GameHandler.java

File Structure

```
gameconfig /
          players /
                  player1.json
                  player2.json
                  player3.json
                  player4.json
          game.json
```

gameconfig: Root folder for holding the directories and files necessary for java persistence
players: A directory that holds the player.json files
playerX.json: Holds all the information regarding each player
game.json: Contains all of the data regarding the game status (tiles etc)

Continuing a game

An instance of **GameHandler** is created

↓

User clicks to continue a game

↓

Run class method **ContinueGame** that checks if the local files are valid and contain data from a previous ongoing game. If so, it runs **Init** that sets the class up for grabbing all the necessary data. If not, it will return false and then you can continue with creating a new game as the data is invalid.

↓

Run class method **getAllPlayersFromFile** that will return **Player[]** containing Player objects that will contain all of the player's data

↓

Run class method **loadBoard** that will return a **Game** object.
Note: *this game object does not serialize the Player field so you will have to assign that*

↓

As the **Game** object doesn't contain the Players, you can simply do
Game.players = getAllPlayersFromFile which will assign all the players and it'll work perfectly.

↓

The game has successfully been continued.

Starting a new game

An instance of **GameHandler** is created
↓
User clicks to create a new game
↓
Run class method **NewGame** that will run **Init** that will set up the class for the data and will also remove all files from the **gameconfig** directory that may already exist and contain data. The directories will stay, just the .json files will be removed.
↓
The class is setup and ready to then begin saving data
↓
Save data using class methods **savePlayer**, **saveAllPlayers** and **saveBoard**

Things to note

Class variables that need to be serialized require the **public** field; it will not convert if the class variables are **private**.

If a class variable doesn't need to be serialized, you can add **transient** to it and gson will not attempt to convert the field.

E.g.

```
public class Test {
      public int a;
      public transient int b;
      private int z;
}
```

Class variable **a** will be serialized by gson, but **b** will not be. **z** will also not be serialized as it is a **private** class variable.