Team U5 - WarZone (Risk Computer Game)

SOEN-6441: Advanced Programming Practices

Build 1

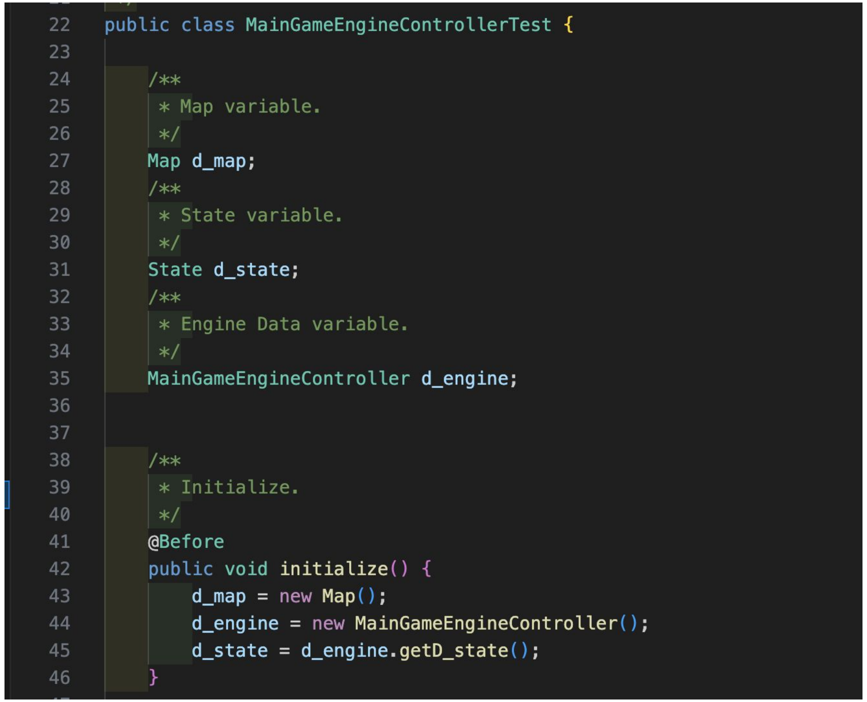
Team Members:

Abhishek Handa	(40231719)
Rajat Sharma	(40196467)
Harman Singh Jolly	(40204947)
Amanpreet Singh	(40221947)
Anurag Teckchandani	(40263724)

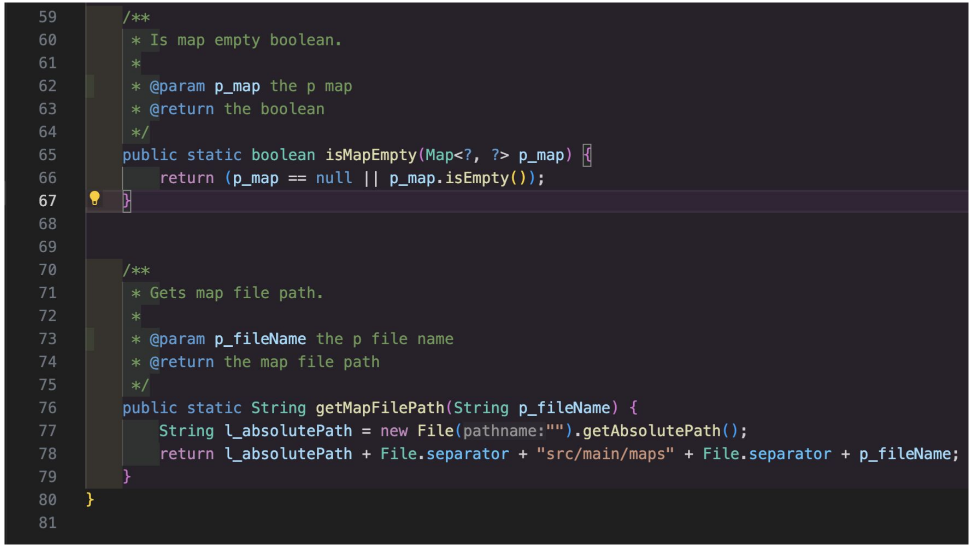
Naming Conventions

- Data members, Member functions and Method Parameters
    - All are in lower camelCase like *int thisIsExampleFunction (int p1, int p2)*
    - And data members like *int d_playerName*
- Classes
    - Class names are in upper CamelCase like
    - *MainGameEngineController.java*
- Local Variables
    - They follow lower camelCase along with "l_." stating it is a local variable like
    - *bool l_playerAvailable = "true";*
- Static Members/Constants
    - All static members are in UPPER_SNAKE_CASE letters with underscores in between the words like *int EXAMPLE_VALUE = alpha;*

Example of Naming Conventions Used (Snapshots)



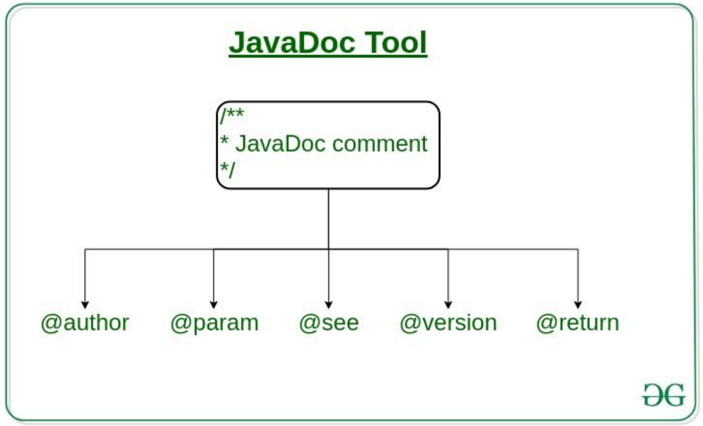
```java
public class MainGameEngineControllerTest {

    /**
     * Map variable.
     */
    Map d_map;
    /**
     * State variable.
     */
    State d_state;
    /**
     * Engine Data variable.
     */
    MainGameEngineController d_engine;


    /**
     * Initialize.
     */
    @Before
    public void initialize() {
        d_map = new Map();
        d_engine = new MainGameEngineController();
        d_state = d_engine.getD_state();
    }
```



```java
    /**
     * Is map empty boolean.
     *
     * @param p_map the p map
     * @return the boolean
     */
    public static boolean isMapEmpty(Map<?, ?> p_map) {
        return (p_map == null || p_map.isEmpty());
    }


    /**
     * Gets map file path.
     *
     * @param p_fileName the p file name
     * @return the map file path
     */
    public static String getMapFilePath(String p_fileName) {
        String l_absolutePath = new File(pathname:"").getAbsolutePath();
        return l_absolutePath + File.separator + "src/main/maps" + File.separator + p_fileName;
    }
}
```

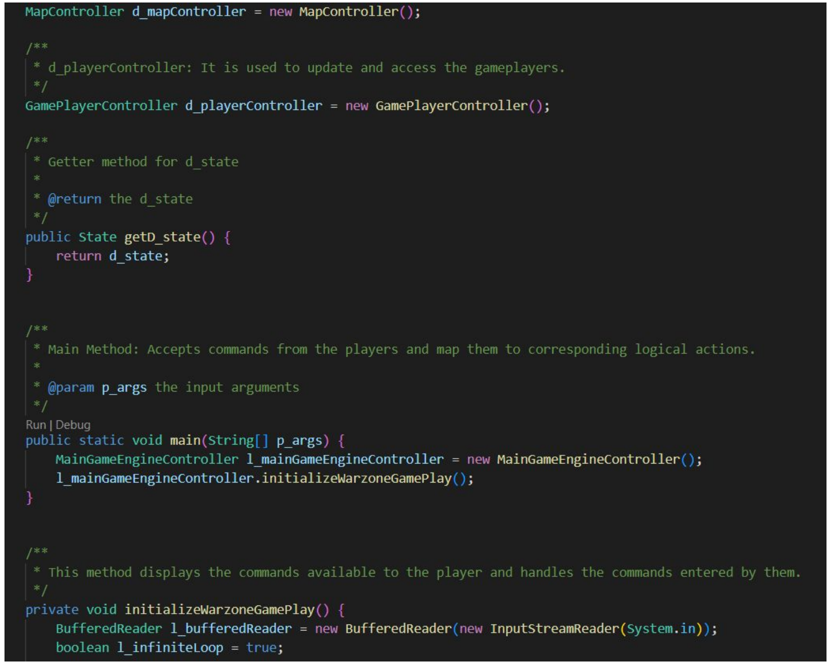
Javadocs

JavaDoc tool is a document generator tool in Java programming language for generating standard documentation in HTML format. It generates API documentation. It parses the declarations ad documentation in a set of source file describing classes, methods, constructors, and fields.

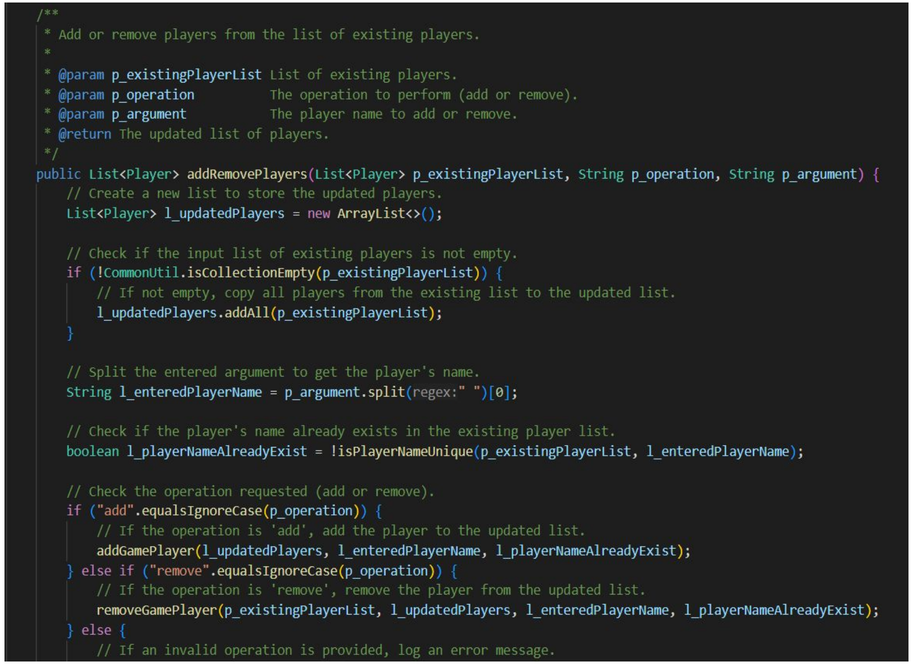


From GeeksForGeeks

Example of Javadocs Used (Snapshots)



```java
MapController d_mapController = new MapController();

/**
 * d_playerController: It is used to update and access the gameplayers.
 */
GamePlayerController d_playerController = new GamePlayerController();

/**
 * Getter method for d_state
 *
 * @return the d_state
 */
public State getD_state() {
    return d_state;
}


/**
 * Main Method: Accepts commands from the players and map them to corresponding logical actions.
 *
 * @param p_args the input arguments
 */
Run | Debug
public static void main(String[] p_args) {
    MainGameEngineController l_mainGameEngineController = new MainGameEngineController();
    l_mainGameEngineController.initializeWarzoneGamePlay();
}


/**
 * This method displays the commands available to the player and handles the commands entered by them.
 */
private void initializeWarzoneGamePlay() {
    BufferedReader l_bufferedReader = new BufferedReader(new InputStreamReader(System.in));
    boolean l_infiniteLoop = true;
```



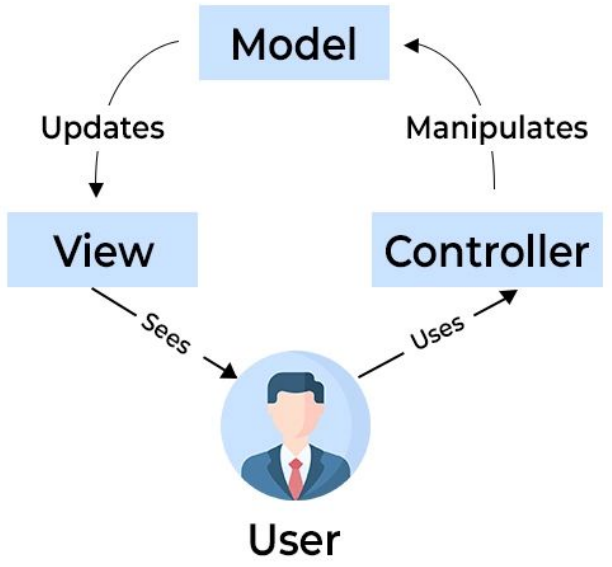
```java
/**
 * Add or remove players from the list of existing players.
 *
 * @param p_existingPlayerList List of existing players.
 * @param p_operation          The operation to perform (add or remove).
 * @param p_argument           The player name to add or remove.
 * @return The updated list of players.
 */
public List<Player> addRemovePlayers(List<Player> p_existingPlayerList, String p_operation, String p_argument) {
    // Create a new list to store the updated players.
    List<Player> l_updatedPlayers = new ArrayList<>();

    // Check if the input list of existing players is not empty.
    if (!CommonUtil.isCollectionEmpty(p_existingPlayerList)) {
        // If not empty, copy all players from the existing list to the updated list.
        l_updatedPlayers.addAll(p_existingPlayerList);
    }

    // Split the entered argument to get the player's name.
    String l_enteredPlayerName = p_argument.split(regex:" ")[0];

    // Check if the player's name already exists in the existing player list.
    boolean l_playerNameAlreadyExist = !isPlayerNameUnique(p_existingPlayerList, l_enteredPlayerName);

    // Check the operation requested (add or remove).
    if ("add".equalsIgnoreCase(p_operation)) {
        // If the operation is 'add', add the player to the updated list.
        addGamePlayer(l_updatedPlayers, l_enteredPlayerName, l_playerNameAlreadyExist);
    } else if ("remove".equalsIgnoreCase(p_operation)) {
        // If the operation is 'remove', remove the player from the updated list.
        removeGamePlayer(p_existingPlayerList, l_updatedPlayers, l_enteredPlayerName, l_playerNameAlreadyExist);
    } else {
        // If an invalid operation is provided, log an error message.
```

Architectural Design

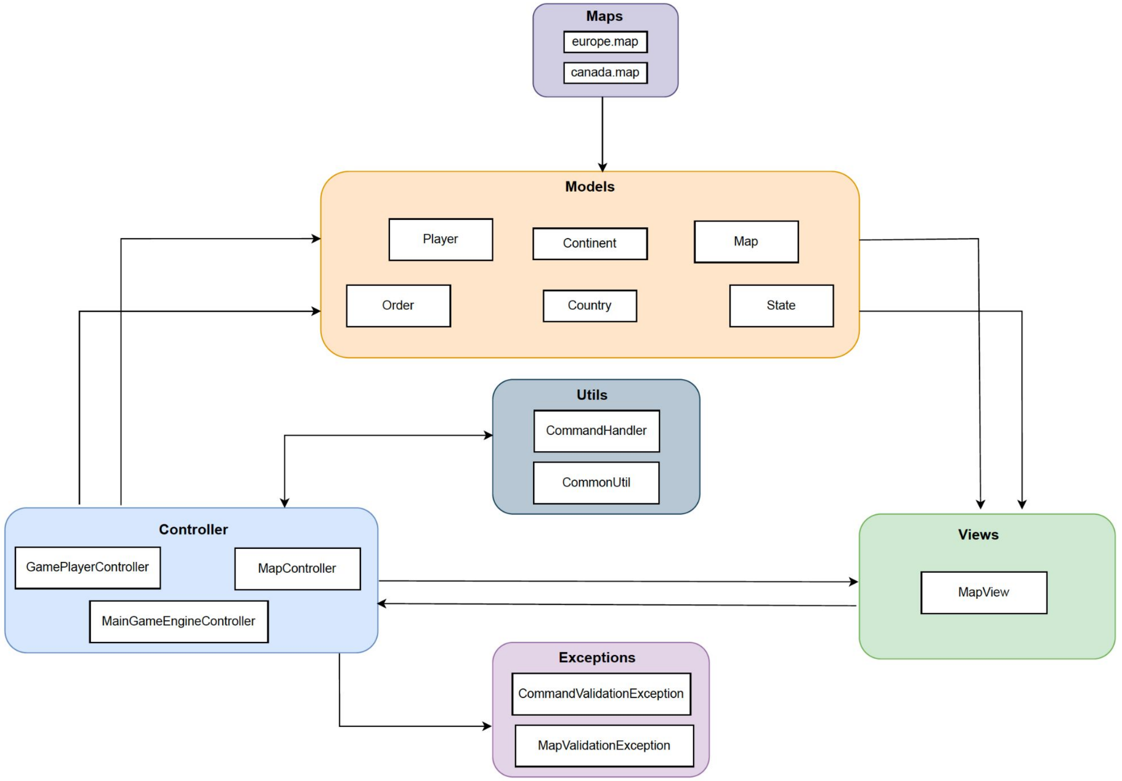
Model - It represents the business layer of application. It is an object to carry the data that can also contain the logic to update controller if data is changed.

View - It represents the presentation layer of application. It is used to visualize the data that the model contains.

Controller - It works on both the model and view. It is used to manage the flow of application, i.e. data flow in the model object and to update the view whenever data is changed.



Our Design


Maps
europe.map
canada.map
Models
Player
Continent
Map
Order
Country
State
Utils
CommandHandler
CommonUtil
Controller
GamePlayerController
MapController
MainGameEngineController
Views
MapView
Exceptions
CommandValidationException
MapValidationException