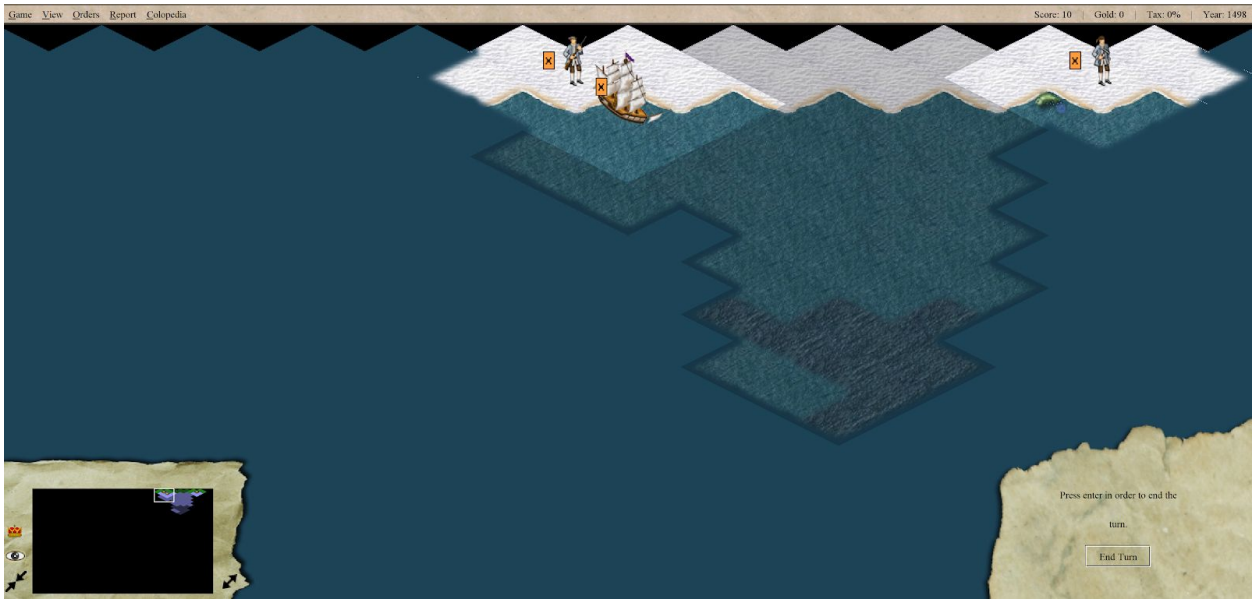
End Turn Feature

As a turn based strategy game, the end turn feature is essential to game play - without it, no other players would be able to play. FreeCol is a client-server game and any changes made to the client also necessitated changes to the server. Below you will find important methods and classes related to “End Turn”.



Client Side

InGameController is a class that regulates general game actions such as claiming a game tile, saving the game, and updating GUI based on specific in-game events. “endTurn” is the method for that calls “doEndTurn” which effectively ends the current player’s turn.

```
/**
 * End the turn command.
 *
 * Called from EndTurnAction, GUI.showEndTurnDialog
 *
 * @param showDialog If false, suppress showing the end turn dialog.
 * @return True if the turn was ended.
 */
public boolean endTurn(boolean showDialog) {
    if (!requireOurTurn()) return false;

    return doEndTurn( showDialog: showDialog
        && getClientOptions().getBoolean(ClientOptions.SHOW_END_TURN_DIALOG));
}
```

It receives a boolean parameter that indicates if a confirmation dialog window should appear to the user before actually ending the game. This method invokes “requireOurTurn”, which checks if the current player is the one responsible for the end turn action.

```
/**
 * Require that it is this client's player's turn.
 * Put up the notYourTurn message if not.
 *
 * @return True if it is our turn.
 */
private boolean requireOurTurn() {
    if (currentPlayerIsMyPlayer()) return true;
    if (getFreeColClient().isInGame()) {
        getGUI().showInformationMessage( messageId: "info.notYourTurn");
    }
    return false;
}
```

“doEndTurn” first checks if a dialog has to be shown to the user. If it does, it returns false, and then prompts the user for confirmation. If the user confirms, “endTurn” is called once again, which in turn calls “doEndTurn”, but this time “showDialog” is set to false.

```
/**
 * End the turn.
 *
 * @param showDialog Show the end turn dialog?
 * @return True if the turn ended.
 */
private boolean doEndTurn(boolean showDialog) {
    final Player player = getMyPlayer();
    if (showDialog) {
        List<Unit> units = transform(player.getUnits(), Unit::couldMove);
        if (!units.isEmpty()) {
            // Modal dialog takes over
            getGUI().showEndTurnDialog(units,
                (Boolean value) -> {
                    if (value != null && value) {
                        endTurn( showDialog: false);
                    }
                });
            return false;
        }
    }
```

After this step, the method checks if there are no other pending actions that might block the player from ending his turn, all the lingering menus are closed, the next active unit is automatically selected, report messages are cleared, and then the client requests the server to end the game.

```
// Ensure end-turn mode sticks.
moveMode = moveMode.maximize(MoveMode.END_TURN);

// Make sure all goto orders are complete before ending turn, and
// that nothing (like a LCR exploration) has cancelled the end turn.
if (!doExecuteGotoOrders()
    || moveMode.ordinal() < MoveMode.END_TURN.ordinal()) return false;

// Check for desync as last thing!
if (FreeColDebugger.isInDebugMode(FreeColDebugger.DebugMode.DESYNC)
    && DebugUtils.checkDesyncAction(getFreeColClient())) {
    logger.warning( msg: "Reconnecting on desync");
    getFreeColClient().getConnectController()
        .requestLogout(LogoutReason.RECONNECT);
    return false;
}

// Clean up lingering menus.
getGUI().closeMenus();

// Restart the selection cycle.
moveMode = MoveMode.NEXT_ACTIVE_UNIT;

// Clear outdated turn report messages.
turnReportMessages.clear();

// Inform the server of end of turn.
return askServer().endTurn();
```

Server Side
InGameController.Java

The “endTurn” method in this class receives information about which player is ending the turn. It checks if there is a winner at the end of the current turn, if not, it enters a loop to check which is the next non-dead player that should succeed the current player. If there are no other players in the game, the current player wins the game.

As part of the new turn logic, this method calls the csNextTurn() and csNewTurn() methods in the serverGame variable, which is the current instance of the ServerGame class.

All the accumulated changes that happened during the current player’s turn and all other game rules controlled by the server are then sent to all players one by one until they have all received the game updates and have updated the game state locally.

ServerGame.java

“ServerGame” is the server representation of the game. The “csNextTurn” method transitions the state of the game to the “next turn”. Within the method, Game.setTurn() is called, which sets the turn to a new instance of “Turn.”

```java
/**
 * Change to the next turn for this game.
 *
 * @param cs A {@code ChangeSet} to update.
 */
public void csNextTurn(ChangeSet cs) {
    String duration = null;
    long now = new Date().getTime();
    if (lastTime >= 0) {
        duration = ", previous turn duration = " + (now - lastTime) + "ms";
    }
    lastTime = now;

    Session.completeAll(cs);
    setTurn(getTurn().next());
    logger.finest( msg: "Turn is now " + getTurn() + "/" + duration);
    cs.add(See.all(), new NewTurnMessage(getTurn()));
}
```

Turn.java
next()

The “Turn” class is the base class that records the amount of turns that have passed since the start of the game, and can be converted to and from a date (Starting year 1492). The turn class is used to calculate mathematically how long it takes for certain actions or features to occur throughout the entirety of the game.

```java
/**
 * Get the next turn, with a turn number one greater.
 *
 * @return The new {@code Turn}.
 */
public Turn next() {
    return new Turn(turn + 1);
}
```

Downstream Diagram - Client & Server