                        ActionEvent actionEvent = simpleEntry.getValue();
                        handlers.get(uuid).handle(actionEvent);

                        //remove from buffer
                        eventBuffer.remove(i);
                    }
                    sleep(50); //prevent using CPU cycles for no reason.
                } catch (InterruptedException e) {
                    throw new RuntimeException(e);
                }
            }
        }).start();
    }
}

```

```java
package edu.fau.eng.cop4331.ttt3d.app.chat;

/**
 * The type of chat the App should use
 * when instantiating a controller for the view
 */
public enum ChatType {
    CHAT_BOT,
    CHAT_CLIENT,
    CHAT_HOST
}

```

```java
package edu.fau.eng.cop4331.ttt3d.app.chat;

import edu.fau.eng.cop4331.ttt3d.app.Updater;
import edu.fau.eng.cop4331.ttt3d.app.View;

import javax.swing.*;
import javax.swing.event.DocumentEvent;
import javax.swing.event.DocumentListener;
import javax.swing.text.DefaultCaret;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.time.Instant;
import java.util.ArrayList;
import java.util.Stack;
import java.util.UUID;

public class ChatView extends View {


    ChatModel model;

    /**
     * Constructor
     *
     * @param chatModel ChatModel
     */
    public ChatView(ChatModel chatModel){
        this.model = chatModel;
        this.model.register(this);
        setup();
    }


    /**
     * Setup the view
     */
    @Override
    public void setup() {
        JPanel mainJPanel = new JPanel();
        mainJPanel.setLayout(new BoxLayout(mainJPanel, BoxLayout.Y_AXIS));
        this.jFrames.put(this.model.MAIN, mainJPanel);


        this.jFrames.get(this.model.MAIN).add(chatLog());
        this.jFrames.get(this.model.MAIN).add(messageBox());
        this.jFrames.get(this.model.MAIN).add(sendMessageButton());
    }


    /////UI elements/////////
    /**
     * The chat log where the user can see the send and
     * recieved messages
     *
     * @author Abir Faisal
     * @return JScrollPane
     */
    JScrollPane chatLog() {
        UUID uuid = this.model.CHAT_LOG;

        JTextArea jTextArea = new JTextArea("");
        jTextArea.setEditable(false);
        DefaultCaret dc = (DefaultCaret) jTextArea.getCaret();
        dc.setUpdatePolicy(DefaultCaret.ALWAYS_UPDATE);

        JScrollPane jScrollPane = new JScrollPane(jTextArea);
        jScrollPane.setPreferredSize(new Dimension(800,600));
        jScrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);


        Updater updater = () -> {
            //get record from model
                ChatModel.chatLog cl =
                        (ChatModel.chatLog) this.model.getData(uuid);
            Stack<String> messages = cl.messages();
            jTextArea.append(messages.peek() + "\n\n");
        };
        updateMethods.put(uuid, updater);

        return jScrollPane;
    }

    /**
     * The message box where the user
     * types in a message that they want to send.
     *
     * @author Abir Faisal
     * @return JTextArea
     */
    JTextArea messageBox() {
        UUID uuid = this.model.MESSAGE_BOX;

        JTextArea jTextArea = new JTextArea();
        jTextArea.setPreferredSize(new Dimension(100,50));


        DocumentListener dl = new DocumentListener() {
            @Override
            public void insertUpdate(DocumentEvent e) {
                controller.handle(uuid,
                        new ActionEvent(jTextArea, 0, jTextArea.getText())
                );
            }
            @Override
            public void removeUpdate(DocumentEvent e) {
                controller.handle(uuid,
                        new ActionEvent(jTextArea, 0, jTextArea.getText())
                );
            }
            @Override
            public void changedUpdate(DocumentEvent e) {}
        };
        jTextArea.getDocument().addDocumentListener(dl);


        Updater updater = () -> {
            ChatModel.messageBox message =
                    (ChatModel.messageBox) this.model.getData(this.model.MESSAGE_BOX);
            String strMessage = message.message();

            //if the text is different then update it, else do nothing
            if (!jTextArea.getText().equals(strMessage)) {
                //set text without triggering event
                jTextArea.getDocument().removeDocumentListener(dl);
                jTextArea.setText(strMessage);
                //restore the change listener
                jTextArea.getDocument().addDocumentListener(dl);
            }

        };
        this.updateMethods.put(uuid, updater);

```

```java
        return jTextArea;
    }

    /**
     * @author Abir Faisal
     * @return JButton
     */
    JButton sendMessageButton(){
        UUID uuid = this.model.SEND_MESSAGE_BUTTON;
        JButton jButton = new JButton("Send");

        jButton.addActionListener(actionEvent -> {
            this.controller.handle(uuid, actionEvent);
        });

        return jButton;
    }


}

```

```java
package edu.fau.eng.cop4331.ttt3d.app.chat;

import edu.fau.eng.cop4331.ttt3d.app.Model;

import java.util.ArrayList;
import java.util.Stack;
import java.util.UUID;

public class ChatModel extends Model {


    /**
     * The View uses these constants to get data from the Model
     * The Controller uses these constants to update data in the Model
     *
     * Every element in a view that needs to be updated
     * needs to have a UUID refrence to it here.
     *
     * These are non-static so the UUID will
     * be unique to each instance of the class
     *
     */
    public UUID MAIN = UUID.randomUUID();
    public UUID HELLO_WORLD_JLABEL = UUID.randomUUID();
    public UUID CHAT_LOG = UUID.randomUUID();
    public UUID MESSAGE_BOX = UUID.randomUUID();
    public UUID SEND_MESSAGE_BUTTON = UUID.randomUUID();

    //data structures

    /**
     * Holds an stack array of String messages
     * to be displayed by the view or updated by the controller
     *
     * @param messages Stack<String>
     */
    public record chatLog(Stack<String> messages){}

    /**
     * Holds the text that the user types into the message box
     *
     * @param message String
     */
    public record messageBox(String message){}


}

```

```java
package edu.fau.eng.cop4331.ttt3d.app.chat;

import edu.fau.eng.cop4331.ttt3d.app.Controller;
import edu.fau.eng.cop4331.ttt3d.app.Handler;

import java.util.ArrayList;
import java.util.Stack;
import java.util.UUID;

public abstract class ChatController extends Controller {

    ChatModel model;
    ChatView view;
    ArrayList<String> sentMessageBuffer;

    /**
     * Constructor
     * @param chatModel ChatModel
     * @param chatView ChatView
     */
    public ChatController(ChatModel chatModel, ChatView chatView) {
        this.model = chatModel;
        this.view = chatView;
        this.view.registerController(this);
        this.sentMessageBuffer = new ArrayList<>();

        runHandlers();
        setup();
    }

    /**
     * Setup the controller
     */
    void setup() {
        handlers.put(this.model.SEND_MESSAGE_BUTTON, sendChatButtonHandler());
        handlers.put(this.model.MESSAGE_BOX, messageBoxEventHandler());

        //init the chat log datastrcture
        Stack<String> s = new Stack<String>();
        s.push("");
        this.model.setData(this.model.CHAT_LOG, new ChatModel.chatLog(s));

        this.model.setData(this.model.MESSAGE_BOX, new ChatModel.messageBox(""));
    }

    //event handlers//////////

    /**
     * Handles what happens when the send chat button is pressed
     *
     * @author Abir Faisal
     * @return
     */
    Handler sendChatButtonHandler() {
        UUID messageBoxUUID = this.model.MESSAGE_BOX;

        return value -> {
            System.out.println("send button pressed");
            //get the text from the message
            ChatModel.messageBox mb =
                    (ChatModel.messageBox) this.model.getData(messageBoxUUID);
            String message = mb.message();

            //clear the message in the model
            this.model.setData(messageBoxUUID, new ChatModel.messageBox(""));

            //append the message to the chat
            appendChatLog("Player 1: " + message);

            //put the message in the message buffer for the chat bot
            this.sentMessageBuffer.add(message);
        };
    }

    /**
     * Updates the data in the model
     * when the text in the message box changes
     *
     * @author Abir Faisal
     * @return
     */
    Handler messageBoxEventHandler() {
        UUID uuid = this.model.MESSAGE_BOX;
        return actionEvent -> {
            //update the model
            this.model.setData(uuid,
                    new ChatModel.messageBox(actionEvent.getActionCommand())
            );
        };
    }

    //controller logic///////////

    /**
     * Monitors the message buffer for any messages from the user
     * if so then it responds to it
     *
     * This can be a chat bot or client it should be implimented such that it
     * reads the message buffer, handles it, then clear the message from the buffer
     *
     * Preferable it should be in it's own thread.
     *
     * @author Abir Faisal
     */
    public abstract void sentMessageBufferHandler();

    /**
     * Append a message to the chatLog data structure in the model
     * This should be called when your messageBufferHandler produces response
     *
     * @author Abir Faisal
     * @param message String message you want to append
     */
    void appendChatLog(String message) {
        UUID chatLogUUID = this.model.CHAT_LOG;

        //append the message to the chat
        ChatModel.chatLog cl =
                (ChatModel.chatLog) this.model.getData(chatLogUUID);
        Stack<String> messages = cl.messages();

        //put the new message on the top of the stack
        messages.push(message);

        //update the chatlog datastructure in the model
        this.model.setData(chatLogUUID, new ChatModel.chatLog(messages));
    }

}

```

```java
package edu.fau.eng.cop4331.ttt3d.app.chat;

import edu.fau.eng.cop4331.ttt3d.app.Handler;

import java.util.*;

import static java.lang.Thread.sleep;

public class ChatBotController extends ChatController {

    /**
     * Constructor
     * @param chatModel ChatModel
     * @param chatView ChatView
     */
    public ChatBotController(ChatModel chatModel, ChatView chatView) {
        super(chatModel, chatView);
        sentMessageBufferHandler();
    }

    //controller logic//////

    /**
     * Monitors the message buffer for any messages from the user
     * if so then it responds to it
     *
     * @author Abir Faisal
     */
    @Override
    public void sentMessageBufferHandler() {
        new Thread(() -> {
            while (true) {
                for (int i = 0; i < this.sentMessageBuffer.size(); i++) {
                    //allow the bot to respond
                    getBotResponse(this.sentMessageBuffer.get(i));
                    //remove from buffer
                    this.sentMessageBuffer.remove(i);
                }

                try {
                    sleep(100); //prevent using CPU cycles for no reason.
                }catch (InterruptedException e) {
                }
            }
        }).start();
    }


    /**
     * gets a computer generated response and puts it into the chat
     *
     * @author Abir Faisal
     */
    void getBotResponse(String message) {
        //TODO make more advanced
        String[] responses = {"Ok", "I understand", "Sure"};
        Random r = new Random();
        int i = r.nextInt(responses.length);

        appendChatLog("Bot: " + responses[i]);
    }

}

```

```java
package edu.fau.eng.cop4331.ttt3d.app.chat;

import java.util.ArrayList;

import static java.lang.Thread.sleep;

public class ChatClientController extends ChatController {

    ArrayList<String> receivedMessageBuffer;

    /**
     * Constructor
     * @param chatModel ChatModel
     * @param chatView ChatView
     */
    public ChatClientController(ChatModel chatModel, ChatView chatView) {
        super(chatModel, chatView);
        sentMessageBufferHandler();
    }

    /**
     * Handles the messages in the message buffer
     * Sends the message to the server
     */
    @Override
    public void sentMessageBufferHandler() {
        new Thread(() -> {
            while (true) {
                for (int i = 0; i < this.sentMessageBuffer.size(); i++) {
                    //send the message
                    System.out.println("Sending Message: " + sentMessageBuffer.get(i));
                    sendMessage(this.sentMessageBuffer.get(i));

                    //remove from buffer
                    this.sentMessageBuffer.remove(i);
                }

                try {
                    sleep(100); //prevent using CPU cycles for no reason.
                }catch (InterruptedException e) {
                }
            }
        }).start();
    }

    /**
     * Handles sending the message to the server
     * @param message String
     */
    void sendMessage(String message) {

    }

    /**
     * Handles recieved messages in the recieved message buffer
     */
    void receivedMessageBufferHandler() {
        new Thread(() -> {
            while (true) {
                for (int i = 0; i < this.receivedMessageBuffer.size(); i++) {
                    System.out.println("Recieved Message: " + receivedMessageBuffer.get(i));
                    //put the recieved message into the view

                    //remove from buffer
                    this.receivedMessageBuffer.remove(i);
                }
                try {
                    sleep(100); //prevent using CPU cycles for no reason.
                }catch (InterruptedException e) {
                }
            }
        }).start();
    }

}

```

```java
package edu.fau.eng.cop4331.ttt3d.app.game;

/**
 * The type of game to launch
 */
public enum GameType {
    SINGLE_PLAYER_GAME,
    MULTI_PLAYER_CLIENT_GAME,
    MULTI_PLAYER_HOST_GAME
}

```

```java
package edu.fau.eng.cop4331.ttt3d.app.game;

import edu.fau.eng.cop4331.ttt3d.app.Updater;
import edu.fau.eng.cop4331.ttt3d.app.View;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.time.Instant;
import java.util.UUID;

public class GameView extends View {

    GameModel model;

    /**
     * Constructor
     *
     * @param gameModel GameModel
     */
    public GameView(GameModel gameModel) {
        this.model = gameModel; //make view aware of model
        this.model.register(this); //make model aware of view
        setup(); //setup the view
    }

    /**
     * Setup the view
     */
    @Override
    public void setup() {
        JPanel mainJPanel = new JPanel();
        this.jFrames.put(model.MAIN, mainJPanel);
        mainJPanel.setLayout(new BoxLayout(mainJPanel, BoxLayout.Y_AXIS));

        this.jFrames.get(model.MAIN).add(winLossTieCounter());

        this.jFrames.get(model.MAIN).add(new JLabel("Layer1"));
        this.jFrames.get(model.MAIN).add(xyButtonGrid(0));

        this.jFrames.get(model.MAIN).add(new JLabel("Layer2"));
        this.jFrames.get(model.MAIN).add(xyButtonGrid(1));

        this.jFrames.get(model.MAIN).add(new JLabel("Layer3"));
        this.jFrames.get(model.MAIN).add(xyButtonGrid(2));

    }

    /////UI elements/////////

    JLabel winLossTieCounter() {
        JLabel jLabel = new JLabel("Win: 0 Loss: 0 Tie: 0");
        UUID uuid = this.model.STAT_COUNTER;

        Updater updater = new Updater() {
            @Override
            public void update() {
                GameModel.stats stats = (GameModel.stats) model.getData(uuid);
                int win = stats.wins();
                int loss = stats.losses();
                int tie = stats.ties();
                String statStr = "Win:" + win + " Loss:" + loss + " Tie:"+ tie;
                jLabel.setText(statStr);
            }
        };
        updateMethods.put(model.STAT_COUNTER, updater);

        return jLabel;
    }


    /**
     * Grid that contains 3x3 button array
     * The l value is used to deterimine
     * which layer of the cube this grid corresponds to
     *
     * @param layer the layer also known as the z axis
     * @return the grid
     */
    JPanel xyButtonGrid(int layer) {
        JPanel grid = new JPanel();
        grid.setLayout(new GridLayout(3,3));
        UUID gameGridUUID = this.model.GAME_GRID;
        UUID[][][] buttonUUIDS = this.model.GAME_GRID_BUTTONS;
        int index = 0;

        //generate the buttons
        for (int y = 0; y < 3; y++) {
            for (int x = 0; x < 3; x++) {
                grid.add(gameButton(x, y, layer, index));
                index +=1;
            }
        }

        //refreshes the buttons
        Updater updater = new Updater() {
            @Override
            public void update() {
                for (int z = 0; z < 3; z++) {
                    for (int y = 0; y < 3; y++) {
                        for (int x = 0; x < 3; x++) {
                            UUID uuid = buttonUUIDS[x][y][z];
                            GameModel.gameState3D gs3d = (GameModel.gameState3D) model.getDa
                            int [][][] gs = gs3d.gameState3D();
                            gs[x][y][z] = 0;
                            model.setData(uuid, new GameModel.gameState3D(gs));
                        }
                    }
                }
            }
        };
        updateMethods.put(gameGridUUID, updater);

        return grid;
    }

    /**
     * generates the game button given the x,y,z cordinates and index
     *
     * @param x cordinate
     * @param y cordinate
     * @param z layer
     * @param index counter
     * @return
     */
    JButton gameButton(int x, int y, int z, int index) {
        UUID gameGridUUID = this.model.GAME_GRID;
        UUID buttonUUID = this.model.GAME_GRID_BUTTONS[x][y][z];

        JButton jButton = new JButton("-");
        jButton.setPreferredSize(new Dimension(50,50));
        jButton.setFont(new Font(null, Font.PLAIN, 40));

        //event handler will recieve this string "x,y"
        //optionally it can use index to identify which button was pressed
        String coordinates = x + "," + y + "," + z;

        //action event to be passed to the controller
        ActionEvent ae = new ActionEvent(jButton, index, coordinates);
        jButton.addActionListener(e -> this.controller.handle(gameGridUUID, ae));

        //if model is updated with a new gameState then do this
        int xf = x; //final
        int yf = y; //final
        int zf = z; //final
        Updater updater = new Updater() {
            @Override
            public void update() {
//                  System.out.println("xyz" + xf + yf + zf);
                //read the state from the game state record int the model datastructures
                GameModel.gameState3D gs3d = (GameModel.gameState3D) model.getData(gameGridU
                int[][][] gs = gs3d.gameState3D();
                int state = gs[xf][yf][zf];

                //if 1 then "X" if -1 then "O" else "-"
                if (state == 1) jButton.setText("X");
                else if (state == -1) jButton.setText("O");
                else jButton.setText("-");
            }
        };
        updateMethods.put(buttonUUID, updater);

        return jButton;
    }


    /**
     *
     * @return GameModel
     */
    public GameModel getGameModel() {
        return model;
    }
}
```

```java
package edu.fau.eng.cop4331.ttt3d.app.game;

import edu.fau.eng.cop4331.ttt3d.app.Model;

import java.util.UUID;

public class GameModel extends Model {

    /**
     * The View uses these constants to get data from the Model
     * The Controller uses these constants to update data in the Model
     *
     * Every element in a view that needs to be updated
     * needs to have a UUID refrence to it here.
     *
     * These are non-static so the UUID will
     * be unique to each instance of the class
     *
     */
    public UUID MAIN = UUID.randomUUID();
    public UUID HELLO_WORLD_JLABEL = UUID.randomUUID();
    public UUID GAME_GRID = UUID.randomUUID();
    public UUID[][][] GAME_GRID_BUTTONS;
    public UUID STAT_COUNTER;

    /**
     * initializes the UUIDs for GAME_GRID_BUTTONS
     */
    public GameModel() {
        this.GAME_GRID_BUTTONS = new UUID[3][3][3];
        for (int z = 0; z < this.GAME_GRID_BUTTONS.length; z++) {
            for (int y = 0; y < this.GAME_GRID_BUTTONS.length; y++) {
                for (int x = 0; x < this.GAME_GRID_BUTTONS.length; x++) {
                    this.GAME_GRID_BUTTONS[x][y][z] = UUID.randomUUID();
                }
            }
        }
    }


    /**
     * Holds the state of the game
     * 1 = X
     * 0 = empty
     * -1 = O
     *
     * @param gameState3D int[][][]
     */
    public record gameState3D(int[][][] gameState3D){}

    /**
     * Holds the number of wins, losses, and ties
     * to be displayed in the view
     *
     * @param wins int
     * @param losses int
     * @param ties int
     */
    public record stats(
            int wins,
            int losses,
            int ties
    ){}


}
```

```java
package edu.fau.eng.cop4331.ttt3d.app.game;

import edu.fau.eng.cop4331.ttt3d.app.Controller;

public class MultiPlayerHostController extends Controller {

    GameModel model;
    GameView view;

    /**
     * Constructor
     *
     * @param gameModel GameModel
     * @param gameView GameView
     */
    public MultiPlayerHostController(GameModel gameModel, GameView gameView) {
        this.model = gameModel;
        this.model.register(gameView);
        this.view = gameView;
        this.view.registerController(this);

        setup();
    }

    /**
     * Setup the controller
     */
    void setup(){
    };


    //used when hosting a game for another player and yourself

}

```

```java
package edu.fau.eng.cop4331.ttt3d.app.game;

import edu.fau.eng.cop4331.ttt3d.app.Controller;
import edu.fau.eng.cop4331.ttt3d.app.Handler;
import edu.fau.eng.cop4331.ttt3d.util.Solver;

import javax.swing.*;
import java.awt.event.ActionEvent;
import java.util.Random;
import java.util.UUID;

public class SinglePlayerGameController extends Controller {
    //controller that connects the view with a single player game model

    GameModel model;
    GameView view;

    public SinglePlayerGameController(GameModel model, GameView view) {
        this.model = model;
        this.view = view;
        this.view.registerController(this);

        runHandlers();
        setup();
    }

    void setup() {
        newGame();
        resetStats();

        this.handlers.put(model.GAME_GRID, gridButtonPressedHandler());
    }

    //Event Handlers///////////////

    /**
     * This handler recieves x,y cordinates of the button that was pressed
     * @return A Handler that reacts to button presses on it's grid.
     */
    Handler gridButtonPressedHandler() {
        return new Handler() {
            @Override
            public void handle(ActionEvent value) {
//                System.out.println(value.getID());
                //get int x,y and z from String "x,y,z"
                String[] s = value.getActionCommand().split(",");
                int x = Integer.parseInt(s[0]);
                int y = Integer.parseInt(s[1]);
                int z = Integer.parseInt(s[2]);
                System.out.println(x + "," + y + "," + z);
                makeMove(x, y, z, 1);
            }
        };
    }

    //Game logic////////////
    Solver solver = new Solver();

    /**
     *
     * Validates and makes a move and updates the model
     * Also tells user if the game was won and if so resets the game
     *
     * @author Abir Faisal
     * @param x
     * @param y
     * @param z
     * @param player
     */
    void makeMove(int x, int y, int z, int player) {
        System.out.format("interpreting move xyz=%d,%d,%d player=%d", x, y, z, player);
        GameModel.gameState3D gs3d = (GameModel.gameState3D) this.model.getData(this.model.G

        //make sure the postion was empty
        boolean isValidMove = isValidMove(x, y, z, gs3d.gameState3D());

        UUID buttonUUID = this.model.GAME_GRID_BUTTONS[x][y][z];

        //if valid then update model
        if (isValidMove) {
            System.out.println(" validMove");
            //update the model
            int[][][] gs = gs3d.gameState3D();
            gs[x][y][z] = (player == 1) ? 1 : -1; //X=1 O=-1
            this.model.setData(buttonUUID, new GameModel.gameState3D(gs));

            //check if there is a winner
            gs3d = (GameModel.gameState3D) this.model.getData(this.model.GAME_GRID);
            int winner = solver.solve(gs3d.gameState3D());

            //if no winner, make next move
            if (winner == 3) { //X
                System.out.println("X wins");
                updateStats(1);
                JOptionPane.showMessageDialog(null, "You won");
                newGame();
            } else if (winner == -3) { //O
                System.out.println("O wins");
                updateStats(-1);
                JOptionPane.showMessageDialog(null, "You lost");
                newGame();
            }
            else if (tiedGame()) {
                System.out.println("Tied Game");
                updateStats(0);
                JOptionPane.showMessageDialog(null, "The game was tied");
                newGame();
            }
            else if (player == 1) makeNextMove(gs);

        } else System.out.println(" invalidMove");
    }


    /**
     * Updates the game stats in the model with the new values
     *
     * @author Abir Faisal
     * @param winLossTie 1=win -1=loss 0=tie
     */
    void updateStats(int winLossTie){
        //get data from model
        GameModel.stats stats = (GameModel.stats) this.model.getData(this.model.STAT_COUNTER
        GameModel.stats  newStats = null;

        switch (winLossTie){
            case 1: {
                //update the stats
                newStats = new GameModel.stats(stats.wins() + 1, stats.losses(), stats.ties(
                break;
            }
            case -1: {
                //update the stats
                newStats = new GameModel.stats(stats.wins(), stats.losses() + 1, stats.ties(
                break;
            }
            case 0: {
                //update the stats
                newStats = new GameModel.stats(stats.wins(), stats.losses(), stats.ties() +
                break;
            }
        }
        //update the model with the new stats
        this.model.setData(this.model.STAT_COUNTER, newStats);
    }


    /**
     * Check if the game is tied
     * @return true = tied, false = not tied
     */
    boolean tiedGame(){
        return false; //TODO
    }

    /**
     * Setup a new game
     * @author Abir Faisal
     * setup a new game
     */
    void newGame() {
        //empty game grid
        //should init to zeros automatically
        int[][][] newGameState = new int[3][3][3];
        this.model.setData(model.GAME_GRID,
                new GameModel.gameState3D(newGameState)
        );
    }

    void resetStats(){
        this.model.setData(model.STAT_COUNTER, new GameModel.stats(0,0,0));
    }

    /**
     * check if the move is a valid move
     *
     * @author Abir Faisal
     * @param x coordinate
     * @param y coordinate
     * @param z coordinate
     * @param gameState
     * @return true if the move is valid, false if it is invalid.
     */
    boolean isValidMove(int x, int y, int z, int[][][] gameState) {
        if (gameState[x][y][z] == 0) return true;
        else return false;
    }

    /**
     * Single player mode
     * Computer makes next move
     *
     * @author Abir Faisal
     */
    void makeNextMove(int[][][] gameState) {
        //select random position
        Random r = new Random();
        int x = r.nextInt(gameState.length);
        int y = r.nextInt(gameState[x].length);
        int z = r.nextInt(gameState[x][y].length);

        System.out.println("\ncomputer move " + x + "," + y + "," + z);

        //validate decision
        while (gameState[x][y][z] != 0) {
            System.out.println("NOT VALID RECALCULATING");
            x = r.nextInt(gameState.length);
            y = r.nextInt(gameState[x].length);
            z = r.nextInt(gameState[x][y].length);
        }
        makeMove(x, y, z, 0); //player 0 is always opponent

    }
}
```

```java
package edu.fau.eng.cop4331.ttt3d.app.game;

import edu.fau.eng.cop4331.ttt3d.app.Controller;

public class MultiPlayerClientController extends Controller {

    GameModel model;
    GameView view;

    public MultiPlayerClientController(GameModel gameModel, GameView gameView) {
        this.model = gameModel;
        this.model.register(gameView);
        this.view = gameView;
        this.view.registerController(this);

        setup();
    }


    void setup(){

    }


    //Use cases
    //When the user wants to connect to a multipleyer server
    //When the user wants to connect to single host

}

```

```java
package edu.fau.eng.cop4331.ttt3d.app.startscreen;

import edu.fau.eng.cop4331.ttt3d.app.App;
import edu.fau.eng.cop4331.ttt3d.app.Updater;
import edu.fau.eng.cop4331.ttt3d.app.View;
import edu.fau.eng.cop4331.ttt3d.app.game.GameType;

import javax.swing.*;
import javax.swing.event.DocumentEvent;
import javax.swing.event.DocumentListener;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.time.Instant;
import java.util.UUID;

public class StartScreenView extends View {

    //The model that this view will reference
    //when it needs to update
    StartScreenModel model;

    /**
     * Instantiate and setup the View
     *
     * @author Abir Faisal
     * @param startScreenModel StartScreenModel
     */
    public StartScreenView(StartScreenModel startScreenModel) {
        this.model = startScreenModel; //make view aware of model
        this.model.register(this); //make model aware of view
        setup(); //setup the view
    }

    //set up the view
    @Override
    public void setup() {
        JPanel mainJPanel = new JPanel();
        this.jFrames.put(model.MAIN, mainJPanel);

        //centering panel for asthetic purposes.
        JPanel centeringPanel = new JPanel();
        centeringPanel.setLayout(new BoxLayout(centeringPanel, BoxLayout.Y_AXIS));

        centeringPanel.add(new JLabel("Server IP"));
        centeringPanel.add(serverIPJTextField());

        centeringPanel.add(new JLabel("Server Port"));
        centeringPanel.add(serverPortJTextField());

        centeringPanel.add(startSinglePlayerGameButton());
        centeringPanel.add(startMultiPlayerGameButton());
        centeringPanel.add(startHostGameButton());


        //put centering panel in mainJpanel
        this.jFrames.get(model.MAIN).add(centeringPanel);
    }

    //NOTE: Try to keep these methods in order as they appear visually

    /**
     * Text field where the user enters the server IP
     *
     * @author Abir Faisal
     * @return a JTextField for the user to type in the server IP and port
     */
    JTextField serverIPJTextField() {
        JTextField serverIPTextField = new JTextField("0.0.0.0");
        serverIPTextField.setMaximumSize(new Dimension(300, 25));
        UUID uuid = this.model.SERVER_IP_TEXT_FIELD;

        //when the text field is changed
        //notify the controller of the change
        DocumentListener dl1 = new DocumentListener() {
            @Override
            public void insertUpdate(DocumentEvent e) {
                controller.handle(uuid,
                        new ActionEvent(serverIPTextField, 0, serverIPTextField.getText())
                );
            }
            @Override
            public void removeUpdate(DocumentEvent e) {
                controller.handle(uuid,
                        new ActionEvent(serverIPTextField, 0, serverIPTextField.getText())
                );
            }
            @Override
            public void changedUpdate(DocumentEvent e) {}
        };
        serverIPTextField.getDocument().addDocumentListener(dl1);

        //updates the UI if there is a change in the Model
        Updater updater = () -> {

            //get the data from the model as ServerInfo
            StartScreenModel.ServerIP ip = (StartScreenModel.ServerIP) this.model.getData(uu

            //if the text is different then update it, else do nothing
            if (!serverIPTextField.getText().equals(ip.ipAddress())) {
                //set text without triggering listner
                serverIPTextField.getDocument().removeDocumentListener(dl1);
                serverIPTextField.setText(ip.ipAddress());
                //restore the change listener
                serverIPTextField.getDocument().addDocumentListener(dl1);
            }
        };
        this.updateMethods.put(uuid, updater);

        return serverIPTextField;
    }

    /**
     *
     * Text field where the user enters the server port number
     *
     * @author Abir Faisal
     * @return
     */
    JTextField serverPortJTextField() {
        JTextField jTextField = new JTextField("1234");
        jTextField.setMaximumSize(new Dimension(300, 25));
        UUID uuid = this.model.SERVER_PORT_TEXT_FIELD;

        //when the text field is changed
        //notify the controller of the change
        DocumentListener dl = new DocumentListener() {
            @Override
            public void insertUpdate(DocumentEvent e) {
                controller.handle(uuid,
                        new ActionEvent(jTextField, 0, jTextField.getText())
                );
            }
            @Override
            public void removeUpdate(DocumentEvent e) {
                controller.handle(uuid,
                        new ActionEvent(jTextField, 0, jTextField.getText())
                );
            }
            @Override
            public void changedUpdate(DocumentEvent e) {}
        };
        jTextField.getDocument().addDocumentListener(dl);


        //updates the UI if there is a change in the Model
        Updater updater = () -> {

            //get the data from the model as ServerInfo
            StartScreenModel.ServerPort port =
                    (StartScreenModel.ServerPort) this.model.getData(uuid);

            //if the text is different then update it, else do nothing
            if (!jTextField.getText().equals(port.port())) {
                //set text without triggering event
                jTextField.getDocument().removeDocumentListener(dl);
                jTextField.setText(port.port());
                //restore the change listener
                jTextField.getDocument().addDocumentListener(dl);
            }


        };
        this.updateMethods.put(uuid, updater);

        return jTextField;
    }



    //TODO convert to a loop

    /**
     *  Button that starts a single player game
     *
     * @author Abir Faisal
     * @return
     */
    JButton startSinglePlayerGameButton() {
        //instantiate the button
        JButton jButton = new JButton("Single Player");
        UUID uuid = this.model.START_SINGLE_PLAYER_GAME_BUTTON;

        jButton.addActionListener(actionEvent -> {
            this.controller.handle(uuid, actionEvent);
        });

        return jButton;
    }

    /**
     * Button that starts a multi player game
     *
     * @author Abir Faisal
     * @return
     */
    JButton startMultiPlayerGameButton() {
        //instantiate the button
        JButton jButton = new JButton("Multi Player");
        UUID uuid = this.model.START_MULTI_PLAYER_GAME_BUTTON;

        jButton.addActionListener(actionEvent -> {
            this.controller.handle(uuid, actionEvent);
        });
        return jButton;
    }

    /**
     * Button that starts a hosting a game for one other player
     *
     * @author Abir Faisal
     * @return
     */
    JButton startHostGameButton() {
        //instantiate the button
        JButton jButton = new JButton("Host Game");
        UUID uuid = this.model.START_MULTI_HOST_GAME_BUTTON;

        jButton.addActionListener(actionEvent -> {
            this.controller.handle(uuid, actionEvent);
        });

        return jButton;
    }
}
```

```java
package edu.fau.eng.cop4331.ttt3d.app.startscreen;

import edu.fau.eng.cop4331.ttt3d.app.Model;

import java.util.UUID;


public class StartScreenModel extends Model {

    /**
     * The View uses these constants to get data from the Model
     * The Controller uses these constants to update data in the Model
     *
     * Every element in a view that needs to be updated
     * needs to have a UUID refrence to it here.
     *
     * These are non-static so the UUID will
     * be unique to each instance of the class
     *
     */
    public UUID MAIN = UUID.randomUUID();
    public UUID HELLO_WORLD_JLABEL = UUID.randomUUID();
    public UUID TEST_BUTTON = UUID.randomUUID();
    public UUID SERVER_IP_TEXT_FIELD = UUID.randomUUID();
    public UUID SERVER_PORT_TEXT_FIELD = UUID.randomUUID();
    public UUID START_SINGLE_PLAYER_GAME_BUTTON = UUID.randomUUID();
    public UUID START_MULTI_PLAYER_GAME_BUTTON = UUID.randomUUID();
    public UUID START_MULTI_HOST_GAME_BUTTON = UUID.randomUUID();


    //example data strcuture holding some information to be
    //used by the view or updated by the controller
    public record ExampleDataStruct(
            String s,
            double n,
            int i,
            int[] arrayList
    ){}

    public record ServerIP(String ipAddress){}
    public record ServerPort(String port){}

}
```

```java
package edu.fau.eng.cop4331.ttt3d.app.startscreen;

import edu.fau.eng.cop4331.ttt3d.app.App;
import edu.fau.eng.cop4331.ttt3d.app.Controller;
import edu.fau.eng.cop4331.ttt3d.app.Handler;
import edu.fau.eng.cop4331.ttt3d.app.game.GameType;
import edu.fau.eng.cop4331.ttt3d.util.SettingsManager;

import java.awt.event.ActionEvent;
import java.util.UUID;

public class StartScreenController extends Controller {

    StartScreenModel model;
    StartScreenView view;

    /**
     * Constructor
     * @param scm StartScreenModel
     * @param scv StartScreenView
     */
    public StartScreenController(StartScreenModel scm, StartScreenView scv) {
        this.model = scm;
        this.view = scv;
        this.view.registerController(this);

        runHandlers();
        System.out.println("running event handlers");
        setup();
    }

    /**
     * Setup the view
     */
    void setup() {
        handlers.put(model.START_SINGLE_PLAYER_GAME_BUTTON, startSinglePlayerGameHandler());
        handlers.put(model.START_MULTI_PLAYER_GAME_BUTTON, startMultiPlayerGameHandler());
        handlers.put(model.START_MULTI_HOST_GAME_BUTTON, startHostGameHandler());

        handlers.put(model.SERVER_IP_TEXT_FIELD, serverIPInfoUpdateHandler());
        handlers.put(model.SERVER_PORT_TEXT_FIELD, serverPortUpdateHandler());

        //load settings
        SettingsManager sm = SettingsManager.getInstance();
        String ipAddress = sm.getSettings().getString("userDefinedServer");
        String port = sm.getSettings().getString("userDefinedPort");

        //data type used by model
        StartScreenModel.ServerIP serverIP = new StartScreenModel.ServerIP(ipAddress);
        StartScreenModel.ServerPort serverPort = new StartScreenModel.ServerPort(port);

        //set the data in the model
        this.model.setData(model.SERVER_IP_TEXT_FIELD, serverIP);
        this.model.setData(model.SERVER_PORT_TEXT_FIELD, serverPort);

    }


    //Action handlers
    /**
     * Save server settings and tell Application
     * to launch GameType.SINGLE_PLAYER_GAME
     *
     * @author Abir Faisal
     * @return a Handler that launches a single player game
     */
     Handler startSinglePlayerGameHandler() {
         StartScreenController instance = StartScreenController.this;

         return new Handler() {
             @Override
             public void handle(ActionEvent value) {
                 System.out.println("Start Single Player Button Pressed");
                 //save the settings
                 saveUserSettings();
                 //launch the game
                 App.getInstance().launchGame(GameType.SINGLE_PLAYER_GAME);
             }
         };
     }

     /**
      * Save server settings and tell Application
      * to launch GameType.MULTI_PLAYER_CLIENT_GAME
      *
      * @author Abir Faisal
      * @return a Handler that launches a single player game
      */
     Handler startMultiPlayerGameHandler() {
         StartScreenController instance = StartScreenController.this;

         return new Handler() {
             @Override
             public void handle(ActionEvent value) {
                 System.out.println("Start Multi Player Button Pressed");
                 //save the settings
                 saveUserSettings();
                 //launch the game
                 App.getInstance().launchGame(GameType.MULTI_PLAYER_CLIENT_GAME);
             }
         };
     }

     /**
      * Save server settings and tell Application
      * to launch GameType.MULTI_PLAYER_HOST_GAME
      *
      * @author Abir Faisal
      * @return a Handler that launches a single player game
      */
     Handler startHostGameHandler() {
         StartScreenController instance = StartScreenController.this;

         return new Handler() {
             @Override
             public void handle(ActionEvent value) {
                 System.out.println("Start Host Game Button Pressed");
                 //save the settings
                 saveUserSettings();
                 //launch the game
                 App.getInstance().launchGame(GameType.MULTI_PLAYER_HOST_GAME);
             }
         };
     }



     /**
      * When the user changes the server IP
      *
      * @author Abir Faisal
      * @return A handler that updates the model with a new value
      */
     Handler serverIPInfoUpdateHandler(){
         UUID uuid = model.SERVER_IP_TEXT_FIELD;
         StartScreenController instance = StartScreenController.this;

         return new Handler() {
             @Override
             public void handle(ActionEvent value) {
                 String serverIP = value.getActionCommand(); //get the IP:Port
                 //Update the model with the IP
                 StartScreenController.this.model.setData(uuid, new StartScreenModel.ServerIP
                 System.out.println(instance.model.getData(uuid));
             }
         };
     }

     /**
      * When the user changes the server Port
      *
      * @author Abir Faisal
      * @return A handler that updates the model with a new value
      */
     Handler serverPortUpdateHandler(){
         UUID uuid = model.SERVER_PORT_TEXT_FIELD;
         StartScreenController instance = StartScreenController.this;

         return new Handler() {
             @Override
             public void handle(ActionEvent value) {
                 String serverPort = value.getActionCommand();
                 //update the model with the port
                 model.setData(uuid, new StartScreenModel.ServerPort(serverPort));
                 System.out.println(instance.model.getData(uuid));
             }
         };
     }


     //controller logic//////////////////////////

     /**
      * Save the user settings from the ServerIP and port text input fields
      *
      * @author Abir Faisal
      */
     void saveUserSettings() {
         StartScreenController instance = StartScreenController.this;


         //save the settings
         StartScreenModel.ServerIP serverIPRecord =
                 (StartScreenModel.ServerIP) instance.model.getData(model.SERVER_IP_TEXT_FIEL

         StartScreenModel.ServerPort serverPortRecord =
                 (StartScreenModel.ServerPort) instance.model.getData(model.SERVER_PORT_TEXT_

         SettingsManager.getInstance().setValue("userDefinedServer", serverIPRecord.ipAddress
         SettingsManager.getInstance().setValue("userDefinedPort", serverPortRecord.port());
     }

 }

```

```java
package edu.fau.eng.cop4331.ttt3d.util;

public class Solver {
    //class that contains game solvers

    public Solver() {
    }

    /**
     * Solves the game given a 1D representation of the gameState
     * @param gameState1D
     */
    public void solve(int[] gameState1D) {
        //TODO check winner given a 1D game state array
    }

    /**
     * Solves the game given a 2D representation of the gameState
     * @param gameState2D
     */
    public void solve(int[][] gameState2D) {
        //TODO check winner given a 2D game state array
    }

    /**
     * Solves the game given a 3D representation of the gameState
     *
     * @param gameState3D
     */
    public int solve(int[][][] gameState3D) {
        int y0;
        int y1;
        int y2;

        int x0;
        int x1;
        int x2;

        //solve for horizontal and vertical wins
        for (int z = 0; z < 3; z++) { //layer
            for (int i = 0; i < 3; i++) {
                y0 = gameState3D[i][0][z];
                y1 = gameState3D[i][1][z];
                y2 = gameState3D[i][2][z];
                int hSum = y0 + y1 + y2;

                x0 = gameState3D[0][i][z];
                x1 = gameState3D[1][i][z];
                x2 = gameState3D[2][i][z];
                int vSum = x0 + x1 + x2;

                if (hSum == 3) return hSum;
                if (hSum == -3) return hSum;
                if (vSum == 3) return vSum;
                if (vSum == -3) return vSum;
            }
        }

        //check for diagonal wins
        for (int z = 0; z < 3; z++) {
            int topLeft = gameState3D[0][0][z]; int topRight = gameState3D[2][0][z];
                              int center = gameState3D[1][1][z];
            int bottomLeft = gameState3D[0][2][z];int bottomRight = gameState3D[2][2][z];

            int diag1 = topLeft + center + bottomRight;
            int diag2 = bottomLeft + center + topRight;

            if (diag1 == 3) return diag1;
            if (diag1 == -3) return diag1;
            if (diag2 == 3) return diag2;
            if (diag2 == -3) return diag2;
        }

        //check for orthogonal wins TODO

        //no winners found
        return 0;
    }

}

```

```java
package edu.fau.eng.cop4331.ttt3d.util;

import org.json.JSONObject;

import java.io.File;
import java.io.IOException;
import java.io.PrintWriter;
import java.nio.file.Files;
import java.nio.file.Path;

import static java.lang.System.exit;

public class SettingsManager {

    private JSONObject settings;
    private final String settingsFileName = "settings.json";

    //singleton pattern
    private static SettingsManager instance;
    private SettingsManager() {
    }
    public static synchronized SettingsManager getInstance(){
        if (instance == null) instance = new SettingsManager();
        return instance;
    }

    public void loadSettings() {
        //check if settings.json exists
        File file = new File(settingsFileName);

        try {
            //if exist then load from file
            if (file.exists()) {
                String jsonString = Files.readString(Path.of(file.getPath()));
                this.settings = new JSONObject(jsonString);

            } else {
                //get default settings
                String jsonString = new String(
                        SettingsManager.class.getClassLoader().getResourceAsStream(settingsFi
                );
                //load into this and save to file
                this.settings = new JSONObject(jsonString);
                saveSettingsToFile();
            }
        }catch (IOException e) {
            System.out.println("Failed to load settings");
        }
        System.out.println(this.settings);//TODO remove?
    }

    public JSONObject getSettings() {
        return settings;
    }

    public synchronized void setValue(String key, Object value) {
        this.settings.put(key, value);
        saveSettingsToFile();
    }

    synchronized void saveSettingsToFile() {
        //save the changes to settings.json
        File file = new File(settingsFileName);

        try {
            PrintWriter writer = new PrintWriter(file);
            writer.write(this.settings.toString());
            writer.close();
        }catch (IOException e) {
            System.out.println(e);
            System.out.println("failed to save settings");
        }
    }

    @Override
    public String toString() {
        return this.settings.toString();
    }
}

```

```java
package edu.fau.eng.cop4331.ttt3d.server;

import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.ArrayList;

public class Server {

    /**
     * This is the server for clients
     *
     * @author Abir Faisal, Jamahl Farrington
     */
    ArrayList threads = new ArrayList<>();

    ServerSocket server;
    int serverPort = 32034;

    /**
     * Constructor
     * @throws IOException
     */
    public Server() throws IOException {
        server = new ServerSocket(serverPort);
    }

    /**
     * Run the server
     * @throws IOException
     */
    public void run() throws IOException {
        while (true) {
            System.out.println("Waiting for connection: " + server);

            //created socket waits for connection
            Socket socket = server.accept();
            System.out.println("Accepted Connection from : " + socket.getInetAddress());

            //read from socket
            ObjectInputStream ois = new ObjectInputStream(socket.getInputStream());
            System.out.println(ois.readAllBytes());

            //process the input

            //respond to the client
            ObjectOutputStream oos = new ObjectOutputStream(socket.getOutputStream());
            oos.writeObject(new String("Hello World"));


            //close the connection
            ois.close();
            oos.close();
            socket.close();
        }
    }


    /**
     * TODO write debug info without blocking
     * @param str String you want to print to terminal
     * @param log true=append to log file, false=do nothing
     */
    void nonBlockingPrintln(String str, boolean log) {

        if (log) {
            //TODO log to file
        }
    }



}

```

```java
package edu.fau.eng.cop4331.ttt3d.server;

//TODO make abstract
public interface Service {
    Object getResponse();
}

```

```java
package edu.fau.eng.cop4331.ttt3d.server.services;

public class ChatService {
    /**
     * Handle the message sent from a client
     * @param message Object that will be deserialized into a String
     */
    void processMessage(Object message){};
}

```

```java
package edu.fau.eng.cop4331.ttt3d.server.services;

public class TTT3DService {

    /**
     * Handle the move sent from a client
     * @param move Object that will be deserialized into a gameState: int[][][]
     */
    void handleGame(Object move){
    }
}

```

File - settings.json

```json
{
  "defaultServer": "127.0.0.1",
  "userDefinedServer": "0.0.0.0",
  "defaultPort": "32034",
  "userDefinedPort": "1",
  "rateLimitSeconds": "1"
}
```

```java
import edu.fau.eng.cop4331.ttt3d.app.App;


import org.junit.*;

import javax.swing.*;

public class AppTest {


    public AppTest(){

    }


    @BeforeClass
    public static void setUpClass() {

    }

    @AfterClass
    public static void tearDownClass() {

    }

    @Before
    public void setUp(){

    }

    @After
    public void tearDown() {

    }



    //Test player ID generation
    @Test
    public void playerIDTest() throws Exception {
        App app = App.getInstance();
        String pidb = "";
        byte[] playerID = app.getClientID();

        for (int i = 0; i < playerID.length; i++) {
            pidb += playerID[i] + " ";
        }

        System.out.println("player ID Bytes: " + pidb);
    }


    @Test
    public void test() {
        JOptionPane.showInputDialog("hello");
        JOptionPane.showConfirmDialog(null, "message", "tiTle", 1, 2);

    }


}

```

```java
public class ViewTest {
}

```

```java
import edu.fau.eng.cop4331.ttt3d.app.Model;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;


public class ModelTest {

    //Serialize Deserialize test
    @Test
    public void serDeserTest() throws Exception {
//        Model m1 = new Model("Model1");
//        System.out.println("Player ID: " + m1.getplayerIDasString());

//        ObjectOutputStream objectOutputStream = new ObjectOutputStream();

//        System.out.println(m1.getplayerID());

//        assertEquals(m1.hashCode(), m1.hashCode());
    }


}
```

```java
import edu.fau.eng.cop4331.ttt3d.server.Server;
import org.junit.jupiter.api.Test;

import java.io.IOException;

public class ServerTest {




    @Test
    public void testServer() throws IOException {
        Server server = new Server();
//        server.run();

    }

}

```

```java
import org.junit.jupiter.api.Test;

import javax.crypto.Cipher;
import javax.crypto.CipherOutputStream;
import javax.crypto.CipherSpi;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.spec.IvParameterSpec;
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.io.OutputStream;
import java.math.BigDecimal;
import java.math.BigInteger;
import java.security.NoSuchAlgorithmException;
import java.util.Arrays;
import java.util.Random;

public class AliceAndBob {

    BigInteger[] genPrimesArray(int len) {
        BigInteger[] secretNumbers = new BigInteger[len];

        for (int i = 0; i < len; i++) {
            secretNumbers[i] = BigInteger.probablePrime(16, new Random());
        }
        return secretNumbers;
    }

    BigInteger[] genSecretMods(BigInteger[] commonMods, BigInteger[] commonBases, BigInteger[
        BigInteger[] secretSauce = new BigInteger[secretMods.length];

        for (int i = 0; i < secretMods.length; i++) {
            BigInteger base = commonBases[i];
            BigInteger mod = secretMods[i]; //exponent

            //= base^mod % common_mod
            secretSauce[i] = base.modPow(mod, commonMods[i]);
        }
        return secretSauce;
    }


    private class TestObject {
        //object for encrypt/decrypt test
        private int i;
        public TestObject(int i){
            this.i = i;
        }
        public int getI() {
            return i;
        }
        public TestObject setI(int i) {
            this.i = i;
            return this;
        }
    }


    @Test
    public void simDiffieHellmanKeyExchange() throws Exception {

        int len = 1;

        BigInteger[] pubMods = genPrimesArray(len);
        BigInteger[] pubBases = genPrimesArray(len);

        System.out.println("pubMods: " + Arrays.toString(pubMods));
        System.out.println("pubBases: " +  Arrays.toString(pubBases));

        //Alice's Secret
        BigInteger[] alicePrivateMods = genPrimesArray(len);
        System.out.println("aliceSecret: " +  Arrays.toString(alicePrivateMods));

        //Bob's Secret
        BigInteger[] bobPrivateMods = genPrimesArray(len);
        System.out.println("bobSecret: " +  Arrays.toString(bobPrivateMods));

        //generate a public key given the public and private mods
        BigInteger[] alicePubMods = genSecretMods(pubMods, pubBases, alicePrivateMods);
        System.out.println("alicePubMods: " +  Arrays.toString(alicePubMods));

        BigInteger[] bobPubMods = genSecretMods(pubMods, pubBases, bobPrivateMods);
        System.out.println("bobPubMods:   " +  Arrays.toString(bobPubMods));

        //Alice x Bob Key Exchange and Mix
        BigInteger[] commonSecret1 = genSecretMods(pubMods, bobPubMods, alicePrivateMods);
        System.out.println("commonSecret1: " +  Arrays.toString(commonSecret1));

        //Bob x Alice Key Exchange and Mix
        BigInteger[] commonSecret2 = genSecretMods(pubMods, alicePubMods, bobPrivateMods);
        System.out.println("commonSecret2: " +  Arrays.toString(commonSecret2));


        //Test Encryption/Decryption
        TestObject testObject = new TestObject(123);


        String cipherMode = "AES/CBC/PKCS5Padding";
        Cipher cipher;
        cipher = Cipher.getInstance(cipherMode);
        //TODO cipher.init(Cipher.ENCRYPT_MODE, secretKey, IvParameterSpec);

        OutputStream outputStream = null; //TODO

        CipherOutputStream cipherOutputStream = new CipherOutputStream(outputStream, cipher)

        cipherOutputStream.write(0);



        for (int i = 0; i < len; i++) {
            assert commonSecret1[i].equals(commonSecret2[i]) : "Secrets are not common";
        }
    }

}

```

```java
public class ControllerTest {
}

```

```java
import org.junit.jupiter.api.Test;

import java.util.Random;

public class PerformanceTest {




    //Standard array compare
    //Built in array compare
    //Arrays.equals()
    //Arrays.
    //Vector

    private int[] gameState1D = new int[27];
    private int[][][] getGameState3D = new int[3][3][3];

    private void initRandGameState() {
        Random r = new Random();

        for (int i = 0; i < gameState1D.length; i++) {
            //randomly assign a 1, 0, or -1
            int a = (r.nextBoolean()) ? 1:-1; // X or O
            this.gameState1D[i] = r.nextBoolean() ? a : 0; //played or empty
        }
    }



    private void checkWinnerAlgo3DV2(int[][][] gameState3D) {

        for (int z = 0; z < gameState3D.length; z++) {

            int xSum = 0;

            for (int x = 0; x < gameState3D[z].length; x++) {

                int ySum = 0;

                for (int y = 0; y < gameState3D[z][x].length; y++) {
                    int a = gameState3D[x][y][z];
                    xSum += a;
                    ySum += a;

                }

                if (ySum == 3) {
                    System.out.println("winner found");
                } else ySum = 0; //reset ySum
            }

            if (xSum == 3) {
                System.out.println("winner found");
            } else xSum = 0; //reset ySum
        }
    }


    //TODO try 3d array and compare performance
    private void checkWinnerAlgo3dV1(int[][][] gamestate3D) {
        //TODO find better names for these variables

        int originH;
        int originV;
        int pos2;
        int pos3;
        int pos4;
        int pos5;

        //for each layer as i
        for (int i = 0; i < 3; i++) {

            //check horizontal and vertical wins
            for (int j = 0; j < 3; j++) {
                originH = gamestate3D[0][j][i];
                pos2 = gamestate3D[1][j][i];
                pos3 = gamestate3D[2][j][i];

                originV = gamestate3D[j][0][i];
                pos4 = gamestate3D[j][1][i];
                pos5 = gamestate3D[j][2][i];

                // if these values are 3 or -3 we know
                // there is a winner
                // and that either X(3) or O(-3) has won
                int hWinner = originH + pos2 + pos3;

                int vWinner = originV + pos4 + pos5;

            }
        }

        //TODO check diagonals



    }


    //Check 1D gamestate
    private void checkWinnerAlgoV1(int[] gamestate){

        //check horizontal wins for each layer in the cube

        //TODO REFACTOR, maybe each check should be it's own method?

        int gs1 = 0;
        int gs2 = 0;
        int gs3 = 0;

        int i = 0;

        while (i < 27) {
            gs1 = gamestate[i];
            gs2 = gamestate[i+1];
            gs3 = gamestate[i+2];
            System.out.println("Game State: " + i + " " + gs1 + gs2 + gs3);

            if ((gs1 == gs2) && (gs2 == gs3)) {
                System.out.println("Horizontal win: " + gs1 + gs2 + gs3);
            }
            i += 3;
        }

        //check vertical wins
        System.out.println("check Horizontal wins");
        i = 0;
        int j = 0;
        while (i < 27) {
            j = 0;
            while (j < 3) {
                gs1 = gamestate[i+j];
                gs2 = gamestate[i+j+3];
                gs3 = gamestate[i+j+6];

                System.out.println("Game State: " + i + " " + j + " " + gs1 + gs2 + gs3);

                if ((gs1 == gs2) && (gs2 == gs3)) {
                    System.out.println("Horizontal win: " + gs1 + gs2 + gs3);
                }

                j++;
            }
            i += 9;
        }

        //check side direction wins (Z-Axis wins)
        System.out.println("check Z-Axis wins");
        i=0;
        while (i < 9) {
            gs1 = gamestate[i];
            gs2 = gamestate[i+9];
            gs3 = gamestate[i+18];

            System.out.println("Game State: " + i + " " + gs1 + gs2 + gs3);

            if ((gs1 == gs2) && (gs2 == gs3)) {
                System.out.println("Horizontal win: " + gs1 + gs2 + gs3);
            }
            i++;
        }

        //Check diagonal wins

        //Check diagonal wins from front
        System.out.println("\nCheck diagonal wins from front");
        i=0;
        while (i < 27) {
            gs1 = gamestate[i];
            gs2 = gamestate[i+4];
            gs3 = gamestate[i+8];

            System.out.println("Game State \\: " + i + " " + gs1 + gs2 + gs3);

            //check other way
            gs1 = gamestate[i+2];
            //gs2 = gamestate[i+4]; // center no need to do this twice
            gs3 = gamestate[i+6];

            System.out.println("Game State /: " + i + " " + gs1 + gs2 + gs3);

            i += 9;
        }

        //Check diagonal wins from top
        System.out.println("\nCheck diagonal wins from top");
        i=0;
        while (i < 6) {
            gs1 = gamestate[i];
            gs2 = gamestate[i+10];
            gs3 = gamestate[i+20];

            System.out.println("Game State /: " + i + " " + gs1 + gs2 + gs3);

            //check other way
            gs1 = gamestate[i+2];
            //gs2 = gamestate[i+4]; // center no need to do this twice
            gs3 = gamestate[i+18];

            System.out.println("Game State \\: " + i + " " + gs1 + gs2 + gs3);

            i += 3;
        }

        //Check diagonal wins from side
        System.out.println("\nCheck diagonal wins from side");
        i=0;
        while (i < 3) {
            gs1 = gamestate[i];
            gs2 = gamestate[i+12];
            gs3 = gamestate[i+24];

            System.out.println("Game State /: " + i + " " + gs1 + gs2 + gs3);

            //check other way
            gs1 = gamestate[i+6];
            //gs2 = gamestate[i+12]; // center no need to do this twice
            gs3 = gamestate[i+18];

            System.out.println("Game State \\: " + i + " " + gs1 + gs2 + gs3);

            i ++;
        }

        //TODO check though center 3d wins

        printGameStateLayer(1, gamestate);

    }


    public void printGameStateLayer(int layer, int[] gamestate1d) {

        String s = "";

        for (int i = (layer * 9); i < 9; i++) {
            s += gamestate1d[i] + " ";
        }

        System.out.println("Game layer:" + layer + " " + s);
    }




    @Test
    public void perfTest() {
        initRandGameState();
        //byte[][][] anotherGameState = this.gamestate.clone();

        long startTime = 0;
        long endTime = 0;
        long diff = 0;
        long i = 1;
        long avg = 0;

        for (int j = 0; j < i; j++) {
            startTime = System.nanoTime();

            checkWinnerAlgoV1(this.gameState1D);

            endTime = System.nanoTime();
            diff = endTime - startTime;
            avg = (avg + diff) / 2;
        }


        System.out.println("avg ns : " + avg);
    }


}

```

```java
public class ServerTortureTest {

}
```

```java
import org.junit.jupiter.api.Test;

public class FunctionPrototypes {


    //used for prototpying methods and functions before implimenting into production


    //Transforms game from 3D to 1D
    @Test
    public void transform3Dto1D(int[][][] gameState3D) {
        int[] gamestate2D = new int[27];

        for (int z = 0; z < 3; z++) {
            for (int x = 0; x < 3;x++) {
                for (int y = 0; y < 3; y++) {
                    gamestate2D[x+y+z] = gameState3D[x][y][z];
                }
            }
        }
//        return gamestate2D;
    }

}

```

```java
import edu.fau.eng.cop4331.ttt3d.util.SettingsManager;
import org.junit.jupiter.api.Test;

import java.util.Objects;
import java.util.Random;

public class SettingsManagerTest {


    /**
     * Test of settings load/save successfully.
     */
    @Test
    void settingsManagerTest() {
        SettingsManager sm = SettingsManager.getInstance();
        sm.loadSettings();

        String customPort = sm.getSettings().getString("userDefinedPort");
        System.out.println("before: " + customPort);

        sm.setValue("userDefinedPort", "0");

        sm.loadSettings();
        String newCustomPort = sm.getSettings().getString("userDefinedPort");
        System.out.println("after: " + newCustomPort);

        //i guarantee that this works in runtime.
//        assert (!customPort.equals(newCustomPort)) : "Settings did not change";
    }


}
```