Dungeon of Dooom

CM50109 Coursework 2

Document Two

[TEAM MEMBERS]

MATTSI JANSKY
ARYA NALINKUMAR
SELIN KUTLAMIS
ANASTASIOS GEMTOS
QIAN ZHOU
XIAOXIAO FAN

1 Contents

2 TEST PLAN

Testing is a crucial part of designing a software system. Testing enables us to make a clear concise design decisions early in development and by providing tests that match these designs ensures that if these design decisions are changed, appropriate attention and fair warning will be given to their impact. Kaner **(**2006**)** suggests that tests should not be absolute and final but should start off simply and evolve over time with the system. In line with this we aim to start with few, basic tests and add new tests as we add new features via Test Driven Development.
In this sense, we are closer to using Exploratory Testing rather than Automated Testing- that is, the responsibility for running tests belongs with the developer and not an automated system. We must vigilantly run tests ourselves.
Our intention is to exclude unit testing on Presentation layer for two reasons. On the one hand, the time we must complete this project is limited and we might not have time to do Integration tests on this layer. On the other hand, testing the graphical user interface is a difficult task that we would like to avoid because of our deadline.
We intend to test the system using Unit and Integration Tests, both Black-Box and White-Box, including boundary cases. Unit testing will likely necessitate that we use stubs and build our system in a component-oriented or modular way (ISTQB Exam Certification, n.d.).

2.1 Test-Driven Development

We will follow test-driven development, writing interfaces or stubs of our components first and tests for those unimplemented components.

2.2 Black-box Testing

2.2.1 Database connection tests

Test name	**Component being tested**	**Input**	**Expected output**	**Purpose**
ShouldConnectToDatabase	DatabaseConnection	n/a	An open database connection	Can generate a database connection
ShouldCloseDatabase	DatabaseConnection	n/a		Can close a generated connection

2.2.2 Database usage tests

Test name	**Component being tested**	**Input**	**Expected output**	**Purpose**
shouldReturnTrueIfNewPlayerValueIsAddedInDatabase	PlayerRepository	n/a	True if a new value is added on database	Can add a new player on the database.
shouldReturnTrueIfPlayerValueExistsInDatabase	PlayerRepository	n/a	True if the Player exists.	Check if a value exists on the

				database.
shouldReturnTrueIfPlayerValueIsDeleted	PlayerRepository	n/a	True if the Player was removed successfully.	Deletes an entry on the database.
shouldReturnTrueIfNewScoreValueIsAdded	ScoreRepository	n/a	True if a new value is added on the database.	Can add a new score on the database.
shouldReturnTrueIfScoreValueExistsInDatabase	ScoreRepository	n/a	True if the score exists.	Check if a value exists on the database
shouldReturnTrueIfScoreValueIsDeleted	ScoreRepository	n/a	True if the Score was removed successfully.	Deletes an entry on the database.

2.2.3 GameControllerTests

Test name	Component being tested	Input	Expected output	Purpose
shouldRespondToStatus	GameController	n/a	True if response status is equal to 200	Test the GameController response for web services (status).
shouldRespondToMove	GameController	n/a	Unimplemented	Test the GameController response for web services (move)

2.2.4 MatchControllerTests

Test name	Component being tested	Input	Expected output	Purpose
shouldGiveCurrentMatchStatus	MatchController	n/a	True if match status response from web service is equal to	Test the MatchController response for web services (status).

			expected value	
whenPlayerHasNoOngoingMatchStatusShouldReturnNull	MatchController	n/a	False if a player does not have an ongoing match	Test the MatchController response for web services (status)
shouldCreateNewMatch	MatchController	n/a	True if create a new match successfully	Test the MatchController use of MatchStatus class to create a new match.
shouldStartMatch	MatchController	n/a	True if start a new match successfully	Test the MatchController response for web services (start).
joinShouldAddUserToMatch	MatchController	n/a	True if join a match successfully	Test the MatchController response for web services (join).
listShouldListAllLobbyingMatches	MatchController	n/a	True if list of all matches is returned successfully	Test the MatchController response for web services (list).
leaveShouldRemovePlayerFromMatch	MatchController	n/a	True if match is removed from the list successfully	Test the MatchController response for web services (leave).

2.2.5 MyResourceTest

Test name	Component being tested	Input	Expected output	Purpose
testGetIt	Main class (server)	n/a	True if web service path is valid	Test if our server initialize correctly.

2.2.6 PlayerControllerTest

Test name	Component being tested	Input	Expected output	Purpose
whenDetailsAreValidShouldRegisterPlayer	PlayerController	n/a	True if player is registered successfully	Test the PlayerController response for web services (register).
whenUsernameEmptyRegisterShouldReturnValidationError	PlayerController	n/a	False if username is empty string	Test the PlayerController response for web services (register)
whenPasswordEmptyRegisterShouldReturnValidationError	PlayerController	n/a	False if password is empty string	Test the PlayerController response for web services (register)
whenPasswordTooLongRegisterShouldReturnValidationError	PlayerController	n/a	False if password is too long (>255 chars)	Test the PlayerController response for web services (register)
whenUsernameTooLongRegisterShouldReturnValidationError	PlayerController	n/a	False if username is too long (>255 chars)	Test the PlayerController response for web services (register)
whenUsernameAlreadyTakenRegisterShouldReturnValidationError	PlayerController	n/a	False if username already exists	Test the PlayerController response for web services (register)
whenDetailsValidLoginShouldReturnBlankOkStatus	PlayerController	n/a	True if register and login were successful	Test the PlayerController response for web services (register/login)

whenUsernameEmptyLoginShouldReturnValidationError	PlayerController	n/a	False if username is empty string	Test the PlayerController response for web services (login)
whenPasswordEmptyLoginShouldReturnValidationError	PlayerController	n/a	False if password is empty string	Test the PlayerController response for web services (login)
whenPasswordTooLongLoginShouldReturnValidationError	PlayerController	n/a	False if password is too long (>255 chars)	Test the PlayerController response for web services (login)
whenUsernameTooLongLoginShouldReturnValidationError	PlayerController	n/a	False if username is too long (>255 chars)	Test the PlayerController response for web services (login)
whenUsernameDoesNotExistLoginShouldReturnBlankAuthorisationError	PlayerController	n/a	False if Player does not exist in the database	Test the PlayerController response for web services (login)

2.3 White-box Testing

2.3.1 MatchListTests

Test name	Component being tested	Input	Expected output	Purpose
shouldGetLobbyingMatches	MatchList	n/a	True if we get all the initiated matches	Test the MatchList functionality by adding 2 type of matches (lobby and ongoing matches)
shouldGetMatchById	MatchList	n/a	True if we get the id of a match	Test the MatchList functionality
shouldGetMatchForPlayer	MatchList	n/a	True if a match exists for specific player	Test the MatchList functionality

2.3.2 MatchTests

Test name	Component being tested	Input	Expected output	Purpose
shouldAddCharacter	Match	n/a	True if a character is added to a match	Test the Match functionality
whenThereAreMultipleCharactersShouldGetCorrectCharacter	Match	n/a	True if we get a correct Player object when searching a match with character	Test the Match functionality

2.3.3 AuthenticationServiceTests

Test name	Component being tested	Input	Expected output	Purpose
whenUsernameDoesNotExistRegisterShouldCreatePlayerAndReturnTrue	Authentication Service	n/a	True if player entry is adding to database when register	Test the Authentication Service functionality (Register)
whenUsernameDoesExistRegisterShouldReturnFalse	Authentication Service	n/a	False if username exists on registration	Test the Authentication Service functionality (Register)
whenDetailsAreValidLoginShouldReturnTrue	Authentication Service	n/a	True if details are valid	Test the Authentication Service functionality (Login)
whenPlayerDoesNotExistLoginShouldReturnFalse	Authentication Service	n/a	False if player does not exist and try to log in	Test the Authentication Service functionality (Login)
whenPasswordIsWrongLoginShouldReturnFalse	Authentication Service	n/a	False if player's password is wrong	Test the Authentication Service functionality (Login)

2.3.4 IOServiceTests

Test name	Component being tested	Input	Expected output	Purpose
shouldGetAssetAtPath	IOService	n/a	True if path is set correctly	Test the IOService functionality (getString)
whenPathIsInvalidShouldThrowException	IOService	n/a	False if path is invalid	Test the IOService functionality (getString)
shouldParseJsonFile	IOService	n/a	True if JSON object was created succesfully	Test the IOService functionality (parseJSONObject)
whenJsonIsInvalidShouldThrownParseException	IOService	n/a	False if JSON file is invalid	Test the IOService functionality (parseJSONObject)

2.3.5 MatchServiceTests

Test name	Component being tested	Input	Expected output	Purpose
shouldCreateMatch	MatchService	n/a	True if a match was created	Test the MatchService functionality (createMatch)
WhenCreatingMatchShouldAssignRandomCharacterAndCoinPositions	MatchService	n/a	True if match was created and coins and character position were successfully set	Test the MatchService functionality (createMatch)
shouldStartMatch	MatchService	n/a	True if a match is started	Test the MatchService functionality (startMatch)

shouldGetMatchStatus	MatchService	n/a	True if match status was fetched successfully	Test the MatchService functionality (getStatus)
whenPlayerHasNoMatchGetStatusShouldReturnNull	MatchService	n/a	False if a match has no players	Test the MatchService functionality (getStatus)
endMatchShouldRemoveMatchFromMatchList	MatchService	n/a	True if a match was removed from a list when finished	Test the MatchService functionality (endMatch)
joinMatchShoulAddPlayerToMatch	MatchService	n/a	True if a player was added to a match	Test the MatchService functionality (joinMatch)
whenSqlExceptionoccursJoinMatchShouldThrowException	MatchService	n/a	False if SQL exception is thrown	Test the MatchService functionality (joinMatch)
getLobbyingMatchesShouldOnlyReturnMatchesInLobbyState	MatchService	n/a	True if only lobby matches and not ongoing matches are returned	Test the MatchService functionality (getLobbying Matces)
whenNoMatchesInLobbyStateGetLobbyingMatchesShouldReturnEmptyArray	MatchService	n/a	True if no lobby matches exist and empty array is returned	Test the MatchService functionality (getLobbying Matces)

2.3.6 MovementServiceTests

Test name	Component being tested	Input	Expected output	Purpose
shouldReturnTrueIfPlayerMovedToRightTile	MovementService	n/a	True if player moves to a valid tile (i.e. floor)	Test the MovementService functionality (move)
shouldReturnFalseIfPlayerMovedToRightTile	MovementService	n/a	False if player moves to an invalid tile (i.e. floor)	Test the MovementService functionality (move)
shouldReturnFalseIfPlayerMovesToWall	MovementService	n/a	False if player moves to a wall	Test the MovementService functionality (move)
shouldReturnTrueIfPlayerCantMoveToWall	MovementService	n/a	True if player does not move when he tries to move to a wall.	Test the MovementService functionality (move)

2.3.7 ParseServiceTests

Test name	Component being tested	Input	Expected output	Purpose
shouldGenerateMapFromJson	ParseService	n/a	True if a map was created from JSON file.	Test the ParseService functionality (parseMap)
whenJsonIsInvalidShouldThrowException	ParseService	n/a	False if JSON file is invalid	Test the ParseService functionality (parseMap)

2.3.8 StateServiceTests

Test name	Component being tested	Input	Expected output	Purpose
shouldGetCurrentStateOfGame	StateService	n/a	True if current state of game was fetched successfully	Test the StateService functionality (GetState)

2.3.9 VisibilityServiceTests

Test name	Component being tested	Input	Expected output	Purpose
shouldReturnTrueIfTheTile34IsVisible	VisibilityService	n/a	True if a specified tile is visible by the character	Test the VisibilityService functionality (createVisibleMap)
shouldReturnFalseIfTheTile77IsNotVisible	VisibilityService	n/a	False if a specified tile is not visible by the charatacter	Test the VisibilityService functionality (createVisibleMap))

3 Maintenance Guide

3.1 Overview

This maintenance guide will attempt to assist future engineers in understanding, fixing and improving the “Dungeon of Dooom” source code. cover two distinct components to the project source code- our Java code and our Javascript code.

3.2 Java Overview

We have three Java projects: The domain library, the web service, and the bot. The domain contains models and database functionality common across the projects. The service is a web API using the Jersey framework that hosts a number of “matches” in-memory each of which has a “map” and some “players”. The service has endpoints allowing players to do such things as move around the map and when the players interact with some tiles special game events occur.
The bot is a (very simple) agent that uses Jersey’s client to connect to the web service and make automated requests, in order to make a bot character move around the screen- using the exact same API that the client for humans uses.

The package structure we use follows this project separation:

- com.dod- the root Dungeon of Doom (“dod”) package.
- com.dod.db- database classes
- com.dod.game- domain classes that relate game logic, that aren’t beans or models.
- com.dod.models- models, mostly simple beans
- com.dod.service- the web service
- com.dod.service.controller- controllers for the web service
- com.dod.service.filters- Jersey API filters
- com.dod.service.model- JAXB annotated models for returning JSON data from the service.
- com.dod.service.service- services that perform game logic functions. These generic services ensure separation of functionality from controllers and allow us to re-use that functionaliy between controllers.
- com.dod.bot- the root of the bot source code
- com.dod.bot.communicators- the “Communicator” classes that the bot uses to contact the web service.

In the following sections, we detail each individual Java class using individual documentation pages generated through Javadoc.

3.3 com.dod.db.DatabaseConnection

```
public class DatabaseConnection
extends java.lang.Object
```

Stores a connection to the database using the singleton pattern

Constructors
`DatabaseConnection()`

All Methods	
Modifier and Type	Method and Description
`static void`	`Close()` Closes the connection
`static java.sql.Connection`	`getConnection()` A static connection to ensure that all sessions use the same MySql connection Could be done more intelligently with connection pooling

Methods inherited from class java.lang.Object

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

DatabaseConnection

```
public DatabaseConnection()
```

Method Detail

Close

```
public static void Close()
```

Closes the connection

getConnection

- ```
  public static java.sql.Connection getConnection()
                                              throws java.sql.SQLException
  ```

A static connection to ensure that all sessions use the same MySql connection Could be done more intelligently with connection pooling

Returns:

Connection instance

Throws:

`java.sql.SQLException` **- when the database connection cannot be established**

3.4 com.dod.db.repositories.DatabaseRepository<T>

- Direct Known Subclasses:

  PlayerRepository, **ScoreRepository**

```
public class DatabaseRepository<T>
extends java.lang.Object
```

A base class of the Repository pattern
Introduces the generic getStatement() method to reuse that code across the different repositories
```

Field Summary

Fields	
Modifier and Type	Field and Description
`protected java.sql.PreparedStatement`	`ps`

Constructor Summary

Constructors
`DatabaseRepository()`

Method Summary

All Methods	
Modifier and Type	Method and Description
`boolean`	`delete(T object)` Make a DELETE query to delete the object in question from the database
`T`	`get(T object)` Make a SELECT query to fetch the unique object in question from the database
`protected java.sql.PreparedStatement`	`getStatement(java.lang.String text)` Prepares a statement from a string using the database connection
`boolean`	`insert(T object)` Make an INSERT query to insert the object in question into the database

Field Detail

- ps

```
protected java.sql.PreparedStatement ps
```

Constructor Detail

- DatabaseRepository

```
public DatabaseRepository()
```

Method Detail

- delete

- `public boolean delete(T object)`
  `throws java.sql.SQLException`

  Make a DELETE query to delete the object in question from the database

  Parameters:
  `object` **- the object in question with the unique field (but not necessarily others) filled out**

  Returns:
  true if successful, false otherwise

  Throws:
  `java.sql.SQLException` **- when the statement fails**

- get

- `public T get(T object)`
  `throws java.sql.SQLException`

  Make a SELECT query to fetch the unique object in question from the database

  Parameters:
  `object` **- an instance of the object in question with the unique field (but not necessarily others) filled out**

  Returns:
  An instance of the object

  Throws:
  `java.sql.SQLException` **- if the statement fails or connection cannot be established**

- getStatement

- `protected java.sql.PreparedStatement getStatement(java.lang.String text)`
  `throws java.sql.SQLException`

  Prepares a statement from a string using the database connection

  Parameters:
  `text` **- the text of the statement**

  Returns:
  a PreparedStatement instance

  Throws:
  `java.sql.SQLException` **- when the statement fails**

- insert

- `public boolean insert(T object)`
  `throws java.sql.SQLException`

  Make an INSERT query to insert the object in question into the database

  Parameters:
  `object` **- the object in question**

  Returns:
  true if successful, false otherwise

Throws:
`java.sql.SQLException`

3.5 com.dod.db.repositories.IPlayerRepository

- All Known Implementing Classes:
  PlayerRepository

```
public interface IPlayerRepository
```

**Follows the Repository pattern.
Intended for selecting/inserting/deleting "Player" entries from the database.**

Method Summary

All Methods	
Modifier and Type	Method and Description
boolean	delete(Player object) Make an INSERT query to insert the Player in question into the database
Player	get(Player object) Make a SELECT query to fetch the unique Player in question from the database
boolean	insert(Player object) Make a DELETE query to delete the Player in question from the database

Method Detail

- delete

- `boolean delete(Player object)`
  `throws java.sql.SQLException`

Make an INSERT query to insert the Player in question into the database

Parameters:
`object` **- the Player in question**

Returns:
true if successful, false otherwise

Throws:
`java.sql.SQLException` **- when the statement fails**

- get

- `Player get(Player object)`
  `throws java.sql.SQLException`

Make a SELECT query to fetch the unique Player in question from the database

Parameters:

`object` **- an instance of the Player in question with the unique field (but not necessarily others) filled out**

Returns:

Player object fetched from the database

Throws:

`java.sql.SQLException` **- if the statement fails or connection cannot be established**

- insert

- `boolean insert(Player object)`
  `throws java.sql.SQLException`

Make a DELETE query to delete the Player in question from the database

Parameters:

`object` **- the Player in question with the unique field (but not necessarily others) filled out**

Returns:

true if successful, false otherwise

Throws:

`java.sql.SQLException` **- when the statement fails**

3.6 com.dod.db.repositories.IScoreRepository

- All Known Implementing Classes:

ScoreRepository

public interface `IScoreRepository`

Follows the Repository pattern.
Intended for selecting/inserting/deleting "Score" entries from the database.

Method Summary

All Methods	
Modifier and Type	Method and Description
`boolean`	`delete(Score object)` Make a DELETE query to delete the Score in question from the database
`Score`	`get(Score object)` Make a SELECT query to fetch the unique Score in question from the database
`Score[]`	`getHighestScores()` Get the 10 highest scores from database
`Score[]`	`getPlayerScores(Player object)` Get the 10 highest scores of the player
`boolean`	`insert(Score object)`

Make an INSERT query to insert the Score in question into the database

Method Detail

- delete

- `boolean delete(Score object)`
  `throws java.sql.SQLException`

  Make a DELETE query to delete the Score in question from the database

  Parameters:
  `object` **- the Score in question with the unique field (but not necessarily others) filled out**

  Returns:
  true if successful, false otherwise

  Throws:
  `java.sql.SQLException`

- get

- `Score get(Score object)`
  `throws java.sql.SQLException`

  Make a SELECT query to fetch the unique Score in question from the database

  Parameters:
  `object` **- an instance of the Score in question with the unique field (but not necessarily others) filled out**

  Returns:
  Score fetched from the database

  Throws:
  `java.sql.SQLException` **- if the statement fails or connection cannot be established**

- getHighestScores

- `Score[] getHighestScores()`
  `throws java.sql.SQLException`

  Get the 10 highest scores from database

  Returns:
  Score[] array of 10 Score objects

  Throws:
  `java.sql.SQLException` **- when the statement fails**

- getPlayerScores

- `Score[] getPlayerScores(Player object)`
  `throws java.sql.SQLException`

  Get the 10 highest scores of the player

Parameters:
`object` **- Player object**

Returns:
Score[] array of 10 Score objects

Throws:
`java.sql.SQLException` **- when the statement fails**

- insert

- `boolean insert(Score object)`
  `throws java.sql.SQLException`

Make an INSERT query to insert the Score in question into the database

Parameters:
`object` **- the Score in question**

Returns:
true if successful, false otherwise

Throws:
`java.sql.SQLException`

3.7 com.dod.db.repositories.ScoreRepository

- All Implemented Interfaces:
  IScoreRepository

```
public class ScoreRepository
extends DatabaseRepository<Score>
implements IScoreRepository
```

Implements IPlayerRepository.
Follows the Repository pattern.
Intended for selecting/inserting/deleting "Score" entries from the database.

Field Summary

- Fields inherited from class com.dod.db.repositories.DatabaseRepository

ps

Constructor Summary

Constructors
Constructor and Description
`ScoreRepository()`

Method Summary

All Methods	
Modifier and Type	Method and Description
boolean	delete(Score object) Delete a score row from database !! We should not use that.
Score	get(Score object) returns a Score based on id from the database
Score[]	getHighestScores() Get the 10 highest scores from database
Score[]	getPlayerScores(Player object) Get the 10 highest scores of the player
boolean	insert(Score scoreObject) Inserts a score value to score table of database based on player's username.

- Methods inherited from class com.dod.db.repositories.DatabaseRepository

getStatement

Constructor Detail

- ScoreRepository

```
public ScoreRepository()
```

Method Detail

- delete

- ```
  public boolean delete(Score object)
                 throws java.sql.SQLException
  ```

  Delete a score row from database !! We should not use that.

  Specified by:
  delete in interface IScoreRepository

  Overrides:
  delete in class DatabaseRepository<Score>

  Parameters:
  object **- score object to delete**

  Returns:
  true if the deletion was successful else false

  Throws:
  java.sql.SQLException **- when the statement fails**

- get

- ```
  public Score get(Score object)
           throws java.sql.SQLException
  ```

returns a Score based on id from the database

Specified by:
get in interface IScoreRepository

Overrides:
get in class DatabaseRepository<Score>

Parameters:
Score **- to be fetched must have unique identifier populated**

Returns:
Score object

Throws:
java.sql.SQLException **- when the statement fails**

- getHighestScores

- `public Score[] getHighestScores()`
  `throws java.sql.SQLException`

Get the 10 highest scores from database

Specified by:
getHighestScores in interface IScoreRepository

Returns:
Score[] array of 10 Score objects

Throws:
java.sql.SQLException **- when the statement fails**

- getPlayerScores

- `public Score[] getPlayerScores(Player object)`
  `throws java.sql.SQLException`

Get the 10 highest scores of the player

Specified by:
getPlayerScores in interface IScoreRepository

Parameters:
object **- Player object**

Returns:
Score[] array of 10 Score objects

Throws:
java.sql.SQLException **- when the statement fails**

- insert

- `public boolean insert(Score scoreObject)`
  `throws java.sql.SQLException`

Inserts a score value to score table of database based on player's username.

Specified by:
insert in interface IScoreRepository

Overrides:
insert in class DatabaseRepository<Score>

Parameters:
`scoreObject` **- current score that we need to score**

Returns:
true if insertion was successful else false

Throws:
`java.sql.SQLException` **- when the statement fails**

3.8 com.dod.db.repositories.PlayerRepository

- All Implemented Interfaces:
  IPlayerRepository

```
public class PlayerRepository
extends DatabaseRepository<Player>
implements IPlayerRepository
```

**Implements IPlayerRepository.
Follows the Repository pattern.
Intended for selecting/inserting/deleting "Player" entries from the database.**

Field Summary

- Fields inherited from class com.dod.db.repositories.DatabaseRepository

ps

Constructor Summary

Constructors
Constructor and Description
`PlayerRepository()`

Method Summary

All Methods	
Modifier and Type	Method and Description
`boolean`	`delete(Player object)` Make an INSERT query to insert the Player in question into the database
`Player`	`get(Player object)` Make a SELECT query to fetch the unique Player in question from the database
`boolean`	`insert(Player object)`

Make a DELETE query to delete the Player in question from the database

Methods *inherited* from class com.dod.db.repositories.DatabaseRepository

getStatement

Constructor Detail

- PlayerRepository

```
public PlayerRepository()
```

Method Detail

- delete

- ```
  public boolean delete(Player object)
                 throws java.sql.SQLException
  ```

  Make an INSERT query to insert the Player in question into the database

  Specified by:
  delete in interface IPlayerRepository

  Overrides:
  delete in class DatabaseRepository<Player>

  Parameters:
  object **- the Player in question**

  Returns:
  true if successful, false otherwise

  Throws:
  java.sql.SQLException **- when the statement fails**

- get

- ```
  public Player get(Player object)
              throws java.sql.SQLException
  ```

  Make a SELECT query to fetch the unique Player in question from the database

  Specified by:
  get in interface IPlayerRepository

  Overrides:
  get in class DatabaseRepository<Player>

  Parameters:
  object **- an instance of the Player in question with the unique field (but not necessarily others) filled out**

  Returns:
  Player object fetched from the database

  Throws:

java.sql.SQLException **- if the statement fails or connection cannot be established**

- insert

- `public boolean insert(Player object)`
  `throws java.sql.SQLException`

Make a DELETE query to delete the Player in question from the database

Specified by:
insert in interface IPlayerRepository

Overrides:
insert in class DatabaseRepository<Player>

Parameters:
`object` **- the Player in question with the unique field (but not necessarily others) filled out**

Returns:
true if successful, false otherwise

Throws:
`java.sql.SQLException` **- when the statement fails**

3.9 com.dod.game.IMatchList

- All Known Implementing Classes:
  MatchList

public interface IMatchList

Stores ongoing matches in memory and provides functions to access these matches.

Method Summary

All Methods	
Modifier and Type	Method and Description
void	addMatch(Match match) Returns a singleton instance of MatchList, creating it if it hasn't been initialised yet.
java.util.List<Match>	getLobbyingMatches() Gets all matches that are in the Lobbying state
Match	getMatch(java.util.UUID id) Gets a Match by a particular ID.
Match	getMatchForPlayer(java.lang.String username) Gets a match by player name.
boolean	playerHasMatch(java.lang.String username) Returns true if the player has a match in the list

void	removeMatch(java.util.UUID id) Removes the match fitting the specified ID from the list

Method Detail

- addMatch

```
void addMatch(Match match)
```

Returns a singleton instance of MatchList, creating it if it hasn't been initialised yet.

- getLobbyingMatches

```
java.util.List<Match> getLobbyingMatches()
```

Gets all matches that are in the Lobbying state

Returns:

List of Match objects

- getMatch

```
Match getMatch(java.util.UUID id)
```

Gets a Match by a particular ID. Returns null if the match is missing.

Parameters:

id **- the UUID that corresponds to the match to be fetched**

Returns:

Match

- getMatchForPlayer

```
Match getMatchForPlayer(java.lang.String username)
```

Gets a match by player name. Each player should only have one match. Returns null if player has no match.

Parameters:

username **- the username of the player**

Returns:

Match

- playerHasMatch

```
boolean playerHasMatch(java.lang.String username)
```

Returns true if the player has a match in the list

Parameters:

username **- the player's username**

Returns:

true if the player has a match in the list otherwise false

- removeMatch

```
void removeMatch(java.util.UUID id)
```

Removes the match fitting the specified ID from the list

Parameters:
`id` **- the UUID that corresponds to the particular Match to be removed**

3.10 com.dod.game.MatchList

- All Implemented Interfaces:
  IMatchList

```
public class MatchList
extends java.lang.Object
implements IMatchList

 Implementation of IMatchList
 Stores ongoing matches in memory and provides functions to access these
matches.
 Uses a singleton so that we can fetch the same object between requests
     (And because this is much easier to test than making all methods
static)
```

Constructor Summary

Constructors
`MatchList()`

Method Summary

All Methods

Modifier and Type	Method and Description
`void`	`addMatch(Match match)` Adds a match to the list
`java.util.List<Match>`	`getLobbyingMatches()` Gets all matches that are in the Lobbying state
`Match`	`getMatch(java.util.UUID id)` Gets a Match by a particular ID.
`Match`	`getMatchForPlayer(java.lang.String username)` Gets a match by player name.
`static IMatchList`	`instance()` Returns a singleton instance of MatchList, creating it if it hasn't been initialised yet.
`boolean`	`playerHasMatch(java.lang.String username)` Returns true if the player has a match in the list

void	removeMatch(java.util.UUID id) Removes the match fitting the specified ID from the list

Constructor Detail

- MatchList

```
public MatchList()
```

Method Detail

- addMatch

```
public void addMatch(Match match)
```

Adds a match to the list

Specified by:
addMatch in interface IMatchList

Parameters:
match **- the match to add**

- getLobbyingMatches

```
public java.util.List<Match> getLobbyingMatches()
```

Gets all matches that are in the Lobbying state

Specified by:
getLobbyingMatches in interface IMatchList

Returns:
List of Match objects

- getMatch

```
public Match getMatch(java.util.UUID id)
```

Gets a Match by a particular ID. Returns null if the match is missing.

Specified by:
getMatch in interface IMatchList

Parameters:
id **- the UUID that corresponds to the match to be fetched**

Returns:
Match

- getMatchForPlayer

```
public Match getMatchForPlayer(java.lang.String username)
```

Gets a match by player name. Each player should only have one match. Returns null if player has no match.

Specified by:
getMatchForPlayer in interface IMatchList

Parameters:
username **- the username of the player**

Returns:
Match

- instance

```
public static IMatchList instance()
```

Returns a singleton instance of MatchList, creating it if it hasn't been initialised yet.

Returns:
MatchList

- playerHasMatch

```
public boolean playerHasMatch(java.lang.String username)
```

Returns true if the player has a match in the list

Specified by:
playerHasMatch in interface IMatchList

Parameters:
username **- the player's username**

Returns:
true if the player has a match in the list otherwise false

- removeMatch

```
public void removeMatch(java.util.UUID id)
```

Removes the match fitting the specified ID from the list

Specified by:
removeMatch in interface IMatchList

Parameters:
id **- the UUID that corresponds to the particular Match to be removed**

3.11 com.dod.models.TileType

- All Implemented Interfaces:
java.io.Serializable, java.lang.Comparable<TileType>

public enum TileType
extends java.lang.Enum<TileType>

The type of a tile, i.e is this tile a wall, floor or something else.

Enum Constant Summary

Enum Constants
Enum Constant and Description
`Coin`
`Empty`
`Exit`
`Wall`

Method Summary

All Methods	
Modifier and Type	Method and Description
`int`	`getValue()`
`static TileType`	`valueOf(java.lang.String name)` Returns the enum constant of this type with the specified name.
`static TileType[]`	`values()` Returns an array containing the constants of this enum type, in the order they are declared.

Enum Constant Detail

- Coin

```
public static final TileType Coin
```

- Empty

```
public static final TileType Empty
```

- Exit

```
public static final TileType Exit
```

- Wall

```
public static final TileType Wall
```

Method Detail

- getValue

```
public int getValue()
```

- valueOf

```
public static TileType valueOf(java.lang.String name)
```

Returns the enum constant of this type with the specified name. The string must match ***exactly*** an identifier used to declare an enum constant in this type. (Extraneous whitespace characters are not permitted.)

Parameters:

`name` **- the name of the enum constant to be returned.**

Returns:

the enum constant with the specified name

Throws:

`java.lang.IllegalArgumentException` **- if this enum type has no constant with the specified name**

`java.lang.NullPointerException` **- if the argument is null**

- values

```
public static TileType[] values()
```

Returns an array containing the constants of this enum type, in the order they are declared. This method may be used to iterate over the constants as follows:

```
for (TileType c : TileType.values())
    System.out.println(c);
```

Returns:

an array containing the constants of this enum type, in the order they are declared

3.12 com.dod.models.Tile

public class Tile
extends java.lang.Object

A Tile represents single tile on the grid that is the Map
A Tile has a Type that indicates whether it is eg a wall, floor, coin or exit tile.
A Tile may or may not be visible

Field Summary

Fields	
Modifier and Type	Field and Description
protected int	type
protected boolean	visibility

Constructor Summary

Constructors
Constructor and Description
Tile(int type)
Tile(int type, boolean visibility)

Method Summary

All Methods	
Modifier and Type	Method and Description
`int`	`getType()`
`boolean`	`isVisible()`
`void`	`setType(int type)`
`void`	`setVisibility(boolean visibility)`
`java.lang.String`	`toString()`

Field Detail

- type

```
protected int type
```

- visibility

```
protected boolean visibility
```

Constructor Detail

- Tile

```
public Tile(int type)
```

- Tile

-
```
public Tile(int type,
            boolean visibility)
```

Method Detail

- getType

```
public int getType()
```

- isVisible

```
public boolean isVisible()
```

- setType

```
public void setType(int type)
```

- setVisibility

```
public void setVisibility(boolean visibility)
```

- toString

```
public java.lang.String toString()
```

Overrides:
toString **in class** java.lang.Object

3.13 com.dod.models.Score

public class Score
extends java.lang.Object

A Score stores the points a Player achieved when they completed a Match.
A Score as an ID in order to store the Score as a unique databaes record
A Score also has a value and the username of the player that the score is related to.

Constructor Summary

Constructors
Constructor and Description
Score(int id, java.lang.String username, int value)
Score(java.lang.String username, int value)

Method Summary

All Methods	
Modifier and Type	Method and Description
int	getId()
java.lang.String	getUsername()
int	getValue()
void	setId(int id)
void	setUsername(java.lang.String username)
void	setValue(int value)

Constructor Detail

- Score

- public Score(int id,
- java.lang.String username,
  int value)

- Score

- public Score(java.lang.String username,
  int value)

Method Detail

- getId

```
public int getId()
```

- getUsername

```
public java.lang.String getUsername()
```

- getValue

```
public int getValue()
```

- setId

```
public void setId(int id)
```

- setUsername

```
public void setUsername(java.lang.String username)
```

- setValue

```
public void setValue(int value)
```

3.14 com.dod.models.Point

```
public class Point
extends java.lang.Object
```

Bean class for storing a point (or vertex) in the map.

Field Summary

Fields	
Modifier and Type	Field and Description
`int`	`x`
`int`	`y`

Constructor Summary

Constructors
Constructor and Description
`Point()`
`Point(int x, int y)`

Method Summary

All Methods	
Modifier and Type	Method and Description

boolean	equals(java.lang.Object obj)

Field Detail

- x

```
public int x
```

- y

```
public int y
```

Constructor Detail

- Point

```
public Point()
```

- Point

  - ```
    public Point(int x,
                 int y)
    ```

Method Detail

- equals

```
public boolean equals(java.lang.Object obj)
```

Overrides:
equals **in class** java.lang.Object

3.15 com.dod.models.Player

public class Player
extends java.lang.Object

A Player represents the user that is in control of the game client
A Player can sign in with a username or password
A Player has a level and a password salt
A Player's password is always hashed

Constructor Summary

Constructors
Constructor and Description
Player(java.lang.String name)
Player(java.lang.String name, java.lang.String hashedPassword, byte[] salt)
```

Method Summary

All Methods	
Modifier and Type	Method and Description
`java.lang.String`	`getHashedPassword()`
`int`	`getLevel()`
`byte[]`	`getSalt()`
`java.lang.String`	`getUsername()`
`void`	`setHashedPassword(java.lang.String hashedPassword)`
`void`	`setLevel(int level)`
`void`	`setSalt(byte[] salt)`
`void`	`setUsername(java.lang.String value)`

- Methods inherited from class java.lang.Object

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

- Player

```
public Player(java.lang.String name)
```

- Player

```
public Player(java.lang.String name,
              java.lang.String hashedPassword,
              byte[] salt)
```

Method Detail

- getHashedPassword

```
public java.lang.String getHashedPassword()
```

- getLevel

```
public int getLevel()
```

- getSalt

```
public byte[] getSalt()
```

- getUsername

```
public java.lang.String getUsername()
```

- setHashedPassword

```
public void setHashedPassword(java.lang.String hashedPassword)
```

- setLevel

```
public void setLevel(int level)
```

- setSalt

```
public void setSalt(byte[] salt)
```

- setUsername

```
public void setUsername(java.lang.String value)
```

3.16 com.dod.models.MatchState

- All Implemented Interfaces:

java.io.Serializable, java.lang.Comparable<MatchState>

---

public enum MatchState
extends java.lang.Enum<MatchState>

The state of a Match.

Enum Constant Summary

Enum Constants
Enum Constant and Description
Ingame
Lobbying
Over

Method Summary

All Methods	
Modifier and Type	Method and Description
static MatchState	valueOf(java.lang.String name) Returns the enum constant of this type with the specified name.
static MatchState[]	values() Returns an array containing the constants of this enum type, in the order they are declared.

Enum Constant Detail

- Ingame

```
public static final MatchState Ingame
```

- Lobbying

```
public static final MatchState Lobbying
```

- Over

```
public static final MatchState Over
```

Method Detail

- valueOf

```
public static MatchState valueOf(java.lang.String name)
```

Returns the enum constant of this type with the specified name. The string must match ***exactly*** an identifier used to declare an enum constant in this type. (Extraneous whitespace characters are not permitted.)

Parameters:

`name` **- the name of the enum constant to be returned.**

Returns:

the enum constant with the specified name

Throws:

`java.lang.IllegalArgumentException` **- if this enum type has no constant with the specified name**

`java.lang.NullPointerException` **- if the argument is null**

values

```
public static MatchState[] values()
```

Returns an array containing the constants of this enum type, in the order they are declared. This method may be used to iterate over the constants as follows:

```
for (MatchState c : MatchState.values())
    System.out.println(c);
```

Returns:

an array containing the constants of this enum type, in the order they are declared

3.17 com.dod.models.Match

public class Match
extends java.lang.Object

A Match represents a particular collection of Players that are playing on a particular Map stored in memory
A Match has a Map
A Match has a unique ID
A Match

Constructor Summary

Constructors	
Constructor and Description	
`Match(com.dod.models.Map map)`	

Method Summary

All Methods	
Modifier and Type	Method and Description
`void`	`addCharacter(com.dod.models.Player player, com.dod.models.Point position)` Adds a Player to this Match with a new Character
`com.dod.models.Character`	`getCharacter(java.lang.String username)`
`java.util.List<com.dod.models.Character>`	`getCharactersOnTile(com.dod.models.Point point)` Returns all Characters on a particular Tile
`com.dod.models.Character`	`getCharacterWithHighestCoins()` Gets the Caracter with the highest score
`java.util.UUID`	`getId()`
`com.dod.models.Map`	`getMap()`
`java.lang.String[]`	`getPlayerNames()` Gets a list of names of each Player currently in this Match
`int`	`getScore()`
`com.dod.models.MatchState`	`getState()`
`long`	`getTimer()`
`boolean`	`hasCharacter(java.lang.String userName)` Returns where or not a character is in this Match
`void`	`removeCharacter(com.dod.models.Player player)`
`void`	`setScore(int score)`
`void`	`setState(com.dod.models.MatchState state)`
`void`	`setTimer(long timer)`
`void`	`startGame()`

Constructor Detail

- Match

```
public Match(com.dod.models.Map map)
```

Method Detail

- addCharacter

- ```
  public void addCharacter(com.dod.models.Player player,
                           com.dod.models.Point position)
  ```

  Adds a Player to this Match with a new Character

  Parameters:

  `player` **- Player the Player who will join this Match as a Character**

  `position` **- Point the position the new Character will occupy**

- getCharacter

  ```
  public com.dod.models.Character getCharacter(java.lang.String username)
  ```

- getCharactersOnTile

  ```
  public java.util.List<com.dod.models.Character> getCharactersOnTile(com.dod
  .models.Point point)
  ```

  Returns all Characters on a particular Tile

  Parameters:

  `point` **- Point the location of the Tile to check**

  Returns:

  List\ a list of Characters that are presently standing on that tile

- getCharacterWithHighestCoins

  ```
  public com.dod.models.Character getCharacterWithHighestCoins()
  ```

  Gets the Caracter with the highest score

  Returns:

  Character with the highest score

- getId

  ```
  public java.util.UUID getId()
  ```

- getMap

  ```
  public com.dod.models.Map getMap()
  ```

- getPlayerNames

  ```
  public java.lang.String[] getPlayerNames()
  ```

  Gets a list of names of each Player currently in this Match

  Returns:

  String[] array of players names

- getScore

  ```
  public int getScore()
  ```
  ```

- getState

```
public com.dod.models.MatchState getState()
```

- getTimer

```
public long getTimer()
```

- hasCharacter

```
public boolean hasCharacter(java.lang.String userName)
```

Returns where or not a character is in this Match

Parameters:
`userName` **- String the name of the Player to check**

Returns:
boolean true if the Player is in this Match otherwise false

- removeCharacter

```
public void removeCharacter(com.dod.models.Player player)
```

- setScore

```
public void setScore(int score)
```

- setState

```
public void setState(com.dod.models.MatchState state)
```

- setTimer

```
public void setTimer(long timer)
```

- startGame

```
public void startGame()
```

3.18 com.dod.models.Map

public class Map
extends java.lang.Object

A Map stores a 2-dimensional grid of Tiles.
A Map has a name, width, height and numbe rof coins total and required to win.

Field Summary

Fields

Modifier and Type	Field and Description
protected int	height
protected java.lang.String	name
protected int	numberOfCoinsNeededToWin
protected Tile[][]	tiles

protected int	totalNumberOfCoins
protected int	width

Constructor Summary

Constructors	
Constructor and Description	
Map(int width, int height)	
Map(java.lang.String name, int totalNumberOfCoins, int numberOfCoinsNeededToWin, int width, int height, Point mapSize)	

Method Summary

All Methods	
Modifier and Type	Method and Description
int	getCoinNo() The total number of coins that should be created in the map.
int	getCoinWin() The total number of coins needed to win on this map
int	getHeight()
java.lang.String	getName()
Point	getRandomFreeTilePoint() Gets a random position of a tile that is not a wall, coin or exit.
Tile	getTile(Point point)
int	getWidth()
void	setCoinNo(int coin_no) The total number of coins that should be created in the map.
void	setCoinWin(int coin_win) The total number of coins needed to win on this map
void	setName(java.lang.String name)
void	setTile(Point position, Tile tile)

Field Detail

- height

```
protected int height
```

- name

```
protected java.lang.String name
```

- numberOfCoinsNeededToWin

```
protected int numberOfCoinsNeededToWin
```

- tiles

```
protected Tile[][] tiles
```

- totalNumberOfCoins

```
protected int totalNumberOfCoins
```

- width

```
protected int width
```

Constructor Detail

- Map

- ```
  public Map(int width,
             int height)
  ```

- Map

- ```
  public Map(java.lang.String name,
  ```
- ```
             int totalNumberOfCoins,
  ```
- ```
             int numberOfCoinsNeededToWin,
  ```
- ```
             int width,
  ```
- ```
             int height,
             Point mapSize)
  ```

Method Detail

- getCoinNo

```
public int getCoinNo()
```

The total number of coins that should be created in the map.

Returns:

int

- getCoinWin

```
public int getCoinWin()
```

The total number of coins needed to win on this map

Returns:

int

- getHeight

```
public int getHeight()
```

- getName

```
public java.lang.String getName()
```

- getRandomFreeTilePoint

```
public Point getRandomFreeTilePoint()
```

Gets a random position of a tile that is not a wall, coin or exit.

Returns:

Point

- getTile

```
public Tile getTile(Point point)
```

- getWidth

```
public int getWidth()
```

- setCoinNo

```
public void setCoinNo(int coin_no)
```

The total number of coins that should be created in the map.

Parameters:

coin_no **- int**

- setCoinWin

```
public void setCoinWin(int coin_win)
```

The total number of coins needed to win on this map

Parameters:

coin_win **- int**

- setName

```
public void setName(java.lang.String name)
```

- setTile

- ```
  public void setTile(Point position,
                      Tile tile)
  ```

3.19 com.dod.models.Character

public class Character
extends java.lang.Object

A Character is a fictional entity that moves around the game world.
A Character belongs to a Player.
A Character has a position and can interact with coins and the exit.

Constructor Summary

Constructors
  ```

Constructor and Description
`Character(Point position, Player player)`

Method Summary

All Methods	
Modifier and Type	Method and Description
`void`	`addCollectedCoinsPos(Point newPoint)` Keeps track of which coins on the map this Character has collected.
`int`	`getCollectedCoins()`
`java.util.List<Point>`	`getCollectedCoinsPos()` Keeps track of which coins on the map this Character has collected.
`Player`	`getPlayer()` The Player that this Character belongs to
`Point`	`getPosition()` The player's position in the game world
`void`	`setCollectedCoins(int collectedCoins)`
`void`	`setPlayer(Player player)` The Player that this Character belongs to
`void`	`setPosition(Point position)` The player's position in the game world

Constructor Detail

- Character

  - `public Character(Point position, Player player)`

Method Detail

- addCollectedCoinsPos

`public void addCollectedCoinsPos(Point newPoint)`

Keeps track of which coins on the map this Character has collected. This enables us to leave the coin on the Map once it has been picked up, thereby allowing other players to pick it up, and yet not send the same coin to the same player's client again.

Parameters:
`newPoint` **- the Point to add to the collection**

- getCollectedCoins

`public int getCollectedCoins()`

- getCollectedCoinsPos

```
public java.util.List<Point> getCollectedCoinsPos()
```

Keeps track of which coins on the map this Character has collected. This enables us to leave the coin on the Map once it has been picked up, thereby allowing other players to pick it up, and yet not send the same coin to the same player's client again.

Returns:

a list of Point objects that represent the points on the map where the Character has collected a coin

- getPlayer

```
public Player getPlayer()
```

The Player that this Character belongs to

Returns:

Player

- getPosition

```
public Point getPosition()
```

The player's position in the game world

Returns:

Point

- setCollectedCoins

```
public void setCollectedCoins(int collectedCoins)
```

- setPlayer

```
public void setPlayer(Player player)
```

The Player that this Character belongs to

Parameters:

player **- Player**

- setPosition

```
public void setPosition(Point position)
```

The player's position in the game world

Parameters:

position **- Point**

3.20 com.dod.service.controller.ScoreController

```
@Path(value="score")
public class ScoreController
extends java.lang.Object
```

Fetches and returns the top scores

Constructor Summary

Constructors
Constructor and Description
`ScoreController()`

Method Summary

All Methods	
Modifier and Type	Method and Description
`javax.ws.rs.core.Response`	`top()` Fetches the top 10 scores across all players.

Constructor Detail

- ScoreController

```
public ScoreController()
```

Method Detail

- top

- `@GET`
- `@Produces(value="application/json")`
- `@Path(value="top")`

```
public javax.ws.rs.core.Response top()
```

Fetches the top 10 scores across all players.

Returns:

Response 200 OK with a JSON encoded ScoreboardModel or 500 if an error occurred

3.21 com.dod.service.controller.PlayerController

```
@Path(value="player")
public class PlayerController
extends java.lang.Object
```

Manages registering and logging in a player
Creates the session that other controllers can use to fetch user details

Constructor Summary

Constructors	
Constructor and Description	
`PlayerController()`	

Method Summary

All Methods	
Modifier and Type	Method and Description
`javax.ws.rs.core.Response`	`login(java.lang.String username, java.lang.String password)` Authorises a user and starts a session with them
`javax.ws.rs.core.Response`	`register(java.lang.String username, java.lang.String password)` Registers a user for the service.

Constructor Detail

- PlayerController

```
public PlayerController()
```

Method Detail

- login

- `@POST`
- `@Produces(value="text/plain")`
- `@Path(value="login")`
- `public javax.ws.rs.core.Response login(@NotNull @Length(min=1,max=255) @FormParam(value="username")`
- `java.lang.String username,`
- `@NotNull @Length(min=1,max=255) @FormParam(value="password")`

  `java.lang.String password)`

Authorises a user and starts a session with them

Parameters:

`username` **- must be unique, not empty and less than 256 characters**

`password` **- must not be empty and less than 256 characters**

Returns:

Response with blank body, 200 if successful otherwise 400 or 500

- register

- `@POST`
- `@Produces(value="text/plain")`

- `@Path(value="register")`
- `public javax.ws.rs.core.Response register(@NotNull @Length(min=1,max=255) @FormParam(value="username")`
- `java.lang.String username,`
- `@NotNull @Length(min=1,max=255) @FormParam(value="password")`

  `java.lang.String password)`

Registers a user for the service. Username must be unique.

Parameters:

`username` **- must be unique, not empty and less than 256 characters**

`password` **- must not be empty and less than 256 characters**

Returns:

Response with blank body, 200 if successful otherwise 400 or 500

3.22 com.dod.service.controller.MatchController

```
@Path(value="match")
public class MatchController
extends java.lang.Object
```

A controller to manage Matches- joining, listing, starting a new one etc.

Constructor Summary

Constructors
Constructor and Description
`MatchController()`

Method Summary

All Methods	
Modifier and Type	Method and Description
`javax.ws.rs.core.Response`	`join(java.util.UUID matchId)` Joins the Player in an ongoing Match
`javax.ws.rs.core.Response`	`leave()` Removes the Player from their current Match
`javax.ws.rs.core.Response`	`list()` Lists all currently lobbying matches in a JSON array
`javax.ws.rs.core.Response`	`newMatch(int level)` Starts a new Match in a particular level and responds with that Match's status
`javax.ws.rs.core.Response`	`result()` Fetches the result of a Match from memory

`javax.ws.rs.core.Response`	`start()` Changes a Match's status to Ingame (marking the start of the Match for all players)
`javax.ws.rs.core.Response`	`status()` Responds with the status of the player's current Match.

Constructor Detail

- MatchController

```
public MatchController()
```

Method Detail

- join

- `@POST`
- `@Produces(value="application/json")`
- `@Path(value="join")`
- `public javax.ws.rs.core.Response join(@NotNull @FormParam(value="matchId")`

`java.util.UUID matchId)`

Joins the Player in an ongoing Match

Parameters:

`matchId` **- the UUID ID of the Match, must not be null**

Returns:

Response 200 OK with the latest MatchStatus encoded in JSON

- leave

- `@POST`
- `@Produces(value="text/plain")`
- `@Path(value="leave")`

```
public javax.ws.rs.core.Response leave()
```

Removes the Player from their current Match

Returns:

Response 200 OK with a blank body

- list

- `@GET`
- `@Produces(value="application/json")`
- `@Path(value="list")`

```
public javax.ws.rs.core.Response list()
```

Lists all currently lobbying matches in a JSON array

Returns:

Response 200 OK JSON array with encoded MatchStatus for each lobbying Match

- newMatch

- @POST
- @Produces(value="application/json")
- @Path(value="new")
- public javax.ws.rs.core.Response newMatch(@NotNull @FormParam(value="level")

  int level)

Starts a new Match in a particular level and responds with that Match's status

Parameters:

level **- int the level to load for this Match, must not be null**

Returns:

Response 200 OK with MatchStatus encoded in JSON or null if a Match cannot be crated

- result

- @GET
- @Produces(value="application/json")
- @Path(value="result")

public javax.ws.rs.core.Response result()

Fetches the result of a Match from memory

Returns:

Resepons 200 OK with JSON encoded MatchResultModel

- start

- @POST
- @Produces(value="text/plain")
- @Path(value="start")

public javax.ws.rs.core.Response start()

Changes a Match's status to Ingame (marking the start of the Match for all players)

Returns:

MatchStatus encoded in JSON

- status

- @GET
- @Produces(value="application/json")
- @Path(value="status")

public javax.ws.rs.core.Response status()

Responds with the status of the player's current Match. If Player has no current Match returns a 500 error.

Returns:

Response 200 OK with MatchStatus encoded in JSON

3.23 com.dod.service.controller.GameController

```
@Path(value="game")
public class GameController
extends java.lang.Object
```

A controller to manage in-game game-related functionality ie getting the current state of the world or moving.

Constructor Summary

Constructors
Constructor and Description
`GameController()`

Method Summary

All Methods	
Modifier and Type	Method and Description
`javax.ws.rs.core.Response`	`move(java.lang.String direction)` An endpoint to request the Player's Character move once in a particular direction.
`javax.ws.rs.core.Response`	`status()` Responds with the current gamestate from the Player's Character's perspective, i.e.

Constructor Detail

- GameController

```
public GameController()
```

Method Detail

- move

- `@POST`
- `@Produces(value="application/json")`
- `@Path(value="move")`
- `public javax.ws.rs.core.Response move(@NotNull @FormParam(value="key")`

  `java.lang.String direction)`

An endpoint to request the Player's Character move once in a particular direction. Responds with game status after move. If Player has no current ongoing Match returns 500 error.

Parameters:

`direction` **- a char from {W,S,A,D} pertaining to a particular direction in the WASD layout, must not be null**

Returns:

Response 200 OK with GameStateModel as a JSON object

- status

- `@GET`
- `@Produces(value="application/json")`
- `@Path(value="status")`

```
public javax.ws.rs.core.Response status()
```

Responds with the current gamestate from the Player's Character's perspective, i.e. only returning visible tiles If Player has no current ongoing Match returns 500 error.

Returns:

Response 200 OK with GameStateModel as a JSON object

3.24 com.dod.service.filters.corsFilter

```
@Provider
public class corsFilter
extends java.lang.Object
implements javax.ws.rs.container.ContainerResponseFilte
```

Constructor Summary

Constructors
Constructor and Description
`corsFilter()`

Method Summary

All Methods	
Modifier and Type	Method and Description
`void`	`filter(javax.ws.rs.container.ContainerRequestContext request, javax.ws.rs.container.ContainerResponseContext response)` Adds CORS headers to the Response before sending it

Constructor Detail

- corsFilter

```
public corsFilter()
```

Method Detail

- filter

- `public void filter(javax.ws.rs.container.ContainerRequestContext request,`
  `javax.ws.rs.container.ContainerResponseContext response)`

Adds CORS headers to the Response before sending it

Specified by:
`filter` **`in interface`** `javax.ws.rs.container.ContainerResponseFilter`

Parameters:
`request` **`- ContainerRequestContext`**

`response` **`- ContainerResponseContext`**

3.25 com.dod.service.model.TileModel

`public class` `TileModel`
`extends java.lang.Object`

A simpler Tile model just for JSON encoding

Constructor Summary

Constructors
Constructor and Description
`TileModel()`
`TileModel(int type, com.dod.models.Point position)`

Method Summary

All Methods	
Modifier and Type	Method and Description
`com.dod.models.Point`	`getPosition()`
`int`	`getType()`
`void`	`setPosition(com.dod.models.Point position)`
`void`	`setType(int type)`

Constructor Detail

- TileModel

`public TileModel()`

- TileModel

- `public TileModel(int type,`
  `com.dod.models.Point position)`

Method Detail

- getPosition

```
public com.dod.models.Point getPosition()
```

- getType

```
public int getType()
```

- setPosition

```
public void setPosition(com.dod.models.Point position)
```

- setType

```
public void setType(int type)
```

3.26 com.dod.service.model.MatchStatus

```
public class MatchStatus
extends java.lang.Object
```

Models the current state of a lobbying match.

Constructor Summary

Constructors	
Constructor and Description	
`MatchStatus()`	
`MatchStatus(com.dod.models.Match match)`	

Method Summary

All Methods	
Modifier and Type	Method and Description
`java.util.UUID`	`getId()`
`java.lang.String[]`	`getPlayerNames()`
`java.lang.String`	`getState()`
`void`	`setId(java.util.UUID id)`
`void`	`setPlayerNames(java.lang.String[] playerNames)`
`void`	`setState(java.lang.String state)`

Constructor Detail

- MatchStatus

```
public MatchStatus()
```

- MatchStatus

```
public MatchStatus(com.dod.models.Match match)
```

Method Detail

- getId

```
public java.util.UUID getId()
```

- getPlayerNames

```
public java.lang.String[] getPlayerNames()
```

- getState

```
public java.lang.String getState()
```

- setId

```
public void setId(java.util.UUID id)
```

- setPlayerNames

```
public void setPlayerNames(java.lang.String[] playerNames)
```

- setState

```
public void setState(java.lang.String state)
```

3.27 com.dod.service.model.MatchResultModel

public class MatchResultModel
extends java.lang.Object

Models the information the client needs to display the end-game screen when the game ends.

Constructor Summary

Constructors
Constructor and Description
`MatchResultModel()`
`MatchResultModel(java.lang.String winner, int winnerCoins, int score)`

Method Summary

All Methods	
Modifier and Type	Method and Description
`int`	`getScore()`
`java.lang.String`	`getWinner()`
`int`	`getWinnerCoins()`

void	setScore(int score)
void	setWinner(java.lang.String winner)
void	setWinnerCoins(int winnerCoins)

Constructor Detail

- MatchResultModel

```
public MatchResultModel()
```

- MatchResultModel

```
public MatchResultModel(java.lang.String winner,
                        int winnerCoins,
                        int score)
```

Method Detail

- getScore

```
public int getScore()
```

- getWinner

```
public java.lang.String getWinner()
```

- getWinnerCoins

```
public int getWinnerCoins()
```

- setScore

```
public void setScore(int score)
```

- setWinner

```
public void setWinner(java.lang.String winner)
```

- setWinnerCoins

```
public void setWinnerCoins(int winnerCoins)
```

3.28 com.dod.service.model.LoginModel

public class LoginModel
extends java.lang.Object

Simple model/bean used to pass information to/from the AuthorisationService

Constructor Summary

Constructors
Constructor and Description

```
LoginModel(java.lang.String userName, java.lang.String password)
```

Method Summary

All Methods	
Modifier and Type	Method and Description
`com.dod.models.Player`	`asPlayer()` Convenience method to return the LoginModel's username in the Player model
`java.lang.String`	`getPassword()`
`java.lang.String`	`getUserName()`
`void`	`setPassword(java.lang.String password)`
`void`	`setUserName(java.lang.String userName)`

- Methods inherited from class java.lang.Object

```
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString,
wait, wait, wait
```

Constructor Detail

- LoginModel

```
public LoginModel(java.lang.String userName,
                  java.lang.String password)
```

Method Detail

- asPlayer

```
public com.dod.models.Player asPlayer()
```

Convenience method to return the LoginModel's username in the Player model

Returns:

`Player`

- getPassword

```
public java.lang.String getPassword()
```

- getUserName

```
public java.lang.String getUserName()
```

- setPassword

```
public void setPassword(java.lang.String password)
```

- setUserName

```
public void setUserName(java.lang.String userName)
```

3.29 com.dod.service.model.GameStateModel

```
public class GameStateModel
extends java.lang.Object
```

Represents the current GameState. Intended to be communicated to the client via JSON encoding.

Constructor Summary

Constructors	
Constructor and Description	
`GameStateModel()`	
`GameStateModel(TileModel[] tiles, CharacterModel[] characters, CharacterModel playerCharacter, boolean hasEnded, int minNumOfCoins)`	

Method Summary

All Methods	
Modifier and Type	Method and Description
`CharacterModel[]`	`getCharacters()`
`int`	`getMinNumOfCoins()` The minimum number of coins needed to win the Match
`CharacterModel`	`getPlayerCharacter()` The Character belonging to the Player that made the request
`TileModel[]`	`getTiles()`
`boolean`	`isHasEnded()` Whether the match is ongoing- triggers the client's endgame if true
`void`	`setCharacters(CharacterModel[] characters)`
`void`	`setHasEnded(boolean hasEnded)` Whether the match is ongoing- triggers the client's endgame if true
`void`	`setMinNumOfCoins(int minNumOfCoins)` * The minimum number of coins needed to win the Match
`void`	`setPlayerCharacter(CharacterModel playerCharacter)` The Character belonging to the Player that made the request
`void`	`setTiles(TileModel[] tiles)`

Constructor Detail

- GameStateModel

```
public GameStateModel()
```

- GameStateModel

```
public GameStateModel(TileModel[] tiles,
                      CharacterModel[] characters,
                      CharacterModel playerCharacter,
                      boolean hasEnded,
                      int minNumOfCoins)
```

Method Detail

- getCharacters

```
public CharacterModel[] getCharacters()
```

- getMinNumOfCoins

```
public int getMinNumOfCoins()
```

The minimum number of coins needed to win the Match

Returns:

int

- getPlayerCharacter

```
public CharacterModel getPlayerCharacter()
```

The Character belonging to the Player that made the request

Returns:

Character

- getTiles

```
public TileModel[] getTiles()
```

- isHasEnded

```
public boolean isHasEnded()
```

Whether the match is ongoing- triggers the client's endgame if true

Returns:

boolean

- setCharacters

```
public void setCharacters(CharacterModel[] characters)
```

- setHasEnded

```
public void setHasEnded(boolean hasEnded)
```

Whether the match is ongoing- triggers the client's endgame if true

Parameters:
`hasEnded` **- boolean**

- setMinNumOfCoins

```
public void setMinNumOfCoins(int minNumOfCoins)
```

*** The minimum number of coins needed to win the Match**

Parameters:
`minNumOfCoins` **- int**

- setPlayerCharacter

```
public void setPlayerCharacter(CharacterModel playerCharacter)
```

The Character belonging to the Player that made the request

Parameters:
`playerCharacter` **- Character**

- setTiles

```
public void setTiles(TileModel[] tiles)
```

3.30 com.dod.service.model.CharacterModel

```
public class CharacterModel
extends java.lang.Object
```

A simpler model of Character for JSON encoding

Constructor Summary

Constructors	
Constructor and Description	
`CharacterModel()`	
`CharacterModel(java.lang.String playerName, int noCoins, com.dod.models.Point position)`	

Method Summary

All Methods	
Modifier and Type	Method and Description
`int`	`getNoCoins()`
`java.lang.String`	`getPlayerName()`
`com.dod.models.Point`	`getPosition()`
`void`	`setNoCoins(int noCoins)`

void	setPlayerName(java.lang.String playerName)
void	setPosition(com.dod.models.Point position)

Constructor Detail

- CharacterModel

```
public CharacterModel()
```

- CharacterModel

```
public CharacterModel(java.lang.String playerName,
                      int noCoins,
                      com.dod.models.Point position)
```

Method Detail

- getNoCoins

```
public int getNoCoins()
```

- getPlayerName

```
public java.lang.String getPlayerName()
```

- getPosition

```
public com.dod.models.Point getPosition()
```

- setNoCoins

```
public void setNoCoins(int noCoins)
```

- setPlayerName

```
public void setPlayerName(java.lang.String playerName)
```

- setPosition

```
public void setPosition(com.dod.models.Point position)
```

3.31 com.dod.service.service.VisibilityService

public class VisibilityService
extends java.lang.Object
implements IVisibilityService

Calculates the visible tiles from the perspective of a particular Character

Constructor Summary

Constructors
Constructor and Description
VisibilityService()

Method Summary

All Methods	
Modifier and Type	Method and Description
`com.dod.models.Map`	`createVisibleMap(com.dod.models.Map dungeonMap, com.dod.models.Character pchar)` Generates a copy of a Map with the correct isVisible flags set for the perspective of a particular Character

Constructor Detail

- VisibilityService

```
public VisibilityService()
```

Method Detail

- createVisibleMap

- `public com.dod.models.Map createVisibleMap(com.dod.models.Map dungeonMap, com.dod.models.Character pchar)`

Generates a copy of a Map with the correct isVisible flags set for the perspective of a particular Character

Specified by:
createVisibleMap in interface IVisibilityService

Parameters:
deungeonMap **- the Map pchar resides in**

pchar **- the Character the perspective of which we're generating visibility with**

Returns:
a copy of dungeonMap with correct isVisible flags set for the perspective of pchar

3.32 com.dod.service.service.StateService

public class StateService
extends java.lang.Object
implements IStateService

Generates a representation of the current game state form the perspective of a particular character

Constructor Summary

Constructors
Constructor and Description

```
StateService(IVisibilityService visibilityService,
com.dod.game.IMatchList matchList)
```

Method Summary

All Methods	
Modifier and Type	Method and Description
GameStateModel	GetState(com.dod.models.Player player) Generates and returns a representation of the current game state form the perspective of a particular character

- Methods inherited from class java.lang.Object

```
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString,
wait, wait, wait
```

Constructor Detail

- StateService

- `public StateService(IVisibilityService visibilityService, com.dod.game.IMatchList matchList)`

Method Detail

- GetState

```
public GameStateModel GetState(com.dod.models.Player player)
```

Generates and returns a representation of the current game state form the perspective of a particular character

Specified by:
GetState in interface IStateService

Parameters:
`player` **- Player the Player a GameStateModel will be generated for**

Returns:
GameStateModel a model of the current game state

3.33 com.dod.service.service.ParseService

public class ParseService
extends java.lang.Object
implements IParseService

Implementation of IParseService.

Constructor Summary

Constructors
Constructor and Description
`ParseService()`

Method Summary

All Methods	
Modifier and Type	Method and Description
`com.dod.models.Map`	`parseMap(org.json.simple.JSONObject input)` Parses a Map object from it's JSON encoding

Constructor Detail

- ParseService

```
public ParseService()
```

Method Detail

- parseMap

- ```
  public com.dod.models.Map parseMap(org.json.simple.JSONObject input)
                              throws java.lang.NullPointerException
  ```

Parses a Map object from it's JSON encoding

Specified by:
parseMap in interface IParseService

Parameters:
`input` **- JSONObject a JSON encoding of the Map**

Returns:
Map an initialised Map parsed from JSON

Throws:
`java.lang.NullPointerException` **- may be thrown by SimpleJson while parsing**

3.34 com.dod.service.service.MovementService

```
public class MovementService
extends java.lang.Object
implements IMovementService
```

Implementation of IMovementService

Constructor Summary
```

Constructors
Constructor and Description
`MovementService()`

Method Summary

All Methods	
Modifier and Type	Method and Description
`com.dod.models.Point`	`Move(java.lang.String direction, com.dod.models.Player player)` Moves the Player in a particular direction.

- Methods inherited from class java.lang.Object

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

- MovementService

```
public MovementService()
```

Method Detail

- Move

```
public com.dod.models.Point Move(java.lang.String direction,
                                 com.dod.models.Player player)
                          throws java.sql.SQLException
```

Moves the Player in a particular direction. Will increment player's gold if interacting with gold coins, can trigger end of the Match when player interacts with Exit.

Specified by:
Move in interface IMovementService

Parameters:
`direction` **- String a char from {W,S,A,D} pertaining to a particular direction in the WASD layout**

`player` **- Player whom's Character will be moved**

Returns:
Point that the Player has moved to

Throws:
`java.sql.SQLException` **- if the database cannot be reached or statement fails while inserting new score**

3.35 com.dod.service.service.MatchService

- All Implemented Interfaces:

  IMatchService

public class MatchService
extends java.lang.Object
implements IMatchService

Manages joining/starting/ending matches.
Makes heavy use of MatchList to store matches in memory.
Uses PlayerRepository to fetch Player data.
Uses IOService and ParseService to load levels when starting a new Match.

Constructor Summary

Constructors
Constructor and Description
`MatchService(IIOService ioService, IParseService parseService, com.dod.db.repositories.IPlayerRepository playerRepository, com.dod.game.IMatchList matchList)`

Method Summary

All Methods	
Modifier and Type	Method and Description
`MatchStatus`	`createMatch(java.lang.String userName, int level)` Creates a new Match
`void`	`endMatch(com.dod.models.Player player)` Changes a Match's state to Over
`MatchStatus[]`	`getLobbyingMatches()` Get all Matches currently in the Lobbying state
`MatchResultModel`	`getMatchResult(com.dod.models.Player player)` Gets the MatchResultModel for a finished Match todo why not remove the Player from the Match at this point rather than send another request?
`MatchStatus`	`getStatus(com.dod.models.Player player)` Returns the MatchStatus for a particular Player's Match
`void`	`joinMatch(com.dod.models.Player player, java.util.UUID matchId)` Adds the Player to a particular Match
`void`	`leaveMatch(com.dod.models.Player player)` Removes a Player from their current ongoing Match
`void`	`startMatch(com.dod.models.Player player)` Changes a Match's state to InGame

Constructor Detail

- MatchService

- public MatchService(IIOService ioService,
- IParseService parseService,
- com.dod.db.repositories.IPlayerRepository playerRepository,
  com.dod.game.IMatchList matchList)

Method Detail

- createMatch

- public MatchStatus createMatch(java.lang.String userName,
  int level)

Creates a new Match

Specified by:
createMatch in interface IMatchService

Parameters:
userName **- String username of the Player who is starting the Match**

level **- int the number of the level to load for this Match**

Returns:
MatchStatus of the newly created Match

- endMatch

public void endMatch(com.dod.models.Player player)

Changes a Match's state to Over

Specified by:
endMatch in interface IMatchService

Parameters:
player **- Player whose ongoing Match will be modified**

- getLobbyingMatches

public MatchStatus[] getLobbyingMatches()

Get all Matches currently in the Lobbying state

Specified by:
getLobbyingMatches in interface IMatchService

Returns:
MatchStatus[] array of all Matches in the Lobbying state

- getMatchResult

public MatchResultModel getMatchResult(com.dod.models.Player player)

Gets the MatchResultModel for a finished Match todo why not remove the Player from the Match at this point rather than send another request?

Specified by:
getMatchResult in interface IMatchService

Parameters:
player **- Player the Player that has a finished Match**

Returns:
MatchResultModel pertaining to the player's Match

- getStatus

```
public MatchStatus getStatus(com.dod.models.Player player)
```

Returns the MatchStatus for a particular Player's Match

Specified by:
getStatus in interface IMatchService

Parameters:
player **- Player whose ongoing Match will be fetched**

Returns:

- joinMatch

```
public void joinMatch(com.dod.models.Player player,
                      java.util.UUID matchId)
               throws java.sql.SQLException
```

Adds the Player to a particular Match

Specified by:
joinMatch in interface IMatchService

Parameters:
player **- Player whom will be added**

matchID **- UUID of the Match that player will be addd to**

Throws:
java.sql.SQLException **- thrown if Player doesn't exist or a SQL connectivity issue occurs**

- leaveMatch

```
public void leaveMatch(com.dod.models.Player player)
```

Removes a Player from their current ongoing Match

Specified by:
leaveMatch in interface IMatchService

Parameters:
player **- Player the Player whom will be removed from their ongoing Match**

- startMatch

```
public void startMatch(com.dod.models.Player player)
```

Changes a Match's state to InGame

Specified by:
startMatch in interface IMatchService

Parameters:
`player` **- Player whose ongoing Match will be modified**

3.36 com.dod.service.service.IVisibilityService

- All Known Implementing Classes:
  VisibilityService

public interface `IVisibilityService`

Calculates the visible tiles from the perspective of a particular Character

Method Summary

All Methods	
Modifier and Type	Method and Description
`com.dod.models.Map`	`createVisibleMap(com.dod.models.Map deungeonMap, com.dod.models.Character pchar)` Generates a copy of a Map with the correct isVisible flags set for the perspective of a particular Character

Method Detail

- createVisibleMap

- `com.dod.models.Map createVisibleMap(com.dod.models.Map deungeonMap, com.dod.models.Character pchar)`

Generates a copy of a Map with the correct isVisible flags set for the perspective of a particular Character

Parameters:
`deungeonMap` **- the Map pchar resides in**

`pchar` **- the Character the perspective of which we're generating visibility with**

Returns:
a copy of dungeonMap with correct isVisible flags set for the perspective of pchar

3.37 com.dod.service.service.IStateService

- All Known Implementing Classes:
  StateService

public interface `IStateService`

Generates a representation of the current game state form the perspective of a particular character

Method Summary

All Methods	
Modifier and Type	Method and Description
`GameStateModel`	`GetState(com.dod.models.Player player)` Generates and returns a representation of the current game state form the perspective of a particular character

Method Detail

- GetState

`GameStateModel GetState(com.dod.models.Player player)`

Generates and returns a representation of the current game state form the perspective of a particular character

Parameters:
`player` **- Player the Player a GameStateModel will be generated for**

Returns:
GameStateModel a model of the current game state

3.38 com.dod.service.service.IParseService

- All Known Implementing Classes:
  ParseService

public interface IParseService

Parses JSON objects- namely the Map

Method Summary

All Methods	
Modifier and Type	Method and Description
`com.dod.models.Map`	`parseMap(org.json.simple.JSONObject input)` Parses a Map object from it's JSON encoding

Method Detail

- parseMap

- `com.dod.models.Map parseMap(org.json.simple.JSONObject input)`
  `throws java.lang.NullPointerException`

Parses a Map object from it's JSON encoding

Parameters:

`input` **- JSONObject a JSON encoding of the Map**

Returns:

Map an initialised Map parsed from JSON

Throws:

`java.lang.NullPointerException` **- may be thrown by SimpleJson while parsing**

3.39 com.dod.service.service.IOService

- All Implemented Interfaces:

  IIOService

```
public class IOService
extends java.lang.Object
implements IIOService
```

Handles IO within the Service

Constructor Summary

Constructors
Constructor and Description
`IOService()`
`IOService(java.lang.String pathToAssets)`

Method Summary

All Methods	
Modifier and Type	Method and Description
`org.json.simple.JSONObject`	`getJsonObject(java.lang.String path)` Fetches an asset as parsed JSON
`java.lang.String`	`getString(java.lang.String path)` Fetches an asset as a String

Constructor Detail

- IOService

```
public IOService()
```

- IOService

```
public IOService(java.lang.String pathToAssets)
```

Method Detail

- getJsonObject

- `public org.json.simple.JSONObject getJsonObject(java.lang.String path)`
- `throws java.io.IOException,`

`org.json.simple.parser.ParseException`

Fetches an asset as parsed JSON

Specified by:
`getJsonObject` in interface `IIOService`

Parameters:
`path` **- String the path to the asset we are to fetch**

Returns:
JSONObject the parsed content of the asset

Throws:
`java.io.IOException` **- if the file is missing**

`org.json.simple.parser.ParseException` **- if the file isn't encoded in valid JSON**

- getString

- `public java.lang.String getString(java.lang.String path)`
  `throws java.io.IOException`

Fetches an asset as a String

Specified by:
`getString` in interface `IIOService`

Parameters:
`path` **- String the path to the asset we are to fetch**

Returns:
String the contents of the asset

Throws:
`java.io.IOException` **- if the file is missing**

3.40 com.dod.service.service.IMovementService

- All Known Implementing Classes:
  `MovementService`

`public interface` IMovementService

Interface for MovementService. Handles game logic to move a character from one point to another.

Method Summary

All Methods	
Modifier and Type	Method and Description
com.dod.models.Point	Move(java.lang.String direction, com.dod.models.Player player) Moves the Player in a particular direction.

Method Detail

- Move

- com.dod.models.Point Move(java.lang.String direction,
- com.dod.models.Player player)
  throws java.sql.SQLException

Moves the Player in a particular direction. Will increment player's gold if interacting with gold coins, can trigger end of the Match when player interacts with Exit.

Parameters:

direction **- String a char from {W,S,A,D} pertaining to a particular direction in the WASD layout**

player **- Player whom's Character will be moved**

Returns:

Point that the Player has moved to

Throws:

java.sql.SQLException **- if the database cannot be reached or statement fails while inserting new score**

3.41 com.dod.service.service.IMatchService

- All Known Implementing Classes:

MatchService

public interface IMatchService

Manages joining/starting/ending matches.

Method Summary

All Methods	
Modifier and Type	Method and Description
MatchStatus	createMatch(java.lang.String userName, int level) Creates a new Match
void	endMatch(com.dod.models.Player player) Changes a Match's state to Over
MatchStatus[]	getLobbyingMatches() Get all Matches currently in the Lobbying state

MatchResultModel	getMatchResult(com.dod.models.Player player) Gets the MatchResultModel for a finished Match todo why not remove the Player from the Match at this point rather than send another request?
MatchStatus	getStatus(com.dod.models.Player player) Returns the MatchStatus for a particular Player's Match
void	joinMatch(com.dod.models.Player player, java.util.UUID matchID) Adds the Player to a particular Match
void	leaveMatch(com.dod.models.Player player) Removes a Player from their current ongoing Match
void	startMatch(com.dod.models.Player player) Changes a Match's state to InGame

Method Detail

- createMatch

- MatchStatus createMatch(java.lang.String userName, int level)

Creates a new Match

Parameters:
userName **- String username of the Player who is starting the Match**
level **- int the number of the level to load for this Match**

Returns:
MatchStatus of the newly created Match

- endMatch

void endMatch(com.dod.models.Player player)

Changes a Match's state to Over

Parameters:
player **- Player whose ongoing Match will be modified**

- getLobbyingMatches

MatchStatus[] getLobbyingMatches()

Get all Matches currently in the Lobbying state

Returns:
MatchStatus[] array of all Matches in the Lobbying state

- getMatchResult

MatchResultModel getMatchResult(com.dod.models.Player player)

Gets the MatchResultModel for a finished Match todo why not remove the Player from the Match at this point rather than send another request?

Parameters:
`player` **- Player the Player that has a finished Match**

Returns:
MatchResultModel pertaining to the player's Match

- getStatus

`MatchStatus getStatus(com.dod.models.Player player)`

Returns the MatchStatus for a particular Player's Match

Parameters:
`player` **- Player whose ongoing Match will be fetched**

Returns:

- joinMatch

- `void joinMatch(com.dod.models.Player player,`
- `java.util.UUID matchID)`
  `throws java.sql.SQLException`

Adds the Player to a particular Match

Parameters:
`player` **- Player whom will be added**

`matchID` **- UUID of the Match that player will be addd to**

Throws:
`java.sql.SQLException` **- thrown if Player doesn't exist or a SQL connectivity issue occurs**

- leaveMatch

`void leaveMatch(com.dod.models.Player player)`

Removes a Player from their current ongoing Match

Parameters:
`player` **- Player the Player whom will be removed from their ongoing Match**

- startMatch

`void startMatch(com.dod.models.Player player)`

Changes a Match's state to InGame

Parameters:
`player` **- Player whose ongoing Match will be modified**

3.42 com.dod.service.service.IIOService

- All Known Implementing Classes:
  IOService

public interface `IIOService`

Handles IO within the Service

Method Summary

All Methods	
Modifier and Type	Method and Description
`org.json.simple.JSONObject`	`getJsonObject(java.lang.String path)` Fetches an asset as parsed JSON
`java.lang.String`	`getString(java.lang.String path)` Fetches an asset as a String

Method Detail

- getJsonObject

- `org.json.simple.JSONObject getJsonObject(java.lang.String path)`
- `throws java.io.IOException,`

  `org.json.simple.parser.ParseException`

  Fetches an asset as parsed JSON

  Parameters:
  `path` **- String the path to the asset we are to fetch**

  Returns:
  JSONObject the parsed content of the asset

  Throws:
  `java.io.IOException` **- if the file is missing**

  `org.json.simple.parser.ParseException` **- if the file isn't encoded in valid JSON**

- getString

- `java.lang.String getString(java.lang.String path)`
  `throws java.io.IOException`

  Fetches an asset as a String

  Parameters:
  `path` **- String the path to the asset we are to fetch**

  Returns:
  String the contents of the asset

  Throws:
  `java.io.IOException` **- if the file is missing**

3.43 com.dod.service.service.IAuthenticationService

- All Known Implementing Classes:
  AuthenticationService

public interface IAuthenticationService

Handles authenticating a user against their user/pass combo

Method Summary

All Methods	
Modifier and Type	Method and Description
`boolean`	`Login(LoginModel model)` Registers a new user
`boolean`	`Register(LoginModel model)` Registers a new user

Method Detail

- Login

`boolean Login(LoginModel model)`

Registers a new user

Parameters:
`model` **- LoginModel containing the user/pass to be authorised**

Returns:
boolean true if the user is authorised, otherwise false

- Register

`boolean Register(LoginModel model)`

Registers a new user

Parameters:
`model` **- LoginModel containing the user/pass to be registered**

Returns:
boolean true if successful otherwise false

3.44 com.dod.service.service.AuthenticationService

- All Implemented Interfaces:
IAuthenticationService

public class AuthenticationService
extends java.lang.Object
implements IAuthenticationService

Handles authenticating a user against their user/pass combo
Uses a salt, generated using a secure RNG
Uses PlayerRepository to fetch Player database details

Constructor Summary

Constructors
Constructor and Description
`AuthenticationService(com.dod.db.repositories.IPlayerRepository repository)`

Method Summary

All Methods	
Modifier and Type	Method and Description
`boolean`	`Login(LoginModel model)` Registers a new user
`boolean`	`Register(LoginModel model)` Registers a new user

Constructor Detail

- AuthenticationService

```
public AuthenticationService(com.dod.db.repositories.IPlayerRepository repository)
```

Method Detail

- Login

```
public boolean Login(LoginModel model)
```

Registers a new user

Specified by:
`Login` in interface `IAuthenticationService`

Parameters:
`model` **- LoginModel containing the user/pass to be authorised**

Returns:
boolean true if the user is authorised, otherwise false

- Register

```
public boolean Register(LoginModel model)
```

Registers a new user

Specified by:
`Register` in interface `IAuthenticationService`

Parameters:
`model` **- LoginModel containing the user/pass to be registered**

Returns:

boolean true if successful otherwise false

3.45 com.dod.service.Main

public class Main
extends java.lang.Object

Main class.

Field Summary

Fields	
Modifier and Type	Field and Description
static java.lang.String	BASE_URI

Constructor Summary

Constructors
Constructor and Description
Main()

Method Summary

All Methods	
Modifier and Type	Method and Description
static void	main(java.lang.String[] args) Main method.
static org.glassfish.grizzly.http.server.HttpServer	startServer() Starts Grizzly HTTP server exposing JAX-RS resources defined in this application.

Field Detail

- BASE_URI

public static final java.lang.String BASE_URI

See Also:

Constant Field Values

Constructor Detail

- Main

```
public Main()
```

Method Detail

- main

- ```
  public static void main(java.lang.String[] args)
                  throws java.io.IOException
  ```

  Main method.

  Parameters:
  `args` -

  Throws:
  `java.io.IOException`

- startServer

```
public static org.glassfish.grizzly.http.server.HttpServer startServer()
```

Starts Grizzly HTTP server exposing JAX-RS resources defined in this application.

Returns:
`Grizzly HTTP server.`

3.46 com.dod.bot.communicators.CommunicatorBase

- Direct Known Subclasses:
  `MatchCommunicator`, **`MoveCommunicator`**, **`stateCommunicator`**

`public class` CommunicatorBase
`extends java.lang.Object`

A base class that handles generic communication to/from the server.

Constructor Summary

Constructors
Constructor and Description
`CommunicatorBase()`

Method Summary

All Methods	
Modifier and Type	Method and Description
`protected javax.ws.rs.core.Response`	`get(javax.ws.rs.client.Invocation.Builder request)`
```

	Invokes the specified Request as a GET request
`protected javax.ws.rs.core.Response`	`get(java.lang.String path)` Sends a GET request to a particular path on the web service
`protected static javax.ws.rs.client.WebTarget`	`getTarget()`
`protected javax.ws.rs.core.Response`	`post(javax.ws.rs.client.Invocation.Builder request, javax.ws.rs.core.MultivaluedMap<java.lang.String,java.lang.String> params)` Invokes the request as a POST request with the specified parameters as form parameters.
`protected javax.ws.rs.core.Response`	`post(java.lang.String path, javax.ws.rs.core.MultivaluedMap<java.lang.String,java.lang.String> params)` Posts a web request to the specified path with the specified parameters as form parameters.
`protected javax.ws.rs.client.Invocation.Builder`	`request(java.lang.String path)` Generates a request to the specified path

- Methods inherited from class java.lang.Object

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

- *Constructor Detail*

- CommunicatorBase

`public CommunicatorBase()`

- *Method Detail*

- get

`protected javax.ws.rs.core.Response get(javax.ws.rs.client.Invocation.Builder request)`

Invokes the specified Request as a GET request

Parameters:

`request` **- Invocation.Builder a Builder that generates an Invocation of a particular web resource.**

Returns:

- get

`protected javax.ws.rs.core.Response get(java.lang.String path)`

Sends a GET request to a particular path on the web service

Parameters:

`path` **- String the path to send the GET request to**

Returns:

Response the response from the server

- getTarget

`protected static javax.ws.rs.client.WebTarget getTarget()`

- post

- `protected javax.ws.rs.core.Response post(javax.ws.rs.client.Invocation.Builder request,`

  `javax.ws.rs.core.MultivaluedMap<java.lang.String,java.lang.String> params)`

  Invokes the request as a POST request with the specified parameters as form parameters.

  Parameters:

  `request` **- Invocation.Builder a Builder that generates an Invocation of a particular web resource.**

  `params` **- MultiValuedHashMap the parameters to send with the POST request**

  Returns:

  Response the response from the service

- post

- `protected javax.ws.rs.core.Response post(java.lang.String path,`

  `javax.ws.rs.core.MultivaluedMap<java.lang.String,java.lang.String> params)`

  Posts a web request to the specified path with the specified parameters as form parameters.

  Parameters:

  `path` **- String the path to send the POST request to**

  `params` **- MultiValuedHashMap the parameters to send with the POST request**

  Returns:

  Response the response from the service

- request

`protected javax.ws.rs.client.Invocation.Builder request(java.lang.String path)`

Generates a request to the specified path

Parameters:

`path` **- String the path to request**

Returns:

Invocation.Builder a Builder that generates an Invocation of the specified web resource.

3.47 com.dod.bot.communicators.stateCommunicator

public class stateCommunicator
extends **CommunicatorBase**

Communicates status requests to the server

Constructor Summary

Constructors
Constructor and Description
`stateCommunicator()`

Method Summary

All Methods	
Modifier and Type	Method and Description
`com.dod.service.model.GameStateModel`	`getState()` Gets the current state from the web service.

- Methods inherited from class com.dod.bot.communicators.CommunicatorBase

get, **get**, **getTarget**, **post**, **post**, **request**

Constructor Detail

- stateCommunicator

```
public stateCommunicator()
```

Method Detail

- getState

```
public com.dod.service.model.GameStateModel getState()
```

Gets the current state from the web service.

Returns:

GameStateModel a model representing the game's current state.

3.48 com.dod.bot.communicators.MoveCommunicator

public class MoveCommunicator
extends **CommunicatorBase**

Communicates move requests to the server

Constructor Summary

Constructors	
Constructor and Description	
`MoveCommunicator()`	

Method Summary

All Methods	
Modifier and Type	Method and Description
`void`	`moveDirection(java.lang.String direction)` Sends a request to the web service to move in a particular direction

- Methods inherited from class com.dod.bot.communicators.CommunicatorBase

get, **get**, **getTarget**, **post**, **post**, **request**

Constructor Detail

- MoveCommunicator

```
public MoveCommunicator()
```

Method Detail

- moveDirection

```
public void moveDirection(java.lang.String direction)
```

Sends a request to the web service to move in a particular direction

Parameters:

`direction` **- String the direction to move in, a char from the set {W,A,S,D} corresponding to WASD directions.**

3.49 com.dod.bot.communicators.MatchCommunicator

public class MatchCommunicator
extends **CommunicatorBase**

Handles match requests to the server

Constructor Summary

Constructors	
Constructor and Description	
`MatchCommunicator()`	

Method Summary

All Methods	
Modifier and Type	Method and Description
void	joinMatch(java.util.UUID matchId) Sends a request to the web service to join the specified Match.

- Methods inherited from class com.dod.bot.communicators.CommunicatorBase

get, **get**, **getTarget**, **post**, **post**, **request**

- Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

- MatchCommunicator

```
public MatchCommunicator()
```

Method Detail

- joinMatch

```
public void joinMatch(java.util.UUID matchId)
```

Sends a request to the web service to join the specified Match.

Parameters:
matchId **- UUID the ID of the Match to join**

3.50 com.dod.bot.Map

public class Map
extends java.lang.Object

Map for the bot modeled on the responses from the server.
Should work in theory but not tested as we ran out of time.

Constructor Summary

Constructors
Constructor and Description
Map()

Method Summary

All Methods	
Modifier and Type	Method and Description
void	addTile(com.dod.service.model.TileModel[] tiles) Add a tile to the map.

Constructor Detail

- Map

```
public Map()
```

Method Detail

- addTile

```
public void addTile(com.dod.service.model.TileModel[] tiles)
```

Add a tile to the map. Expands the map to the correct size of necessary.

Parameters:
`tiles` **- TileModel[] a collection of Tiles to add to the Map.**

3.51 com.dod.bot.Main

public class Main
extends java.lang.Object

Gets command parameters and intitialises bot

Constructor Summary

Constructors
Constructor and Description
Main()

Method Summary

All Methods	
Modifier and Type	Method and Description
static void	main(java.lang.String[] args) **Parses the input and starts the Bot**

Constructor Detail

- Main

```
public Main()
```

Method Detail

- main

```
public static void main(java.lang.String[] args)
```

Parses the input and starts the Bot

Parameters:
`args` **- expects 1 argument of ID for match to join**

3.52 com.dod.bot.Bot

public class Bot
extends java.lang.Object

The main bot object. Makes basic decisions and uses the Communicators to enact these decisions.
Has no real intelligence at the moment. In the future we could make it much more intelligent using the Map class
to store beliefs about the world and use path-finding to hunt out gold to get the most score.

Constructor Summary

Constructors
Constructor and Description
`Bot()`

Method Summary

All Methods	
Modifier and Type	Method and Description
`void`	`play(java.util.UUID matchId)` Joins a match and then randomly picks a direction to move in every 5th of a second.

Constructor Detail

- Bot

```
public Bot()
```

Method Detail

- play

```
public void play(java.util.UUID matchId)
```

Joins a match and then randomly picks a direction to move in every 5th of a second. Stops when the Match is over.

Parameters:

`matchId` **- UUID The ID of the match to join**

3.53 Javascript

Proprietary Javascript is generally written in a single file because of the additional overhead of having multiple HTTP request to fetch various Javascript files, and also because having many separate Javascript files can introduce race conditions as different files load and execute in unpredictable times. Our proprietary Javascript file is *main.js*.

This makes it difficult to arrange Javascript in a readable manner but we've accounted for this by using Javascript namespacing **(Croll, 2010)**. We created the root namespace "game" and from there have the following namespaces:

- game.menu- functionality surrounding menu buttons and switching between menu pages
- game.auth- functionality surrounding sending authorisation requests to and from the web service
- game.constants- a central location for storing constant values
- game.func- generic functionalities used across various namespaces
- game.match- functionality regarding joining, leaving, listing etc matches.
- game.match.var- variables pertaining to matches, for keeping track of a match status or list of matches.
- game.var- variables pertaining to the game, particularly involving graphics and timesteps.
- game.var.colours- the colours of various tiles in the first game display we wrote. Deprecated since we started using bitmap graphics.

Each function is, as much as possible, named in the most literal way to describe exactly what it does- such as "setAllTilesNotVisible" or "displayMatchMenu".

The structure of our Javascript is to firstly declare each individual function and then when the document loads assigns functions to buttons, prepares the login screen and initialises a key press event listener for game interactivity.

What follows is a documentation of some but not all of the Javascript functions:

- game.func.get
    - Generic method to make a GET request. Uses xhr fields to ensure cookies are sent across domain.
    - Param "url": {string} to send the request to
    - Param "data": {string}to send with the request
    - Param "success": function to execute on success
    - Param "error": function to execute on failure
- game.func.post
- Generic method to make a POST request. Uses xhr fields to ensure cookies are sent across domain.
    - Param "url": {string}to send the request to
    - Param "data": {string}to send with the request

    - Param “success”: function to execute on success
    - Param “error”: function to execute on failure
- game.func.getApiPath
    - Constructs an url of an endpoint given the endpoint's controller and action names.
    - Param “controller”: {string} the controller to contact
    - Param “action”: {string} the action to contact
    - Returns: (string) the constructed path
- game.menu.displayScoreboard
    - Renders the scoreboard in the #score-table table.
    - Param “scoreboard”: the JSON object returned from a query to the score/top endpoint
- game.match.new
    - Starts a new Match by sending a request to the web service
- game.match.start
    - Starts the player's current Match by sending a request to the web service
- game.match.initGameScreen
    - Initialises the game screen. Resets game variables and creates a new HTML5 canvas. Begins the game loop.
- game.render
    - Renders the current game state to the canvas.
- game.updateStatus
    - Updates the current game status based on the result from a status request.
    - Will end the gam eif the status response indicates that the game is over.
- game.var.addTile
    - Adds a tile if it doesn't already exist in memory, or updates the tile if it does.
    - Will expand the size of game.var.tiles if it isn't large enough.
- game.updateGame
    - Makes a game status request if game.var.timestep has passed since the last request.
    - Loops while game.var.isRunning
- game.match.updateMatchList
    - Makes a match list request if game.match.var.timeStep has passed since the last request
- game.match.updateStatus
    - Makes a Match Status request if game.match.var.timeStep has passed since the last request.
    - Loops while game.match.var.isWaitingToStart
- game.match.displayMatchMenu
    - Generates the Match Status details on the Match screen.
    - Param “data”: Match Status as a JSON object
- game.menu.showEndGameScreen
    - Builds the end-game screen and switches to it
    - Param “result”: a MatchResultModel object

4 Implementation

Source code:

4.1 DungeonOfDooom-master\Sourcecode\project\assets\maps

4.1.1 level1.json

```
{
 "lev":
	{
	"name":"level1",
	"coin_num":30,
	"coin_win":20,
	"Width" : 26,
	"Height" : 18,
	"tiles":
	[
	{"type":0,"visibility":true},
	{"type":1,"visibility":true},
	{"type":1,"visibility":true}
	],
	"map":
	[
	[{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"
type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},
{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0
},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0}],
	[{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"
type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},
{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1
},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0}],
	[{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"
type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},
{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1
},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0}],
	[{"type":3},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"
type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},
{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1
},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0}],
	[{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"
type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},
{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1
},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0}],
	[{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"
type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},
{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1
},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0}],
	[{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"
type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},
{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1
},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0}],
	[{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"
type":0},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},
{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1
},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0}],
	[{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"
type":0},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},
{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1
},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0}],
	[{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"
```

```
type":0},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},
{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1
},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0}],
      [{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"
type":0},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},
{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1
},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0}],
      [{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"
type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},
{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1
},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0}],
      [{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"
type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},
{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1
},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0}],
      [{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"
type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},
{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1
},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0}],
      [{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"
type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},
{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1
},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":3}],
      [{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"
type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},
{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1
},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0}],
      [{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"
type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},
{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1
},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0}],
      [{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"
type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},
{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0
},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0}]
      ]
      }
}
```

4.1.2 level2.json

```
{
 "lev":
      {
      "name":"level2",
      "coin_num":25,
      "coin_win":15,
    "Width" : 26,
      "Height" : 18,
      "map":
      [
      [{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"
type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},
{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0
},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":
1},{"type":1},{"type":0},{"type":1},{"type":1},{"type":0},{"type":0},{"type
":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"ty
pe":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":
```

0},{"type":1},{"type":0},{"type":1},{"type":1},{"type":0},{"type":0},{"type
":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"ty
pe":0},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0}],

[{"type":3},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":
0},{"type":1},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type
":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"ty
pe":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":
0},{"type":1},{"type":0},{"type":1},{"type":1},{"type":0},{"type":0},{"type
":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"ty
pe":0},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":
0},{"type":1},{"type":0},{"type":1},{"type":1},{"type":0},{"type":0},{"type
":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"ty
pe":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":0},{"type":
1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0},{"type
":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"ty
pe":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":
0},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type
":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"ty
pe":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":
0},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type
":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"ty
pe":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":
0},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type
":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"ty
pe":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":
0},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type
":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"ty
pe":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":
1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type
":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":0},{"type":1},{"ty
pe":1},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":1},{"type":
1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type
":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":0},{"type":1},{"ty
pe":1},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":0},{"type":
0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type
":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"ty
pe":1},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":
1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type

```
":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"ty
pe":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":3}],

[{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":
0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":0},{"type
":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"ty
pe":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":
0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type
":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"ty
pe":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0}],
      [{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"
type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},
{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0
},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0}]
      ]
      }

}
```


4.1.3 level3.json


```
{
 "lev":
      {
      "name":"level3",
      "coin_num":25,
      "coin_win":20,
    "Width" : 26,
      "Height" : 18,
      "map":
      [
      [{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"
type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},
{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0
},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":
0},{"type":1},{"type":1},{"type":0},{"type":1},{"type":0},{"type":0},{"type
":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"ty
pe":1},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":
0},{"type":1},{"type":0},{"type":0},{"type":1},{"type":0},{"type":0},{"type
":0},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"ty
pe":1},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0}],

[{"type":3},{"type":1},{"type":1},{"type":1},{"type":0},{"type":1},{"type":
0},{"type":1},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type
":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":1},{"type":0},{"ty
pe":1},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":1},{"type":
0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type
":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"ty
pe":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":1},{"type":
0},{"type":1},{"type":0},{"type":0},{"type":1},{"type":0},{"type":0},{"type
":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"ty
pe":0},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0}],
```

[{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":
1},{"type":1},{"type":0},{"type":1},{"type":1},{"type":0},{"type":0},{"type
":0},{"type":0},{"type":1},{"type":0},{"type":1},{"type":0},{"type":1},{"ty
pe":0},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":
0},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type
":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"ty
pe":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":
0},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type
":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"ty
pe":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":
0},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type
":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"ty
pe":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":
0},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type
":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"ty
pe":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":1},{"type":
1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type
":0},{"type":0},{"type":1},{"type":0},{"type":1},{"type":0},{"type":1},{"ty
pe":1},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":1},{"type":
0},{"type":1},{"type":0},{"type":0},{"type":1},{"type":0},{"type":0},{"type
":0},{"type":0},{"type":1},{"type":0},{"type":1},{"type":0},{"type":0},{"ty
pe":1},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":1},{"type":
1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type
":0},{"type":0},{"type":1},{"type":0},{"type":1},{"type":1},{"type":1},{"ty
pe":1},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":
0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type
":1},{"type":1},{"type":1},{"type":0},{"type":1},{"type":0},{"type":0},{"ty
pe":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":3}],

[{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":0},{"type":
0},{"type":1},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0},{"type
":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"ty
pe":0},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":1},{"type":
1},{"type":1},{"type":0},{"type":1},{"type":1},{"type":0},{"type":0},{"type
":0},{"type":0},{"type":1},{"type":0},{"type":1},{"type":0},{"type":1},{"ty
pe":1},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":
0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type
":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"ty
pe":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0}]
]

```
      }
}
```

4.2 DungeonOfDooom-master\Sourcecode\project\assets\test

4.2.1 test.json

```
{
    "id": 1
}
```

4.2.2 test.asset

```
testasset :)
```

4.3 DungeonOfDooom-master\Sourcecode\project\src\bot\main\java\com\dod\bot\communicators

4.3.1 CommunicatorBase.java

```
package com.dod.bot.communicators;

import javax.ws.rs.client.*;
import javax.ws.rs.core.MultivaluedHashMap;
import javax.ws.rs.core.MultivaluedMap;
import javax.ws.rs.core.Response;
import javax.ws.rs.ext.ContextResolver;

import org.glassfish.jersey.moxy.json.MoxyJsonConfig;
import org.glassfish.jersey.moxy.json.MoxyJsonFeature;

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

/**
 * Handles communication to/from the server
 */
public class CommunicatorBase {
    private static WebTarget target;
    private static String sessionId;

    private static String username;
    private static String password;

    private static final String apiAddress = "http://localhost:8080";

    protected static WebTarget getTarget() {
        if(target == null)
            init();
        return target;
    }

    private static void init() {
        Map<String, String> namespacePrefixMapper = new HashMap<String,
String>();
        namespacePrefixMapper.put("http://www.w3.org/2001/XMLSchema-
instance", "xsi");
        MoxyJsonConfig moxyJsonConfig = new MoxyJsonConfig()
                .setNamespacePrefixMapper(namespacePrefixMapper)
                .setNamespaceSeparator(':');

        final ContextResolver<MoxyJsonConfig> jsonConfigResolver =
moxyJsonConfig.resolver();
        Client c = ClientBuilder.newBuilder()
```

```
                .register(MoxyJsonFeature.class)
                .register(jsonConfigResolver)
                .build();

        //Generate random user/pass
        username = UUID.randomUUID().toString();
        password = UUID.randomUUID().toString();

        target = c.target(apiAddress);
        sessionId = registerUserAndGetSessionId(username, password);
    }

    private static String registerUserAndGetSessionId(String username,
String password) {
        MultivaluedMap<String,       String>       formData       =       new
MultivaluedHashMap<String, String>();
        formData.add("username", username);
        formData.add("password", password);
        Response                    registerResponse                    =
getTarget().path("player/register").request().post(Entity.form(formData));

        //get the sessionId so we can send authorised session cookies with
requests
        return registerResponse.getCookies().get("JSESSIONID").getValue();
    }

    protected Invocation.Builder request(String path) {
        Invocation.Builder request = getTarget().path(path).request();
        request.cookie("JSESSIONID",sessionId);

        return request;
    }

    protected Response post(String path, MultivaluedMap<String, String>
params) {
        return post(request(path), params);
    }

    protected       Response       post(Invocation.Builder       request,
MultivaluedMap<String, String> params) {
        return request.post(Entity.form(params));
    }

    protected Response get(String path) {
        return get(request(path));
    }

    protected Response get(Invocation.Builder request) {
        return request.get();
    }
}
```

4.3.2 MatchCommunicator.java

```
package com.dod.bot.communicators;

import javax.ws.rs.core.MultivaluedHashMap;
import javax.ws.rs.core.MultivaluedMap;
import java.util.UUID;

/**
 * Handles match requests to the server
```

```
 */
public class MatchCommunicator extends CommunicatorBase {
    public void joinMatch(UUID matchId) {
        MultivaluedMap<String,       String>       params       =       new
MultivaluedHashMap<String, String>();
        params.add("matchId", matchId.toString());

        post("match/join", params);
    }
}
```

4.3.3 MoveCommunicator.java

```
package com.dod.bot.communicators;

import javax.ws.rs.core.MultivaluedHashMap;
import javax.ws.rs.core.MultivaluedMap;

/**
 * Communicates move requests to the server
 */
public class MoveCommunicator extends CommunicatorBase {
    public void moveDirection(String direction) {
        MultivaluedMap<String,       String>       params       =       new
MultivaluedHashMap<String, String>();
        params.add("key", direction);

        post("game/move", params);
    }
}
```

4.3.4 stateCommunicator.java

```
package com.dod.bot.communicators;

import com.dod.service.model.GameStateModel;

/**
 * Communicates status requests to the server
 */
public class stateCommunicator extends CommunicatorBase {
    public GameStateModel getState() {
        return get("game/status").readEntity(GameStateModel.class);
    }
}
```

4.4 DungeonOfDooom-master\Sourcecode\project\src\bot\main\java\com\dod\bot

4.4.1 Bot.java

```
package com.dod.bot;

import com.dod.service.model.GameStateModel;
import com.dod.bot.communicators.MatchCommunicator;
import com.dod.bot.communicators.MoveCommunicator;
import com.dod.bot.communicators.stateCommunicator;
import com.dod.service.model.TileModel;

import java.util.List;
import java.util.Random;
import java.util.UUID;

/**
 * The bot
```

```
 */
public class Bot {
    private MatchCommunicator matchCommunicator;
    private MoveCommunicator moveCommunicator;
    private com.dod.bot.communicators.stateCommunicator stateCommunicator;

    private double delta;
    private double timestep = 200 * 1000000;
    private long previousTime;

    private boolean isPlaying = false;
    private GameStateModel state;

    private Random random;

    public Bot() {
        this.matchCommunicator = new MatchCommunicator();
        this.moveCommunicator = new MoveCommunicator();
        this.stateCommunicator = new stateCommunicator();
        random = new Random();
    }

    public void play(UUID matchId) {
        isPlaying = true;
        matchCommunicator.joinMatch(matchId);
        state = stateCommunicator.getState();
        delta = 0;
        previousTime = System.nanoTime();

        while(isPlaying) {
            long currentTime = System.nanoTime();
            delta += currentTime - previousTime;
            previousTime = currentTime;

            if(delta > timestep) {
                delta -= timestep;

                state = stateCommunicator.getState();
                if (random.nextBoolean()) {
                    moveCommunicator.moveDirection(random.nextBoolean()    ?
"A" : "D");
                } else {
                    moveCommunicator.moveDirection(random.nextBoolean()    ?
"W" : "S");
                }

                if (state.isHasEnded()) {
                    isPlaying = false;
                }
            }
        }
    }
}
```

4.4.2 Map.java

```
package com.dod.bot;

import com.dod.models.Point;
import com.dod.service.model.TileModel;

import java.util.ArrayList;
```

```
import java.util.List;

/**
 * Map for the bot modeled on the responses from the server
 */
public class Map {
    private ArrayList<List<TileModel>> map;

    public Map() {
        map = new ArrayList<List<TileModel>>();
    }

    public void addTile(TileModel[] tiles) {
        int xMax = 0;
        int yMax = 0;

        for(TileModel tile : tiles) {
            if(tile.getPosition().x > xMax) xMax = tile.getPosition().x;
            if(tile.getPosition().y > yMax) yMax = tile.getPosition().y;
        }

        for(int x = 0; x < xMax; x++) {
            List<TileModel> row = map.get(x);
            if(row == null) {
                row = new ArrayList<TileModel>();
                map.add(row);
            }

            for(int y =0; y < yMax; y++) {
                TileModel tile = null;

                for(TileModel tileInput : tiles) {
                    if(tileInput.getPosition().equals(new Point(x,y))) {
                        tile = tileInput;
                        break;
                    }
                }

                //row.set(y, tile);
            }
        }
    }
}
```

4.4.3 Main.java

```
package com.dod.bot;

import java.util.UUID;

/**
 * Gets command parameters and intitialises bot
 */
public class Main {

    /**
     * Start the bot
     * @param args expects 1 argument of ID for match to join
     */
    public static void main(String args[]) {
        UUID matchId = null;
        try {
```

```
            matchId = UUID.fromString(args[0]);
        }
        catch(Exception e) {
            e.printStackTrace();
            return;
        }

        Bot bot = new Bot();
        bot.play(matchId);
    }
}
```

4.5 DungeonOfDooom-master\Sourcecode\project\src\bot

4.5.1 pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>dungeon-of-doom</groupId>
    <artifactId>dungeon-of-doom-bot</artifactId>
    <version>1.0-SNAPSHOT</version>
    <dependencies>
        <dependency>
            <groupId>org.glassfish.jersey.media</groupId>
            <artifactId>jersey-media-moxy</artifactId>
            <version>2.24.1</version>
        </dependency>
        <dependency>
            <groupId>dungeon-of-doom</groupId>
            <artifactId>dungeon-of-doom-service</artifactId>
            <version>1.0</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <!-- Mark JAR as executable -->
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-jar-plugin</artifactId>
                <version>3.0.2</version>
                <configuration>
                    <archive>
                        <manifest>
                            <addClasspath>true</addClasspath>
                            <classpathPrefix>lib/</classpathPrefix>
                            <mainClass>com.dod.bot.Main</mainClass>
                        </manifest>
                    </archive>
                </configuration>
            </plugin>
        </plugins>
    </build>

</project>
```

4.6 DungeonOfDooom-master\Sourcecode\project\src\Client\assets

4.6.1 style.css

```
/* Dungon of Doom CSS stylesheet 2016 University of Bath */

ul {
  list-style-type: none;
  margin: 0;
  padding: 0;
  overflow: hidden;
  background-color: #333;
}

ul#logged-in-header {
  display: none;
}

ul#logged-in-header li {
  cursor: pointer;
}

li {
  float: left;
}

li a {
  display: block;
  color: #EEE;
  text-align: center;
  padding: 14px 16px;
  text-decoration: none;
}

li a:hover:not(.active) {
  background-color: #000000;
}

.join-link {
  color:#11D;
  cursor: pointer;
}

.active {
  background-color: #af222a;
}

body {
  background-image: url('header.jpg');
  background-color: #cccccc;
  background-size: cover;
  font-family: 'VT323', monospace;
  color: #EEE;
}

section {
  position: fixed;
  border: #333333;
  background-color: rgba(52, 7, 5, 0.55);
  margin-right: 7%;
  margin-left: 7%;
  margin-top: 3%;
  width: 87%;
  height: 80%;
```

```
  text-align: center;
  display:none; !important
}

.center {
  position :relative;
  margin-left: 30%;
  width: 45%;
  padding: 20px;
}

h1 {
  text-decoration: underline;
  color: #EEE;
  font-size: 40px;
}

#score-table {
  border-collapse: collapse;
  width: 100%;
}

#score-table td, #score-table th {
  border: 1px solid #ddd;
  font-size: 20px;
  text-align: center;
  padding: 8px;
  color: #EEE;
}

#score-table tr:hover {
  background-color: #333;
}

#score-table th {
  padding-top: 12px;
  padding-bottom: 12px;
  text-align: center;
  font-size: 30px;

  color: #EEE;
}

footer {
  background-color: rgba(0, 0, 0, 0.77);
  width: 100%;
  bottom: 0;
  position: fixed;
}

.container {
  margin-left: 30%;
  padding: 40px;
  position: relative;
  margin-top: 9%;
  width: 72%;
  font-size: large;
}

input[type=text], input[type=password] {
  width: 40%;
```

```
  padding: 12px 20px;
  margin: 8px 0;
  display: inline-block;
  border: 1px solid #ccc;
  box-sizing: border-box;
  font-size: medium;
}

input {
  color:#000;
}

button {
  background-color: rgba(0, 0, 0, 0.69);
  color: #EEE;
  padding: 14px 20px;
  margin: 10px;
  border: none;
  cursor: pointer;
  width: 40%;
  margin-left: 8%;
}

ul#guest-header {
  list-style-type: none;
  margin: 0;
  padding: 0;
  overflow: hidden;
  background-color: #333;
}

ul#guest-header li {
  text-align: center;
  color:white;
  font-size: 32px;
}

h3 {
  text-decoration: underline;
}

.validation {
  color:red;
  font-weight:bold;
}
```

4.7 DungeonOfDooom-master\Sourcecode\project\src\Client\scripts

4.7.1 main.js

```
/**
 * 2016 Dungeon of Dooom University of Bath.
 * "Part of the graphic tiles used in this program is the Public domain roguelike tileset "RLTiles".
 * Some of the tiles have been modified by our Team. You can find the original tileset at: http://rltiles.sf.net
 *  You can find Dungeon Crawl Stone Soup modified tilesets at: http://code.google.com/p/crawl-tiles/downloads/list"
 * Tileset was downloaded from opengameart.org/content/dungeon-crawl-32x32-tiles
 */

game = [];
```

```
game.menu = [];
game.auth = [];
game.constants = [];
game.func = [];
game.match = [];
game.var = [];
game.match.var = [];
game.camera = {};

game.var.init = function() {
    game.var.xSize = 900;
    game.var.ySize = 600;
    game.var.playerCharacter = {};
    game.var.scale = 50;
    game.var.tiles = [];
    game.var.characters = [];
    game.var.minCoins = {};
    game.var.winText = [];
    game.var.renderer = {};
    game.var.stage = {};
    game.var.graphics = {};
    game.var.playerTitles = [];
    game.var.isRunning = false;
    game.var.delta = 0;
    game.var.timeStep = 1000 / 20;
    game.var.lastFrameTimestamp = 0;
    game.var.opacityVis = 1.0;
    game.var.opacityInvis = 0.3;
};
game.var.init();
game.var.colours = [];
game.var.colours.background = 0x000000;
game.var.colours.wall = 0x8c8c8c;
game.var.colours.floor = 0xbf8040;
game.var.colours.gold = 0xffff66;
game.var.colours.player = 0xff2222;
game.var.colours.exit = 0x2222ff;
game.var.colours.shaded = [];
game.var.colours.shaded.wall = 0x565656;
game.var.colours.shaded.floor = 0x8c5010;
game.var.colours.shaded.gold = 0xcccc33;
game.var.colours.shaded.player = 0xcc0000;
game.var.colours.shaded.exit = 0x0000cc;

game.match.var.isLobbying = false;
game.match.var.isWaitingTostart = false;
game.match.var.delta = 0;
game.match.var.timeStep = 1000 / 5;
game.match.var.lastFrameTimestamp = 0;

game.constants.api = "http://localhost:8080/";
game.constants.loginFailed = "Oops, that didn't work. Make sure your
username/password are correct.";
game.constants.registrationFailed = "Oops, that didn't work. Fields cannot
be empty or more than 255 characters.";

/**
 * Generic method to make a GET request. Uses xhr fields to ensure cookies
are sent across domain.
 * @param url  {string}to send the request to
 * @param data {string}to send with the request
```

```
 * @param success function to execute on success
 * @param error function to execute on failure
 */
game.func.get = function(url, data, success, error) {
    $.ajax({
        type: "GET",
        url: url,
        data: data,
        success: success,
        error: error,
        xhrFields: {
            withCredentials: true
        }
    });
};

/**
 * Generic method to make a POST request. Uses xhr fields to ensure cookies
are sent across domain.
 * @param url  {string}to send the request to
 * @param data {string}to send with the request
 * @param success function to execute on success
 * @param error function to execute on failure
 */
game.func.post = function(url, data, success, error) {
    $.ajax({
        type: "POST",
        url: url,
        data: data,
        success: success,
        error: error,
        xhrFields: {
            withCredentials: true
        }
    });
};

/**
 * Constructs an url of an endpoint given the endpoint's controller and action
names.
 * @param controller {string}the controller to contact
 * @param action {string}the action to contact
 * @returns {string} the constructed path
 */
game.func.getApiPath = function(controller, action) {
    return game.constants.api + controller + "/" + action;
};

game.func.error = function( data, reason, exception ) {
    alert(' an error occurred :(');
    console.log(reason);
    console.log(exception);
};

game.auth.hook = function( data ) {
    game.menu.clearValidation();
    $('#guest-header').css('display', 'none');
    $('#logged-in-header').css('display','block');
    game.menu.openMatchLobby();
};
```

```
game.menu.loginFormValidation = function(message ) {
    $('#login-validation').html(message);
};

game.menu.clearValidation = function() {
    var validatorElements = $('.validation');
    validatorElements.html('');
    validatorElements.css('display', 'none');
};

game.auth.register = function() {
    var endpoint = game.func.getApiPath("player","register");
    var username = $("#username").val();
    var password = $("#password").val();

    game.func.post(endpoint,
        { "username" : username, "password" : password },
        game.auth.hook,
        function()
{ game.menu.loginFormValidation(game.constants.registrationFailed)});
};

game.auth.login = function() {
    var endpoint = game.func.getApiPath("player","login");
    var username = $("#username").val();
    var password = $("#password").val();

    game.func.post(endpoint,
        { "username" : username, "password" : password },
        game.auth.hook,
        function()
{ game.menu.loginFormValidation(game.constants.loginFailed)});
};

game.menu.openMatchLobby = function() {
    game.menu.allSections.css('display','none');
    game.menu.lobby.css('display','block');
    game.match.var.isLobbying = true;
    requestAnimationFrame(game.match.updateMatchList);
};

game.menu.openTutorial = function() {
    game.match.var.isLobbying = false;
    game.menu.allSections.css('display','none');
    game.menu.tutorial.css('display','block');
};

game.menu.openScoreboard = function() {
    game.match.var.isLobbying = false;
    var endpoint = game.func.getApiPath("score","top");
    game.func.get(endpoint,      {      },      game.menu.displayScoreboard,
game.func.error);
};

/**
 * Renders the scoreboard in the #score-table table.
 * @param scoreBoard the JSON object returned from a query to the score/top
endpoint
 */
game.menu.displayScoreboard = function( scoreBoard ) {
    $('#score-table tbody td').remove();
```

```
    $.each(scoreBoard.scores, function(i, score) {
        if(score != null) {
            $('#score-table
tbody').append($(String.format("<tr><td>{0}</td><td>{1}</td></tr>",
score.username, score.value)))
        }
    });

    game.menu.allSections.css('display','none');
    game.menu.scoreboard.css('display','block');
};

game.match.list = function() {
    //todo what if the webservice thinks you're already in a match?
    var endpoint = game.func.getApiPath("match","list");
    game.func.get(endpoint,         {},         game.menu.displayMatchList,
game.menu.error);
};

game.menu.displayMatchList = function( data ) {
    var matchList = $('#match-list');
    matchList.empty();

    $.each( data, function( i, match ) {
        var  entry  =  $(  String.format("<p><a  data-id='{2}'  class='join-
link'>Join</a>  {0}'s  game  with  {1}  players</p>",  match.playerNames[0],
match.playerNames.length, match.id) );
        matchList.append(entry);
    });

    $(".join-link").click(game.match.join);
};

game.match.join = function( data ) {
    var id = $(data.currentTarget).data("id");
    game.match.var.isLobbying = false;
    game.match.var.isWaitingTostart = true;

    var endpoint = game.func.getApiPath("match","join");
    game.func.post(endpoint, { "matchId" : id }, game.menu.displayMatchMenu,
game.menu.error);
    requestAnimationFrame(game.match.updateStatus);
};

/**
 * Starts a new Match by sending a request to the web service
 */
game.match.new = function() {
    var endpoint = game.func.getApiPath("match","new");
    game.match.var.isLobbying = false;
    game.match.var.isWaitingTostart = true;

    var level = game.menu.levelChooser.val();
    game.func.post(endpoint,      {      "level"      :      level      },
game.menu.displayMatchMenu);
    requestAnimationFrame(game.match.updateStatus);
};

/**
 * Starts the player's current Match by sending a request to the web service
 */
```

```
game.match.start = function() {
    game.match.var.isWaitingTostart = false;
    var endpoint = game.func.getApiPath("match","start");
    requestAnimationFrame(function()     {game.func.post(endpoint,     null,
game.menu.initGameScreen, game.func.error) });
};

/**
 * Initialises the game screen. Resets game variables and creates a new HTML5
canvas. Begins the game loop.
 */
game.menu.initGameScreen = function() {
    game.var.init();
    game.menu.gameContainer.empty();

    game.var.renderer       =       PIXI.autoDetectRenderer(game.var.xSize,
game.var.ySize);
    game.var.renderer.backgroundColor = game.var.colours.background;
    game.var.renderer.transparent = true;
    game.menu.gameContainer.append(game.var.renderer.view);

    game.var.stage = new PIXI.Container();
    // game.var.graphics = new PIXI.Graphics();
    // game.var.stage.addChild(game.var.graphics);

    game.menu.match.css('display', 'none');
    game.menu.game.css('display', 'block');

    game.var.isRunning = true;
    requestAnimationFrame(game.updateGame);
};

game.initTextWinCondition = function( character ) {
    var style = {
        fontFamily : 'Arial',
        fontSize : '18px',
        fontStyle : 'italic',
        fontWeight : 'bold',
        fill : '#F7EDCA',
        stroke : '#4a1850',
        strokeThickness : 5,
        dropShadow : true,
        dropShadowColor : '#000000',
        dropShadowAngle : Math.PI / 6,
        dropShadowDistance : 4
    };

    game.var.winText[character.playerName]   =   new   PIXI.Text('Collect   '+
game.var.minCoins   +'   coins   minimum   to   win!   You   collected   '   +
game.var.playerCharacter.noCoins + ' coins!', style);
}

game.initPlayerTitle = function( character ) {
    var style = {
        fontFamily : 'Arial',
        fontSize : '18px',
        fontStyle : 'italic',
        fontWeight : 'bold',
        fill : '#F7EDCA',
        stroke : '#4a1850',
        strokeThickness : 5,
```

```
        dropShadow : true,
        dropShadowColor : '#000000',
        dropShadowAngle : Math.PI / 6,
        dropShadowDistance : 4
    };

    game.var.playerTitles[character.playerName]            =            new
PIXI.Text(character.playerName, style);
};

/**
 * Renders the current game state to the canvas.
 */
game.render = function() {
    //game.var.graphics.clear();
    game.var.stage = new PIXI.Container();

    for(x = 0; x < game.var.tiles.length; x++) {
        var row = game.var.tiles[x];
        if(typeof row !== 'undefined') {
            for (y = 0; y < game.var.tiles[x].length; y++) {
                var tile = game.var.tiles[x][y];

                if(typeof tile !== 'undefined') {
                    var tilePositionX = (x * game.var.scale) - game.camera.x;
                    var tilePositionY = (y * game.var.scale) - game.camera.y;

                    if (tile.type == 0) {
                        var wall = PIXI.Sprite.fromImage('assets/wall.png');
                        wall.x = tilePositionX;
                        wall.y = tilePositionY;
                        wall.alpha = tile.visible ? game.var.opacityVis :
game.var.opacityInvis;
                        game.var.stage.addChild(wall);
                    }
                    else if (tile.type == 1) {
                        var                    floor                    =
PIXI.Sprite.fromImage('assets/floor.png');
                        floor.x = tilePositionX;
                        floor.y = tilePositionY;
                        floor.alpha = tile.visible ? game.var.opacityVis :
game.var.opacityInvis;
                        game.var.stage.addChild(floor);
                    }
                    else if (tile.type == 2) {
                        var coin = PIXI.Sprite.fromImage('assets/coin.png');
                        coin.x = tilePositionX;
                        coin.y = tilePositionY;
                        coin.alpha = tile.visible ? game.var.opacityVis :
game.var.opacityInvis;
                        game.var.stage.addChild(coin);
                    }
                    else if(tile.type == 3) {
                        var exit = PIXI.Sprite.fromImage('assets/exit.png');
                        exit.x = tilePositionX;
                        exit.y = tilePositionY;
                        exit.alpha = tile.visible ? game.var.opacityVis :
game.var.opacityInvis;
                        game.var.stage.addChild(exit);
                    }
```

```
                    if (tile.visible && tile.character !== null) {
                        var positionX = tilePositionX + game.var.scale / 2;
                        var positionY = tilePositionY + game.var.scale / 2;

                        var char = PIXI.Sprite.fromImage('assets/char.png');
                        char.x = positionX - game.var.scale / 2;
                        char.y = positionY - game.var.scale / 2;
                        game.var.stage.addChild(char);

                        var character = game.var.tiles[x][y].character;
                        var                  playerTitle                  =
game.var.playerTitles[character.playerName];
                        if(typeof playerTitle === 'undefined') {
                            game.initPlayerTitle(character);
                        }
                        else {
                            playerTitle.x = positionX - game.var.scale;
                            playerTitle.y = positionY - game.var.scale;
                        }

game.var.stage.addChild(game.var.playerTitles[character.playerName]);

game.initTextWinCondition(game.var.tiles[x][y].character);

game.var.stage.addChild(game.var.winText[character.playerName]);
                    }
                }
            }
        }
    }

    game.var.isRunning = true;
    game.var.renderer.render(game.var.stage);
};

game.setAllTilesNotVisible = function() {
    $.each(game.var.tiles, function(x, row) {
        if(typeof row != 'undefined') {
            $.each(row, function(y, tile) {
                if(typeof tile != 'undefined')
                tile.visible = false;
            });
        }
    });
};

/**
 * Updates the current game status based on the result from a status request.
 * Will end the gam eif the status response indicates that the game is over.
 * @param status
 */
game.updateStatus = function( status ) {
    game.var.characters = status.characters;
    game.var.playerCharacter = status.playerCharacter;
    game.var.minCoins = status.minNumOfCoins;

    game.camera.x = (game.var.playerCharacter.position.x * game.var.scale) -
(game.var.xSize / 2);
    game.camera.y = (game.var.playerCharacter.position.y * game.var.scale) -
(game.var.ySize / 2);
```

```
    game.setAllTilesNotVisible();
    $.each( status.tiles, function ( i, tile ) {
        tile.character = null;
        game.var.addTile(tile);
    });

    $.each( status.characters, function( i, character) {

game.var.tiles[character.position.x][character.position.y].character      =
character;
    });

    if(status.hasEnded) {
        game.var.isRunning = false;
        game.end();
    }
    else {
        game.render();
    }
};

/**
 * Adds a tile if it doesn't already exist in memory, or updates the tile if
it does.
 * Will expand the size of game.var.tiles if it isn't large enough.
 * @param tile
 */
game.var.addTile = function( tile ) {
    var pos = tile.position;

    if(typeof game.var.tiles[pos.x] === 'undefined') {
        game.var.tiles[pos.x] = [];
    }

    tile.visible = true;
    game.var.tiles[pos.x][pos.y] = tile;
};

game.fetchStatus = function() {
    var endpoint = game.func.getApiPath("game","status");
    game.func.get(endpoint, {}, game.updateStatus, game.func.error);
};

/**
 * Makes a game status request if game.var.timestep has passed since the last
request.
 * Loops while game.var.isRunning
 * @param timestamp
 */
game.updateGame = function( timestamp ) {
    if(game.var.lastFrameTimestamp == 0) {
        game.var.lastFrameTimestamp = timestamp + game.var.timeStep;
    }
    game.var.delta += timestamp - game.var.lastFrameTimestamp;
    game.var.lastFrameTimestamp = timestamp;

    if(game.var.delta > game.var.timeStep) {
        game.fetchStatus();
        game.var.delta -= game.var.timeStep;
    }
```

```
    if(game.var.isRunning) {
        requestAnimationFrame(game.updateGame);
    }
};

/**
 * Makes a match list request if game.match.var.timeStep has passed since the
last request
 * Loops while game.match.var.isLobbying
 * @param timestamp
 */
game.match.updateMatchList = function( timestamp ) {
    if(game.match.var.lastFrameTimestamp == 0) {
        game.match.var.lastFrameTimestamp        =        timestamp        +
game.match.var.timeStep;
    }
    game.match.var.delta += timestamp - game.match.var.lastFrameTimestamp;
    game.match.var.lastFrameTimestamp = timestamp;

    if(game.match.var.delta >= game.match.var.timeStep) {
        game.match.list();
        game.match.var.delta -= game.match.var.timeStep;
    }

    if(game.match.var.isLobbying) {
        requestAnimationFrame(game.match.updateMatchList);
    }
};

/**
 * Makes a Match Status request if game.match.var.timeStep has passed since
the last request.
 * Loops while game.match.var.isWaitingToStart
 * @param timestamp
 */
game.match.updateStatus = function( timestamp ) {
    if(game.match.var.lastFrameTimestamp == 0) {
        game.match.var.lastFrameTimestamp        =        timestamp        +
game.match.var.timeStep;
    }
    game.match.var.delta += timestamp - game.match.var.lastFrameTimestamp;
    game.match.var.lastFrameTimestamp = timestamp;

    if(game.match.var.delta >= game.match.var.timeStep) {
        game.match.fetchStatus();
        game.match.var.delta -= game.match.var.timeStep;
    }

    if(game.match.var.isWaitingTostart) {
        requestAnimationFrame(game.match.updateStatus);
    }
};

game.match.fetchStatus = function() {
    var endpoint = game.func.getApiPath("match","status");
    game.func.get(endpoint,         {},         game.menu.displayMatchMenu,
game.func.error);
};

/**
 * Generates the Match Status details on the Match screen.
```

```
 * @param data Match Status as a JSON object
 */
game.menu.displayMatchMenu = function( data ) {
    game.menu.lobby.css('display','none');
    game.menu.match.css('display','block');

    var matchDeatils = $("#match-details");
    matchDeatils.empty();

    matchDeatils.append($("<h2>Waiting to start.</h2>"));
    matchDeatils.append($(String.format("<p>To add a bot use the following
ID: {0}</p>", data.id)));
    matchDeatils.append($("<h3>Players:</h3>"));

    $.each( data.playerNames, function( i, name ) {
        var entry = $( String.format("<p>{0}</p>", name) );
        matchDeatils.append(entry);
    });

    if(data.state == 'Ingame') {
        game.match.var.isWaitingTostart = false;
        game.menu.initGameScreen();
    }
};

game.match.leave = function() {
    game.var.isRunning = false;

    var endpoint =  game.func.getApiPath("match","leave");
    requestAnimationFrame(function()    {game.func.post(endpoint,    {    },
game.menu.openMatchLobby, game.func.error)});
};

game.menu.move = function( key ) {
    var endpoint = game.func.getApiPath("game","move");
    game.var.status    =    game.func.post(endpoint,    {"key"    :    key},
game.updateStatus, game.func.error);
};

/**
 * Builds the end-game screen and switches to it
 * @param result a MatchResultModel object
 */
game.menu.showEndGameScreen = function( result ) {
    if(result.winner == game.var.playerCharacter.playerName) {
        $('#end-game-title').html("YOU WIN!")
    }
    else {
        $('#end-game-title').html("YOU LOOSE!")
    }
    $('#end-game-detail').html(String.format("{0} wins with {1} coins! Your
score is {2} ", result.winner, result.winnerCoins, result.score));
    game.menu.gameContainer.empty();
    game.menu.game.css('display','none');
    game.menu.end.css('display','block');
};

game.end = function() {
    game.var.isRunning = false;
    var endpoint = game.func.getApiPath("match","result");
    game.func.get(endpoint,      {      },      game.menu.showEndGameScreen,
```

```
game.func.error);
};

$( document ).ready(function() {
    game.menu.login = $('#login');
    game.menu.lobby = $('#lobby');
    game.menu.levelChooser = $('#level');
    game.menu.match = $('#match');
    game.menu.tutorial = $('#tutorial');
    game.menu.end = $('#end-game');
    game.menu.scoreboard = $('#score');
    game.menu.game = $('#game');
    game.menu.gameContainer = $('#game-container');
    game.menu.allSections = $('section');

    game.menu.login.css('display', 'block');

    $('#register-btn').click(game.auth.register);
    $('#login-btn').click(game.auth.login);
    $('#new-match-btn').click(game.match.new);
    $('#start-match-btn').click(game.match.start);
    $('#match-leave-btn').click(game.match.leave);
    $('#return-btn').click(game.match.leave);
    $('#lobby-link').click(game.menu.openMatchLobby);
    $('#tutorial-link').click(game.menu.openTutorial);
    $('#score-link').click(game.menu.openScoreboard);

    window.addEventListener('keydown', function(event) {
        if (game.var.isRunning) {
            switch (event.keyCode) {
                case 65:
                case 37: // Left
                    game.menu.move('A');
                    break;
                case 87:
                case 38: // Up
                    game.menu.move('W');
                    break;
                case 68:
                case 39: // Right
                    game.menu.move('D');
                    break;
                case 83:
                case 40: // Down
                    game.menu.move('S');
                    break;
            }
        }
    }, false);
});
```

4.8 DungeonOfDooom-master\Sourcecode\project\src\Client

4.8.1 index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Dungeon of Dooom</title>
    <link              href="https://fonts.googleapis.com/css?family=Ubuntu"
rel="stylesheet">
    <link                                                  rel="stylesheet"
```

```
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css
">
    <link href="assets/style.css" rel="stylesheet">
</head>
<body>
    <ul id="logged-in-header">
        <li><a id="lobby-link">Lobby</a></li>
        <li><a id="tutorial-link">How to play</a></li>
        <li><a id="score-link">Score Table</a></li>
        <li style="float:right"><a class="active">Dungeon of Doom</a></li>
    </ul>
    <ul id="guest-header">
        <li >Dungeon of Doom</li>
    </ul>

    <section id="login">
        <div class="col-md-5 col-md-offset-3">
            <div class="row">
                <label for="username"><b>Username</b></label>
                <input     type="text"     placeholder="Enter     Username"
id="username" name="username"  required>
            </div>
            <div class="row">
                <label for="password"><b>Password</b></label>
                <input    type="password"    placeholder="Enter    Password"
id="password" name="password" required>
            </div>
            <div class="row">
                <p id="login-validation" class="validation"></p>
            </div>
            <div class="row">
                <button    id="login-btn"    class="btn    btn-danger    btn-
lg">Login</button>
            </div>
            <div class="row">
                <button   id="register-btn"   class="btn   btn-danger   btn-
lg">Register</button>
            </div>
        </div>
    </section>

    <section id="lobby">
        <h1>Matches</h1>
        <p id="match-list"></p>
        <div class="row">
            <button  id="new-match-btn"  class="btn  btn-danger  btn-lg">New
Match</button>
            <label for="level">Level</label>
            <input  type="number"  id="level"  name="level"  min="1"  max="3"
value="1">
        </div>
    </section>

    <section id="match">
        <h1>Match</h1>
        <p id="match-details"></p>
        <button    id="start-match-btn"    class="btn    btn-danger    btn-
lg">Start</button>
    </section>

    <section id="game">
```

```
        <div id="game-container"></div>
        <div clas="row">
            <button   id="match-leave-btn"   class="btn   btn-danger   btn-
lg">Leave</button>
        </div>
    </section>

    <section id="end-game">
        <h1 id="end-game-title"></h1>
        <p id="end-game-detail"></p>
        <button     id="return-btn"     class="btn     btn-danger     btn-
lg">Return</button>
    </section>

    <section id="tutorial">
        <aside class="left">
            <h1> Instructions </h1><br>
            The dungeon of doom is an online multiplayer game, which starts
with registration of player.
            The game icludes the functionality to choose either single player
or multi-player as well as an option to view top score achivied by player.
            <br><br>
            The objective of the game is to collect the specified amount of
gold in the dungeon and get to the exit before other player.

        </aside>

        <div class="right">
            <img src="assets/layout.jpeg"/>
        </div>
    </section>

    <section id="score">
        <h1 id="homeHeading">Score Table</h1>
        <table id= "score-table">
        <thead>
            <tr>
              <th>Username</th>
              <th>Score</th>
            </tr>
        </thead>
        <tbody>

        </tbody>
        </table>
    </section>

    <footer>
        <p>Dungeon of Dooom coursework entry for University of Bath Software
Engineering unit</p>

    </footer>

    <script                                          type="text/javascript"
src="scripts/lib/stringformat.js"></script>
    <script type="text/javascript" src="scripts/lib/jquery.min.js"></script>
    <script type="text/javascript" src="scripts/lib/pixi.js"></script>
    <script type="text/javascript" src="scripts/main.js"></script>
</body>
</html>
```

4.9 DungeonOfDooom-master\Sourcecode\project\src\domain\com\dod\db\repositories

4.9.1 DatabaseRepository.java

```
package com.dod.db.repositories;

import com.dod.db.DatabaseConnection;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;

/**
 * <pre>
 * A base class of the Repository pattern
 * Introduces the generic getStatement() method to reuse that code across the
different repositories
 * </pre>
 */
public class DatabaseRepository<T> {
    /**
     * Make a SELECT query to fetch the unique object in question from the
database
     * @param object an instance of the object in question with the unique
field (but not necessarily others) filled out
     * @return An instance of the object
     * @throws SQLException if the statement fails or connection cannot be
established
     */
    public T get(T object) throws SQLException { return  null;  }

    /**
     * Make an INSERT query to insert the object in question into the database
     * @param object the object in question
     * @return true if successful, false otherwise
     * @throws SQLException
     */
    public boolean insert(T object) throws SQLException  {  return false;  }

    /**
     * Make a DELETE query to delete the object in question from the database
     * @param object the object in question with the unique field (but not
necessarily others) filled out
     * @return true if successful, false otherwise
     * @throws SQLException when the statement fails
     */
    public boolean delete(T object) throws SQLException {  return false;   }

    protected PreparedStatement ps;

    /**
     * Prepares a statement from a string using the database connection
     * @param text the text of the statement
     * @return a PreparedStatement instance
     * @throws SQLException when the statement fails
     */
    protected    PreparedStatement    getStatement(String    text)    throws
SQLException
    {
        Connection con = DatabaseConnection.getConnection();
        PreparedStatement ps = con.prepareStatement(text);
```

```
        return ps;
    }

}
```

4.9.2 IPlayerRepository.java

```
package com.dod.db.repositories;

import com.dod.models.Player;

import java.sql.SQLException;

/**
 * <pre>
 *     Follows the Repository pattern.
 *      Intended for selecting/inserting/deleting "Player" entries from the
database.
 * </pre>
 */
public interface IPlayerRepository {
    /**
     * Make a SELECT query to fetch the unique Player in question from the
database
     * @param object an instance of the Player in question with the unique
field (but not necessarily others) filled out
     * @return Player object fetched from the database
     * @throws SQLException if the statement fails or connection cannot be
established
     */
    Player get(Player object) throws SQLException;
    /**
     * Make an INSERT query to insert the Player in question into the database
     * @param object the Player in question
     * @return true if successful, false otherwise
     * @throws SQLException when the statement fails
     */
    boolean delete(Player object) throws SQLException;
    /**
     * Make a DELETE query to delete the Player in question from the database
     * @param object the Player in question with the unique field (but not
necessarily others) filled out
     * @return true if successful, false otherwise
     * @throws SQLException when the statement fails
     */
    boolean insert(Player object) throws SQLException;
}
```

4.9.3 IScoreRepository.java

```
package com.dod.db.repositories;

import com.dod.models.Player;
import com.dod.models.Score;

import java.sql.SQLException;

/**
 * <pre>
 *     Follows the Repository pattern.
 *      Intended for selecting/inserting/deleting "Score" entries from the
database.
```

```
 * </pre>
 */
public interface IScoreRepository {
    /**
     * Make a SELECT query to fetch the unique Score in question from the
database
     * @param object an instance of the Score in question with the unique
field (but not necessarily others) filled out
     * @return Score fetched from the database
     * @throws SQLException if the statement fails or connection cannot be
established
     */
    Score get(Score object) throws SQLException;
    /**
     * Make an INSERT query to insert the Score in question into the database
     * @param object the Score in question
     * @return true if successful, false otherwise
     * @throws SQLException
     */
    boolean insert(Score object) throws SQLException;
    /**
     * Make a DELETE query to delete the Score in question from the database
     * @param object the Score in question with the unique field (but not
necessarily others) filled out
     * @return true if successful, false otherwise
     * @throws SQLException
     */
    boolean delete(Score object) throws SQLException;

    /**
     * Get the 10 highest scores from database
     * @return Score[] array of 10 Score objects
     * @throws SQLException when the statement fails
     */
    Score[] getHighestScores() throws SQLException;

    /**
     * Get the 10 highest scores of the player
     * @param object Player object
     * @return Score[] array of 10 Score objects
     * @throws SQLException when the statement fails
     */
    Score[] getPlayerScores(Player object) throws SQLException;
}
```

4.9.4 PlayerRepository.java

```
package com.dod.db.repositories;

import com.dod.models.Player;

import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

/**
 * <pre>
 *     Implements IPlayerRepository.
 *     Follows the Repository pattern.
 *      Intended for selecting/inserting/deleting "Player" entries from the
database.
 * </pre>
```

```
 */
public class PlayerRepository extends DatabaseRepository<Player> implements
IPlayerRepository {

    private final String deleteQuery = "DELETE FROM player WHERE username
= ?";
    private final String getQuery = "SELECT username, password, salt FROM
player WHERE username = ?";
    private final String insertQuery = "INSERT INTO player (username,
password, level, salt) VALUES (?, ?, 0, ?)";

    /**
     * Make a SELECT query to fetch the unique Player in question from the
database
     * @param object an instance of the Player in question with the unique
field (but not necessarily others) filled out
     * @return Player object fetched from the database
     * @throws SQLException if the statement fails or connection cannot be
established
     */
    @Override
    public Player get(Player object) throws SQLException {
        PreparedStatement statement = getStatement(getQuery);

        statement.setString(1, object.getUsername());
        ResultSet rs = statement.executeQuery();
        if (rs.next())
            return           new           Player(rs.getString("username"),
rs.getString("password"), rs.getBytes("salt"));
        else
            return null;
    }

    /**
     * Make an INSERT query to insert the Player in question into the database
     * @param object the Player in question
     * @return true if successful, false otherwise
     * @throws SQLException when the statement fails
     */
    @Override
    public boolean delete(Player object) throws SQLException {
        PreparedStatement statement = getStatement(deleteQuery);

        statement.setString(1, object.getUsername());
        if (statement.executeUpdate() == 0) {
            return false;
        } else {
            return true;
        }

    }

    /**
     * Make a DELETE query to delete the Player in question from the database
     * @param object the Player in question with the unique field (but not
necessarily others) filled out
     * @return true if successful, false otherwise
     * @throws SQLException when the statement fails
     */
    @Override
    public boolean insert(Player object) throws SQLException{
```

```
        PreparedStatement statement = getStatement(insertQuery);

        statement.setString(1, object.getUsername());
        statement.setString(2, object.getHashedPassword());
        statement.setBytes(3, object.getSalt());
        try {
            statement.executeUpdate();
        }
        catch (SQLException e) {
            return false;
        }
        statement.close();
        return true;
    }
}
```

4.9.5 ScoreRepository.java

```
package com.dod.db.repositories;

import com.dod.models.Player;
import com.dod.models.Score;

import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

/**
 * <pre>
 *     Implements IPlayerRepository.
 *     Follows the Repository pattern.
 *      Intended for selecting/inserting/deleting "Score" entries from the
database.
 * </pre>
 */
public class ScoreRepository extends DatabaseRepository<Score> implements
IScoreRepository {

    private final String getPlayerQuery = "SELECT * FROM score WHERE
username='?' ORDER BY value DESC LIMIT 10";
    private final String deleteQuery = "DELETE FROM score WHERE id = ?";
    private final String getScoreQuery = "SELECT * FROM score WHERE id = ?";
    private final String getQuery = "SELECT * FROM score ORDER BY value DESC
LIMIT 10";
    private final String insertQuery = "INSERT INTO score (username, value)
VALUES (?, ?)";

    /**
     * Inserts a score value to score table of database based on player's
     * username.
     * @param scoreObject current score that we need to score
     * @return true if insertion was successful else false
     * @throws SQLException when the statement fails
     */
    @Override
    public boolean insert(Score scoreObject) throws SQLException {

        PreparedStatement statement = getStatement(insertQuery);

        statement.setString(1, scoreObject.getUsername());
        statement.setInt(2, scoreObject.getValue());
```

```
        try {
            statement.executeUpdate();
        } catch (SQLException e) {
            return false;
        }

        statement.close();
        return true;
    }

    /**
     * Delete a score row from database
     * !! We should not use that.
     * @param object score object to delete
     * @return true if the deletion was successful else false
     * @throws SQLException when the statement fails
     */
    @Override
    public boolean delete(Score object) throws SQLException {
        PreparedStatement statement = getStatement(deleteQuery);

        statement.setInt(1, object.getId());
        if (statement.executeUpdate() == 0) {
            return false;
        } else {
            return true;
        }
    }

    /**
     * Get the 10 highest scores from database
     * @return Score[] array of 10 Score objects
     * @throws SQLException when the statement fails
     */
    public Score[] getHighestScores() throws SQLException {
        PreparedStatement statement = getStatement(getQuery);

        //statement.setString(1, object.getUsername());
        ResultSet rs = statement.executeQuery();
        Score[] result = new Score[10];
        int i = 0;
        while (rs.next()) {
            Score       temp       =       new       Score(rs.getInt("id"),
rs.getString("username"), rs.getInt("value"));
            result[i] = temp;
            i++;
        }
        return result;
    }

    /**
     * Get the 10 highest scores of the player
     * @param object Player object
     * @return Score[] array of 10 Score objects
     * @throws SQLException when the statement fails
     */
    public Score[] getPlayerScores(Player object) throws SQLException {
        PreparedStatement statement = getStatement(getPlayerQuery);

        statement.setString(1, object.getUsername());
```

```
        ResultSet rs = statement.executeQuery();
        Score[] result = new Score[10];
        int i = 0;
        while (rs.next()) {
            Score       temp       =       new       Score(rs.getInt("id"),
rs.getString("username"), rs.getInt("value"));
            result[i] = temp;
            i++;
        }
        return result;
    }

    /**
     * returns a Score based on id from the database
     * @param Score to be fetched must have unique identifier populated
     * @return Score object
     * @throws SQLException when the statement fails
     */
    @Override
    public Score get(Score object) throws SQLException {
        PreparedStatement statement = getStatement(getScoreQuery);

        statement.setInt(1, object.getId());
        ResultSet rs = statement.executeQuery();
        if (rs.next()) {
            return new Score(rs.getInt(1), rs.getString(2), rs.getInt(3));
        } else {
            return null;
        }
    }
}
```

4.10 DungeonOfDooom-master\Sourcecode\project\src\domain\com\dod\db

4.10.1 DatabaseConnection.java

```
package com.dod.db;

import com.mysql.jdbc.jdbc2.optional.MysqlDataSource;

import java.sql.Connection;
import java.sql.SQLException;

/**
 * Stores a connection to the database using the singleton pattern
 */
public class DatabaseConnection {

    private static Connection connection;

    /**
     * A static connection to ensure that all sessions use the same MySql
connection
     * Could be done more intelligently with connection pooling
     * @return Connection instance
     *  @throws  SQLException  when  the  database  connection  cannot  be
established
     */
    public static Connection getConnection() throws SQLException {
        if(connection != null) {
            return connection;
        }
        else {
```

```
            MysqlDataSource dataSource = new MysqlDataSource();
            dataSource.setUser("dungeonofdoom");
            dataSource.setPassword("Delicate.Sunshine.Twist.Myth32");
            dataSource.setServerName("localhost");
            dataSource.setDatabaseName("dungeonofdoom");

            connection = dataSource.getConnection();

            return connection;
        }
    }

    /**
     * Closes the connection
     */
    public static void Close() {
        try {
            connection.close();
        }
        catch(SQLException e) {
            System.console().printf(e.getMessage());
        }
    }
}
```

4.11 DungeonOfDooom-master\Sourcecode\project\src\domain\com\dod\game

4.11.1 IMatchList.java

```
package com.dod.game;

import com.dod.models.Match;

import java.util.List;
import java.util.UUID;

/**
 * Stores ongoing matches in memory and provides functions to access these
matches.
 */
public interface IMatchList {

    /**
     * Returns a singleton instance of MatchList, creating it if it hasn't
been initialised yet.
     * @return MatchList
     */
    void addMatch(Match match);

    /**
     * Gets all matches that are in the Lobbying state
     * @return List of Match objects
     */
    List<Match> getLobbyingMatches();

    /**
     * Gets a Match by a particular ID. Returns null if the match is missing.
     * @param id the UUID that corresponds to the match to be fetched
     * @return Match
     */
    Match getMatch(UUID id);

    /**
```

```
     * Gets a match by player name. Each player should only have one match.
Returns null if player has no match.
     * @param username the username of the player
     * @return Match
     */
    Match getMatchForPlayer(String username);

    /**
     * Returns true if the player has a match in the list
     * @param username the player's username
     * @return true if the player has a match in the list otherwise false
     */
    boolean playerHasMatch(String username);

    /**
     * Removes the match fitting the specified ID from the list
     * @param id the UUID that corresponds to the particular Match to be
removed
     */
    void removeMatch(UUID id);
}
```

4.11.2 MatchList.java

```
package com.dod.game;

import com.dod.models.Match;
import com.dod.models.MatchState;

import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import java.util.UUID;

/**
 * <pre>
 * Implementation of IMatchList
 * Stores ongoing matches in memory and provides functions to access these
matches.
 * Uses a singleton so that we can fetch the same object between requests
 *      (And because this is much easier to test than making all methods
static)
 * </pre>
 */
public class MatchList implements IMatchList {

    private static IMatchList instance;

    /**
     * Returns a singleton instance of MatchList, creating it if it hasn't
been initialised yet.
     * @return MatchList
     */
    public static IMatchList instance() {
        if(instance == null) {
            instance = new MatchList();
        }
        return instance;
    }

    private List<Match> ongoingMatches = new ArrayList();
```

```
    /**
     * Adds a match to the list
     * @param match the match to add
     */
    public void addMatch(Match match) {
        ongoingMatches.add(match);
    }

    /**
     * Gets all matches that are in the Lobbying state
     * @return List of Match objects
     */
    public List<Match> getLobbyingMatches() {
        List<Match> result = new ArrayList();

        for(Match match : ongoingMatches) {
            if(match.getState() == MatchState.Lobbying) {
                result.add(match);
            }
        }

        return result;
    }

    /**
     * Gets a Match by a particular ID. Returns null if the match is missing.
     * @param id the UUID that corresponds to the match to be fetched
     * @return Match
     */
    public Match getMatch(UUID id) {
        Match result = null;

        for(Match match : ongoingMatches) {
            if(match.getId().equals(id)) {
                result = match;
                break;
            }
        }

        return result;
    }

    /**
     * Gets a match by player name. Each player should only have one match.
Returns null if player has no match.
     * @param username the username of the player
     * @return Match
     */
    public Match getMatchForPlayer(String username) {
        Match result = null;

        for(Match match: ongoingMatches) {
            if(match.hasCharacter(username)) {
                result = match;
                break;
            }
        }

        return result;
    }
```

```
    /**
     * Returns true if the player has a match in the list
     * @param username the player's username
     * @return true if the player has a match in the list otherwise false
     */
    public boolean playerHasMatch(String username) {
        return getMatchForPlayer(username) != null;
    }

    /**
     * Removes the match fitting the specified ID from the list
     * @param id the UUID that corresponds to the particular Match to be
removed
     */
    public void removeMatch(UUID id) {
        for(Match match: ongoingMatches) {
            if(match.getId().equals(id)) {
                ongoingMatches.remove(match);
                break;
            }
        }
    }
}
```

4.12 DungeonOfDooom-master\Sourcecode\project\src\domain\com\dod\models

4.12.1 Character.java

```
package com.dod.models;

import java.util.ArrayList;
import java.util.List;

/**
 * <pre>
 *     A Character is a fictional entity that moves around the game world.
 *     A Character belongs to a Player.
 *     A Character has a position and can interact with coins and the exit.
 * </pre>
 */
public class Character {

    private Point position;
    private Player player;
    private int collectedCoins;
    private List<Point> collectedCoinsPos;

    public Character(Point position, Player player) {
        this.position = position;
        this.player = player;
        this.collectedCoinsPos = new ArrayList<>();
        collectedCoins = 0;
    }

    /**
     * The player's position in the game world
     * @return Point
     */
    public Point getPosition() {
        return position;
    }

    /**
```

```
     * The player's position in the game world
     * @param position Point
     */
    public void setPosition(Point position) {
        this.position = position;
    }

    /**
     * The Player that this Character belongs to
     * @return Player
     */
    public Player getPlayer() {
        return player;
    }

    /**
     * The Player that this Character belongs to
     * @param player Player
     */
    public void setPlayer(Player player) {
        this.player = player;
    }

    public int getCollectedCoins() {
        return collectedCoins;
    }

    public void setCollectedCoins(int collectedCoins) {
        this.collectedCoins = collectedCoins;
    }

    /**
     * Keeps track of which coins on the map this Character has collected.
     * This enables us to leave the coin on the Map once it has been picked
up, thereby allowing other players
     * to pick it up, and yet not send the same coin to the same player's
client again.
     * @return a list of Point objects that represent the points on the map
where the Character has collected a coin
     */
    public List<Point> getCollectedCoinsPos() {
        return collectedCoinsPos;
    }

    /**
     * Keeps track of which coins on the map this Character has collected.
     * This enables us to leave the coin on the Map once it has been picked
up, thereby allowing other players
     * to pick it up, and yet not send the same coin to the same player's
client again.
     * @param newPoint the Point to add to the collection
     */
    public void addCollectedCoinsPos(Point newPoint) {
        this.collectedCoinsPos.add(newPoint);
    }
}
```

4.12.2 Map.java

```
package com.dod.models;

import java.io.Serializable;
```

```
import java.util.Random;

/**
 * <pre>
        A Map stores a 2-dimensional grid of Tiles.
        A Map has a name, width, height and numbe rof coins total and required
to win.
 * </pre>
 */
public class Map {

      protected int width;
      protected int height;
      protected String name;
      protected int totalNumberOfCoins;
      protected int numberOfCoinsNeededToWin;
      protected Tile[][] tiles;

      public Map(int width, int height) {
            tiles = new Tile[width][height];
      }

      public     Map(String     name,     int     totalNumberOfCoins,     int
numberOfCoinsNeededToWin, int width, int height, Point mapSize) {
            this.name = name;
            this.totalNumberOfCoins = totalNumberOfCoins;
            this.numberOfCoinsNeededToWin = numberOfCoinsNeededToWin;
            this.width = width;
            this.height = height;
            tiles = new Tile[mapSize.x][mapSize.y];
      }

      public void setTile(Point position, Tile tile) {
            tiles[position.x][position.y] = tile;
      }

      public String getName(){
            return name;
      }

      public void setName(String name){
            this.name = name;
      }

    /**
     * The total number of coins that should be created in the map.
     * @return int
     */
      public int getCoinNo(){
            return totalNumberOfCoins;
      }

    /**
     * The total number of coins that should be created in the map.
     * @param coin_no int
     */
      public void setCoinNo(int coin_no){
            this.totalNumberOfCoins = coin_no;
      }

    /**
```

```
     * The total number of coins needed to win on this map
     * @return int
     */
      public int getCoinWin(){
            return numberOfCoinsNeededToWin;
      }

    /**
     * The total number of coins needed to win on this map
     * @param coin_win int
     */
      public void setCoinWin(int coin_win){
            this.numberOfCoinsNeededToWin = coin_win;
      }

      public Tile getTile(Point point) {
            return tiles[point.x][point.y];
      }

      public int getWidth() {
            return width;
      }

      public int getHeight() {
            return height;
      }

    /**
     * Gets a random position of a tile that is not a wall, coin or exit.
     * @return Point
     */
      public Point getRandomFreeTilePoint() {
            Random random = new Random();
            Point point = null;

            while(point == null) {
                  int x = random.nextInt(width-1);
                  int y = random.nextInt(height-1);

                  if(tiles[x][y].getType() == TileType.Empty.getValue()) {
                        point = new Point(x,y);
                  }
            }

            return point;
      }
}
```

4.12.3 Match.java

```
package com.dod.models;

import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import java.util.UUID;

/**
 * Represents a match
 */
public class Match {
```

```
    private UUID id;
    private Map map;
    private List<Character> characters;
    private MatchState state;
    private long timer;
    private int score;

    public Match(Map map) {
        this.id = UUID.randomUUID();
        this.map = map;
        this.characters = new ArrayList();
        state = MatchState.Lobbying;
        timer = 0;
        score = 0;
    }

    public Map getMap() {
        return map;
    }

    public void addCharacter(Player player, Point position) {
        characters.add(new Character(position, player));
    }

    public void removeCharacter(Player player) {
        for(Character character : characters) {

if(character.getPlayer().getUsername().equals(player.getUsername())) {
                characters.remove(character);
                break;
            }
        }
    }

    public Character getCharacter(String username) {
        Character result = null;

        for(Character character : characters) {
            if(character.getPlayer().getUsername().equals(username)) {
                result = character;
                break;
            }
        }

        return result;
    }

    public String[] getPlayerNames() {
        String[] names = new String[characters.size()];

        for(int i = 0; i < characters.size(); i++) {
            names[i] = characters.get(i).getPlayer().getUsername();
        }

        return names;
    }

    public boolean hasCharacter(String userName) {
        return getCharacter(userName) != null;
    }
```

```
    public UUID getId() {
        return id;
    }

    public void startGame() {
        state = MatchState.Ingame;
    }

    public MatchState getState() {
        return state;
    }

    public void setState(MatchState state) {
        this.state = state;
    }

    public List<Character> getCharactersOnTile(Point point) {
        List<Character> charactersOnTile = new ArrayList();

        for(Character character :characters) {
            if(character.getPosition().equals(point)) {
                charactersOnTile.add(character);
            }
        }

        return charactersOnTile;
    }

    public Character getCharacterWithHighestCoins() {
        Character character = null;

        for(Character c : characters) {
            if(character    ==    null    ||    c.getCollectedCoins()    >
character.getCollectedCoins()) {
                character = c;
            }
        }

        return character;
    }

    public long getTimer() {
        return timer;
    }

    public void setTimer(long timer) {
        this.timer = timer;
    }

    public int getScore() { return this.score; }

    public void setScore(int score) { this.score = score; }
}
```

4.12.4 MatchState.java

```
package com.dod.models;

/**
 * The state of a Match.
 */
public enum MatchState {
```

```
    Lobbying,
    Ingame,
    Over
}
```

4.12.5 Player.java

```
package com.dod.models;

/**
 * <pre>
 *     A Player represents the user that is in control of the game client
 *     A Player can sign in with a username or password
 *     A Player has a level and a password salt
 *     A Player's password is always hashed
 * </pre>
 */
public class Player {

    private String username;
    private String hashedPassword;
    private int level;
    private byte[] salt;

    public Player(String name) {
        this.username = name;
    }

    public Player(String name, String hashedPassword, byte[] salt) {
        this.username = name;
        this.hashedPassword = hashedPassword;
        this.salt = salt;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String value) {
        username = value;
    }

    public String getHashedPassword() {
        return hashedPassword;
    }

    public void setHashedPassword(String hashedPassword) {
        this.hashedPassword = hashedPassword;
    }

    public int getLevel() {
        return level;
    }

    public void setLevel(int level) {
        this.level = level;
    }

    public byte[] getSalt() {
        return salt;
    }
```

```
    public void setSalt(byte[] salt) {
        this.salt = salt;
    }
}
```

4.12.6 Point.java

```
package com.dod.models;

import javax.xml.bind.annotation.XmlRootElement;

/**
 * Bean class for storing a point (or vertex) in the map.
 */
@XmlRootElement
public class Point {
    public int x;
    public int y;

    public Point() {}

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    @Override
    public boolean equals(Object obj) {
        boolean result = false;

        if (obj instanceof Point) {
            Point point = (Point) obj;

            if (point.x == x && point.y == y) {
                result = true;
            }
        }

        return result;
    }
}
```

4.12.7 Score.java

```
package com.dod.models;

import javax.xml.bind.annotation.XmlRootElement;

/**
 * <pre>
 *      A Score stores the points a Player achieved when they completed a
Match.
 *      A Score as an ID in order to store the Score as a unique databaes
record
 *     A Score also has a value and the username of the player that the score
is related to.
 * </pre>
 */
@XmlRootElement
public class Score {
    private int id;
    private String username;
    private int value;
```

```
    public Score(int id, String username, int value) {
        this.id = id;
        this.username = username;
        this.value = value;
    }

    public Score(String username, int value) {
        this.id = -1;
        this.username = username;
        this.value = value;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public int getValue() {
        return value;
    }

    public void setValue(int value) {
        this.value = value;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }
}
```

4.12.8 Tile.java

```
package com.dod.models;

import java.io.Serializable;

/**
 * <pre>
 *     A Tile represents single tile on the grid that is the Map
 *     A Tile has a Type that indicates whether it is eg a wall, floor, coin
or exit tile.
 *     A Tile may or may not be visible
 * </pre>
 */
public class Tile {

      protected int type;
      protected boolean visibility;

      public Tile(int type, boolean visibility){
            this.setType(type);
            this.setVisibility(visibility);
      }
```

```
	public Tile(int type) {
		this.type = type;
	}

	public int getType() {
		return type;
	}
	public void setType(int type) {
		this.type = type;
	}
	public boolean isVisible() {
		return visibility;
	}
	public void setVisibility(boolean visibility) {
		this.visibility = visibility;
	}
	public String toString(){
		return "Type: "+this.type+"\nVisibility: "+this.visibility;
	}

}
```

4.12.9 TileType.java

```
package com.dod.models;

/**
 * The type of a tile, i.e is this tile a wall, floor or something else.
 */
public enum TileType {
    Wall(0),
    Empty(1),
    Coin(2),
    Exit(3);

    private final int value;
    TileType(int value) {
        this.value = value;
    }

    public int getValue() {
        return value;
    }
}
```

4.13 DungeonOfDooom-master\Sourcecode\project\src\service\src\main\java\com\dod\service\constant

4.13.1 Assets.java

```
package com.dod.service.constant;

/**
 * A set of static constant strings that define the paths to our assets.
 * Must always start with a slash.
 */
public class Assets {
    public static final String MapLevelOne = "/maps/level1.json";
    public static final String MapLevelFormat = "/maps/level%s.json";
}
```

4.14 DungeonOfDooom-master\Sourcecode\project\src\service\src\main\java\com\dod\service\controller

4.14.1 GameController.java

```
package com.dod.service.controller;

import com.dod.game.MatchList;
import com.dod.models.Map;
import com.dod.models.Player;
import com.dod.service.model.GameStateModel;
import com.dod.service.service.MovementService;
import com.dod.service.service.StateService;
import com.dod.service.service.VisibilityService;
import org.glassfish.grizzly.http.server.Request;

import javax.validation.constraints.NotNull;
import javax.ws.rs.*;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
import java.sql.SQLException;

/**
 * A controller to manage in-game game-related functionality ie getting the
current state of the world or moving.
 */
@Path("game")
public class GameController {

    @Context
    private Request request;
    StateService stateService;
    MovementService movementService;

    public GameController() {
        stateService    =    new    StateService(new    VisibilityService(),
MatchList.instance());
        movementService = new MovementService();
    }

    /**
     * Responds with the current gamestate from the Player's Character's
perspective, i.e. only returning visible tiles
     * If Player has no current ongoing Match returns 500 error.
     * @return Response 200 OK with GameStateModel as a JSON object
     */
    @GET
    @Produces(MediaType.APPLICATION_JSON)
    @Path("status")
    public Response status() {
        String                          username                          =
(String)request.getSession().getAttribute("player");
        GameStateModel state = stateService.GetState(new Player(username));

        return Response
                .ok()
                .entity(state)
                .build();
    }

    /**
     * An endpoint to request the Player's Character move once in a particular
direction.
     * Responds with game status after move.
```

```
     * If Player has no current ongoing Match returns 500 error.
     * @param direction a char from {W,S,A,D} pertaining to a particular
direction in the WASD layout, must not be null
     * @return Response 200 OK with GameStateModel as a JSON object
     */
    @POST
    @Produces(MediaType.APPLICATION_JSON)
    @Path("move")
    public Response move(@NotNull @FormParam("key") String direction) {
        String                          username                          =
(String)request.getSession().getAttribute("player");
        try {
            movementService.Move(direction, new Player(username));
        }
        catch(SQLException e) {
            e.printStackTrace();
            return Response.serverError().build();
        }
        GameStateModel state = stateService.GetState(new Player(username));

        return Response
                .ok()
                .entity(state)
                .build();
    }
}
```

4.14.2 MatchController.java

```
package com.dod.service.controller;

import com.dod.db.repositories.PlayerRepository;
import com.dod.game.MatchList;
import com.dod.models.Match;
import com.dod.models.Player;
import com.dod.service.model.MatchStatus;
import com.dod.service.service.IOService;
import com.dod.service.service.MatchService;
import com.dod.service.service.ParseService;
import org.glassfish.grizzly.http.server.Request;

import javax.validation.constraints.NotNull;
import javax.ws.rs.*;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
import java.sql.SQLException;
import java.util.UUID;

/**
 * A controller to manage Matches- joining, listing, starting a new one etc.
 */
@Path("match")
public class MatchController {

    @Context
    private Request request;

    private MatchService matchService;


    public MatchController() {
```

```
        this.matchService = new MatchService(
                new IOService(),
                new ParseService(),
                new PlayerRepository(),
                MatchList.instance());
    }

    /**
     * Responds with the status of the player's current Match.
     * If Player has no current Match returns a 500 error.
     * @return Response 200 OK with MatchStatus encoded in JSON
     */
    @GET
    @Produces(MediaType.APPLICATION_JSON)
    @Path("status")
    public Response status() {
        String                              username                            =
(String)request.getSession().getAttribute("player");
        return Response
            .ok()
            .entity(matchService.getStatus(new Player(username)))
            .build();
    }

    /**
     * Starts a new Match in a particular level and responds with that Match's
status
     * @param level int the level to load for this Match, must not be null
     * @return Response 200 OK with MatchStatus encoded in JSON or null if a
Match cannot be crated
     */
    @POST
    @Produces(MediaType.APPLICATION_JSON)
    @Path("new")
    public Response newMatch(
            @NotNull @FormParam("level") int level
    ) {
        String                              userName                            =
(String)request.getSession().getAttribute("player");

        MatchStatus newMatch = matchService.createMatch(userName, level);

        if(newMatch != null) {
            return Response
                    .ok()
                    .entity(newMatch)
                    .build();
        }
        else {
            return Response.serverError().build();
        }
    }

    /**
     * Changes a Match's status to Ingame (marking the start of the Match
for all players)
     * @return MatchStatus encoded in JSON
     */
    @POST
    @Produces(MediaType.TEXT_PLAIN)
    @Path("start")
```

```
    public Response start() {
        String                              username                              =
(String)request.getSession().getAttribute("player");

        matchService.startMatch(new Player(username));

        return Response
                .ok()
                .build();
    }

    /**
     * Lists all currently lobbying matches in a JSON array
     * @return Response 200 OK JSON array with encoded MatchStatus for each
lobbying Match
     */
    @GET
    @Produces(MediaType.APPLICATION_JSON)
    @Path("list")
    public Response list() {
        MatchStatus[] matches = matchService.getLobbyingMatches();

        return Response
                .ok()
                .entity(matches)
                .build();
    }

    /**
     * Joins the Player in an ongoing Match
     * @param matchId the UUID ID of the Match, must not be null
     * @return Response 200 OK with the latest MatchStatus encoded in JSON
     */
    @POST
    @Produces(MediaType.APPLICATION_JSON)
    @Path("join")
    public Response join(
            @NotNull @FormParam("matchId") UUID matchId
    ) {
        String                              username                              =
(String)request.getSession(false).getAttribute("player");
        try {
            matchService.joinMatch(new Player(username), matchId);
            return Response
                    .ok()
                    .entity(matchService.getStatus(new Player(username)))
                    .build();
        }
        catch(SQLException e) {
            e.printStackTrace();
            return Response
                    .serverError()
                    .build();
        }
    }

    /**
     * Removes the Player from their current Match
     * @return Response 200 OK with a blank body
     */
    @POST
```

```
    @Produces(MediaType.TEXT_PLAIN)
    @Path("leave")
    public Response leave() {
        String                              username                                =
(String)request.getSession(false).getAttribute("player");
        matchService.leaveMatch(new Player(username));

        return Response
                .ok()
                .build();
    }

    /**
     * Fetches the result of a Match from memory
     * @return Resepons 200 OK with JSON encoded MatchResultModel
     */
    @GET
    @Produces(MediaType.APPLICATION_JSON)
    @Path("result")
    public Response result() {
        String                              username                                =
(String)request.getSession(false).getAttribute("player");

        return Response
                .ok()
                .entity(matchService.getMatchResult(new Player(username)))
                .build();
    }
}
```

4.14.3 PlayerController.java

```
package com.dod.service.controller;

import com.dod.db.repositories.PlayerRepository;
import com.dod.service.model.LoginModel;
import com.dod.service.service.AuthenticationService;
import com.dod.service.service.IAuthenticationService;

import javax.servlet.http.HttpServletRequest;
import javax.validation.constraints.Max;
import javax.validation.constraints.Min;
import javax.validation.constraints.NotNull;
import javax.ws.rs.FormParam;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
import java.sql.SQLException;
import org.glassfish.grizzly.http.server.Request;
import org.hibernate.validator.constraints.Length;

/**
 * <pre>
 *      Manages registering and logging in a player
 *       Creates the session that other controllers can use to fetch user
details
 * </pre>
 */
@Path("player")
```

```
public class PlayerController {

    @Context
    private Request request;
    IAuthenticationService service;

    public PlayerController() {
        service = new AuthenticationService(new PlayerRepository());
    }

    /**
     * Authorises a user and starts a session with them
     * @param username must be unique, not empty and less than 256 characters
     * @param password must not be empty and less than 256 characters
     * @return Response with blank body, 200 if successful otherwise 400 or
500
     */
    @POST
    @Produces(MediaType.TEXT_PLAIN)
    @Path("login")
    public Response login(
            @NotNull @Length(min = 1, max =255) @FormParam("username") String
username,
            @NotNull @Length(min = 1, max =255) @FormParam("password") String
password
    ) {
        boolean isAuthorised = service.Login(new LoginModel(username,
password));
        if(isAuthorised) {
            request.getSession(true);
            request.getSession().setAttribute("player",username);
            return Response.ok().build();
        }
        else {
            return Response
                    .status(403)
                    .build();
        }
    }

    /**
     * Registers a user for the service. Username must be unique.
     * @param username must be unique, not empty and less than 256 characters
     * @param password must not be empty and less than 256 characters
     * @return Response with blank body, 200 if successful otherwise 400 or
500
     */
    @POST
    @Produces(MediaType.TEXT_PLAIN)
    @Path("register")
    public Response register(
            @NotNull @Length(min = 1, max =255) @FormParam("username") String
username,
            @NotNull @Length(min = 1, max = 255) @FormParam("password")
String password
    ) {
        boolean success = service.Register(new
LoginModel(username,password));
        if(success) {
            request.getSession(true);
            request.getSession().setAttribute("player", username);
```

```
            return Response.ok().build();
        }
        else {
            return Response.status(400).build();
        }
    }
}
```

4.14.4 ScoreController.java

```
package com.dod.service.controller;

import com.dod.db.repositories.IScoreRepository;
import com.dod.db.repositories.ScoreRepository;
import com.dod.service.model.ScoreboardModel;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
import java.sql.SQLException;

/**
 * Fetches and returns the top scores
 */
@Path("score")
public class ScoreController {

    private IScoreRepository repository;

    public ScoreController() {
        this.repository = new ScoreRepository();
    }

    /**
     * Fetches the top 10 scores across all players.
     * @return Response 200 OK with a JSON encoded ScoreboardModel or 500 if
an error occurred
     */
    @GET
    @Produces(MediaType.APPLICATION_JSON)
    @Path("top")
    public Response top() {
        ScoreboardModel scoreBoard = null;

        try {
            scoreBoard = new ScoreboardModel(repository.getHighestScores());
        }
        catch(SQLException e) {
            e.printStackTrace();
            return Response.serverError().build();
        }

        return
                Response.ok()
                .entity(scoreBoard)
                .build();
    }
}
```

4.15 DungeonOfDooom-master\Sourcecode\project\src\service\src\main\java\com\dod\service\filters

4.15.1 corsFilter.java

```
package com.dod.service.filters;

/**
 * Adds CORS filter to header, enabling cross-origin AJAX requests
 *  Based  on:  https://stackoverflow.com/questions/28065963/how-to-handle-
cors-using-jax-rs-with-jersey
 */
import java.io.IOException;
import javax.ws.rs.container.ContainerRequestContext;
import javax.ws.rs.container.ContainerResponseContext;
import javax.ws.rs.container.ContainerResponseFilter;
import javax.ws.rs.ext.Provider;

@Provider
public class corsFilter implements ContainerResponseFilter {

    /**
     * Adds CORS headers to the Response before sending it
     * @param request ContainerRequestContext
     * @param response ContainerResponseContext
     */
    @Override
    public void filter(ContainerRequestContext request,
                       ContainerResponseContext response) {
        response.getHeaders().add("Access-Control-Allow-Origin",
"http://localhost:63342");
        response.getHeaders().add("Access-Control-Allow-Headers",
                "origin, content-type, accept, authorization");
        response.getHeaders().add("Access-Control-Allow-Credentials",
"true");
        response.getHeaders().add("Access-Control-Allow-Methods",
                "GET, POST, PUT, DELETE, OPTIONS, HEAD");
    }
}
```

4.16 DungeonOfDooom-master\Sourcecode\project\src\service\src\main\java\com\dod\service\model

4.16.1 CharacterModel.java

```
package com.dod.service.model;

import com.dod.models.Point;

import javax.xml.bind.annotation.XmlRootElement;

/**
 * A simpler model of Character for JSON encoding
 */
@XmlRootElement
public class CharacterModel {
    private String playerName;
    private int noCoins;
    private Point position;

    public CharacterModel() { }

    public CharacterModel(String playerName, int noCoins, Point position) {
        this.playerName = playerName;
```

```
        this.noCoins = noCoins;
        this.position = position;
    }

    public String getPlayerName() {
        return playerName;
    }

    public void setPlayerName(String playerName) {
        this.playerName = playerName;
    }

    public int getNoCoins() {
        return noCoins;
    }

    public void setNoCoins(int noCoins) {
        this.noCoins = noCoins;
    }

    public Point getPosition() {
        return position;
    }

    public void setPosition(Point position) {
        this.position = position;
    }
}
```

4.16.2 GameStateModel.java

```
package com.dod.service.model;

import javax.xml.bind.annotation.XmlRootElement;

/**
 * Represents the current GameState. Intended to be communicated to the client
via JSON encoding.
 */
@XmlRootElement
public class GameStateModel {
    private TileModel[] tiles;
    private CharacterModel[] characters;
    private CharacterModel playerCharacter;
    private int minNumOfCoins;
    private boolean hasEnded;

    public GameStateModel() { }

    public GameStateModel(TileModel[] tiles, CharacterModel[] characters,
CharacterModel playerCharacter, boolean hasEnded, int minNumOfCoins) {
        this.tiles = tiles;
        this.characters = characters;
        this.playerCharacter = playerCharacter;
        this.hasEnded = hasEnded;
        this.minNumOfCoins = minNumOfCoins;
    }

    public TileModel[] getTiles() {
        return tiles;
    }

    public void setTiles(TileModel[] tiles) {
```

```
        this.tiles = tiles;
    }

    public CharacterModel[] getCharacters() {
        return characters;
    }

    public void setCharacters(CharacterModel[] characters) {
        this.characters = characters;
    }

    /**
     * The Character belonging to the Player that made the request
     * @return Character
     */
    public CharacterModel getPlayerCharacter() {
        return playerCharacter;
    }

    /**
     * The Character belonging to the Player that made the request
     * @param playerCharacter Character
     */
    public void setPlayerCharacter(CharacterModel playerCharacter) {
        this.playerCharacter = playerCharacter;
    }

    /**
     * Whether the match is ongoing- triggers the client's endgame if true
     * @return boolean
     */
    public boolean isHasEnded() {
        return hasEnded;
    }

    /**
     * Whether the match is ongoing- triggers the client's endgame if true
     * @param hasEnded boolean
     */
    public void setHasEnded(boolean hasEnded) {
        this.hasEnded = hasEnded;
    }

    /**
     * The minimum number of coins needed to win the Match
     * @return int
     */
    public int getMinNumOfCoins() { return minNumOfCoins; }

    /**
     * * The minimum number of coins needed to win the Match
     * @param minNumOfCoins int
     */
    public void setMinNumOfCoins(int minNumOfCoins) { this.minNumOfCoins =
minNumOfCoins; }
}
```

4.16.3 LoginModel.java

```
package com.dod.service.model;

import com.dod.models.Player;
```

```
/**
 *   Simple   model/bean   used   to   pass   information   to/from   the
AuthorisationService
 */
public class LoginModel {

    private String userName;
    private String password;

    public LoginModel(String userName, String password) {
        this.userName = userName;
        this.password = password;
    }

    public String getUserName() {
        return userName;
    }

    public void setUserName(String userName) {
        this.userName = userName;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    /**
     * Convenience method to return the LoginModel's username in the Player
model
     * @return Player
     */
    public Player asPlayer() {
        return new Player(userName);
    }
}
```

4.16.4 MatchResultModel.java

```
package com.dod.service.model;

import javax.xml.bind.annotation.XmlRootElement;

/**
 * Models the information the client needs to display the end-game screen
when the game ends.
 */
@XmlRootElement
public class MatchResultModel {

    private String winner;
    private int winnerCoins;
    private int score;

    public MatchResultModel(String winner, int winnerCoins, int score) {
        this.winner = winner;
        this.winnerCoins = winnerCoins;
        this.score = score;
```

```
    }

    public MatchResultModel() { }

    public String getWinner() {
        return winner;
    }

    public void setWinner(String winner) {
        this.winner = winner;
    }

    public int getWinnerCoins() {
        return winnerCoins;
    }

    public void setWinnerCoins(int winnerCoins) {
        this.winnerCoins = winnerCoins;
    }

    public int getScore() { return score; }

    public void setScore(int score) { this.score = score; }
}
```

4.16.5 MatchStatus.java

```
package com.dod.service.model;

import com.dod.models.Match;

import javax.xml.bind.annotation.XmlID;
import javax.xml.bind.annotation.XmlRootElement;
import java.util.UUID;

/**
 * Models the current state of a lobbying match.
 */
@XmlRootElement
public class MatchStatus
{
    private String[] playerNames;
    @XmlID
    private UUID id;
    private String state;

    public MatchStatus() {}

    public MatchStatus(Match match) {
        this.playerNames = match.getPlayerNames();
        this.id = match.getId();
        this.state = match.getState().toString();
    }

    public String[] getPlayerNames() {
        return playerNames;
    }

    public UUID getId() {
        return id;
    }
```

```
    public void setPlayerNames(String[] playerNames) {
        this.playerNames = playerNames;
    }

    public void setId(UUID id) {
        this.id = id;
    }

    public String getState() {
        return state;
    }

    public void setState(String state) {
        this.state = state;
    }
}
```

4.16.6 ScoreboardModel.java

```
package com.dod.service.model;

import com.dod.models.Score;

import javax.xml.bind.annotation.XmlRootElement;

/**
 * Models a collection of scores to be displayed on a score table
 */
@XmlRootElement
public class ScoreboardModel {
    Score[] scores;

    public ScoreboardModel(Score[] scores) {
        this.scores = scores;
    }

    public ScoreboardModel() {

    }

    public Score[] getScores() {
        return scores;
    }

    public void setScores(Score[] scores) {
        this.scores = scores;
    }
}
```

4.16.7 TileModel.java

```
package com.dod.service.model;

import com.dod.models.Point;

import javax.xml.bind.annotation.XmlRootElement;

/**
 * A simpler Tile model just for JSON encoding
 */
@XmlRootElement
public class TileModel {
    private int type;
```

```
    private Point position;

    public TileModel() { }

    public TileModel(int type, Point position) {
        this.type = type;
        this.position = position;
    }

    public int getType() {
        return type;
    }

    public void setType(int type) {
        this.type = type;
    }

    public Point getPosition() {
        return position;
    }

    public void setPosition(Point position) {
        this.position = position;
    }
}
```

4.17 DungeonOfDooom-master\Sourcecode\project\src\service\src\main\java\com\dod\service\service

4.17.1 AuthenticationService.java

```
package com.dod.service.service;

import com.dod.db.repositories.IPlayerRepository;
import com.dod.models.Player;
import com.dod.service.model.LoginModel;
import org.apache.commons.codec.binary.Base64;

import javax.crypto.SecretKey;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.PBEKeySpec;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;
import java.security.spec.InvalidKeySpecException;
import java.sql.SQLException;

/**
 * <pre>
 *     Handles authenticating a user against their user/pass combo
 *     Uses a salt, generated using a secure RNG
 *     Uses PlayerRepository to fetch Player database details
 * </pre>
 */
public class AuthenticationService implements IAuthenticationService {

    IPlayerRepository repository;

    public AuthenticationService(IPlayerRepository repository) {
        this.repository = repository;
    }

    /**
     * Registers a new user
```

```
     * @param model LoginModel containing the user/pass to be registered
     * @return boolean true if successful otherwise false
     */
    @Override
    public boolean Register(LoginModel model) {
        boolean result = false;
        Player player = player = model.asPlayer();
        Player repositoryPlayer = null;

        try {
            repositoryPlayer = repository.get(player);
        }
        catch(SQLException e) {
            e.printStackTrace();
        }

        if(repositoryPlayer == null) {
            try {
                generateSalt(player);
                player.setHashedPassword(hashAndSalt(model.getPassword(),
player.getSalt()));
                repository.insert(player);
                result = true;
            } catch (Exception e) {
                result = false;
            }
        }

        return result;
    }

    /**
     * Registers a new user
     * @param model LoginModel containing the user/pass to be authorised
     * @return boolean true if the user is authorised, otherwise false
     */
    @Override
    public boolean Login(LoginModel model) {
        boolean result = false;

        try {
            Player player = repository.get(model.asPlayer());
            if                              (hashAndSalt(model.getPassword(),
player.getSalt()).equals(player.getHashedPassword())) {
                result = true;
            }
        }
        catch(Exception e) {
            e.printStackTrace();
        }

        return result;
    }

    /**
     * Generates a random secure salt
     * @param player Player to set the salt for- gets inserted into the
database later
     * @throws NoSuchAlgorithmException could be thrown due to a dependency
problem
     */
```

```
    private void generateSalt(Player player) throws NoSuchAlgorithmException
{
        byte[] salt = SecureRandom.getInstance("SHA1PRNG").generateSeed(32);
        player.setSalt(salt);
    }

    /**
     * Hashes and salts a password
     * @param password the password to be hashed/salted
     * @param salt the salt to salt the password with
     * @return String the hashed/salted password
     * @throws NoSuchAlgorithmException could be thrown due to a dependency
problem
     * @throws InvalidKeySpecException could be thrown due to a dependency
problem
     */
    private  String  hashAndSalt(String  password,  byte[]  salt)  throws
NoSuchAlgorithmException, InvalidKeySpecException {
        String hashedPassword = hash(password, salt);
        return Base64.encodeBase64String(salt) + hashedPassword;
    }

    /**
     * Hashes a password
     * @param password the password to be hashed
     * @param salt the salt to secure the password with
     * @return String the hashed password
     * @throws NoSuchAlgorithmException could be thrown due to a dependency
problem
     * @throws InvalidKeySpecException could be thrown due to a dependency
problem
     */
    private   String   hash(String   password,   byte[]   salt)   throws
NoSuchAlgorithmException, InvalidKeySpecException {
        SecretKeyFactory                          f                          =
SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1");
        SecretKey key = f.generateSecret(new PBEKeySpec(
                password.toCharArray(), salt, 20*1000, 256)
        );
        return Base64.encodeBase64String(key.getEncoded());
    }
}
```

4.17.2 IAuthenticationService.java

```
package com.dod.service.service;

import com.dod.service.model.LoginModel;

import java.sql.SQLException;

/**
 * <pre>
 *     Handles authenticating a user against their user/pass combo
 * </pre>
 */
public interface IAuthenticationService {
    /**
     * Registers a new user
     * @param model LoginModel containing the user/pass to be registered
     * @return boolean true if successful otherwise false
     */
```

```
    boolean Register(LoginModel model);
    /**
     * Registers a new user
     * @param model LoginModel containing the user/pass to be authorised
     * @return boolean true if the user is authorised, otherwise false
     */
    boolean Login(LoginModel model);

}
```

4.17.3 IIOService.java

```
package com.dod.service.service;

import org.json.simple.JSONObject;
import org.json.simple.parser.ParseException;

import java.io.IOException;

/**
 * Handles IO within the Service
 */
public interface IIOService {
    /**
     * Fetches an asset as a String
     * @param path String the path to the asset we are to fetch
     * @return String the contents of the asset
     * @throws IOException if the file is missing
     */
    String getString(String path) throws IOException;

    /**
     * Fetches an asset as parsed JSON
     * @param path String the path to the asset we are to fetch
     * @return JSONObject the parsed content of the asset
     * @throws IOException if the file is missing
     * @throws ParseException if the file isn't encoded in valid JSON
     */
    JSONObject    getJsonObject(String    path)    throws    IOException,
ParseException;
}
```

4.17.4 IMatchService.java

```
package com.dod.service.service;

import com.dod.models.Player;
import com.dod.service.model.MatchResultModel;
import com.dod.service.model.MatchStatus;

import java.sql.SQLException;
import java.util.UUID;

/**
 * Manages joining/starting/ending matches.
 */
public interface IMatchService {

    /**
     * Creates a new Match
     * @param userName String username of the Player who is starting the
Match
     * @param level int the number of the level to load for this Match
```

```
     * @return MatchStatus of the newly created Match
     */
    MatchStatus createMatch(String userName, int level);

    /**
     * Changes a Match's state to InGame
     * @param player Player whose ongoing Match will be modified
     */
    void startMatch(Player player);

    /**
     * Returns the MatchStatus for a particular Player's Match
     * @param player Player whose ongoing Match will be fetched
     * @return
     */
    MatchStatus getStatus(Player player);

    /**
     * Removes a Player from their current ongoing Match
     * @param player Player the Player whom will be removed from their ongoing
Match
     */
    void leaveMatch(Player player);

    /**
     * Changes a Match's state to Over
     * @param player Player whose ongoing Match will be modified
     */
    void endMatch(Player player);

    /**
     * Adds the Player to a particular Match
     * @param player Player whom will be added
     * @param matchID UUID of the Match that player will be addd to
     *  @throws SQLException thrown if Player doesn't exist or a SQL
connectivity issue occurs
     */
    void joinMatch(Player player, UUID matchID) throws SQLException;

    /**
     * Get all Matches currently in the Lobbying state
     * @return MatchStatus[] array of all Matches in the Lobbying state
     */
    MatchStatus[] getLobbyingMatches();

    /**
     * Gets the MatchResultModel for a finished Match
     * todo why not remove the Player from the Match at this point rather
than send another request?
     * @param player Player the Player that has a finished Match
     * @return MatchResultModel pertaining to the player's Match
     */
    MatchResultModel getMatchResult(Player player);
}
```

4.17.5 IMovementService.java

```
package com.dod.service.service;

import com.dod.models.Character;
import com.dod.models.Map;
import com.dod.models.Player;
```

```
import com.dod.models.Point;

import java.sql.SQLException;

/**
 * Interface for MovementService.
 * Handles game logic to move a character from one point to another.
 */
public interface IMovementService {

    /**
     * Moves the Player in a particular direction. Will increment player's
gold if interacting with gold coins, can
     * trigger end of the Match when player interacts with Exit.
     * @param direction String a char from {W,S,A,D} pertaining to a
particular direction in the WASD layout
     * @param player Player whom's Character will be moved
     * @return Point that the Player has moved to
     * @throws SQLException if the database cannot be reached or statement
fails while inserting new score
     */
    Point Move(String direction, Player player) throws SQLException;

}
```

4.17.6 IOService.java

```
package com.dod.service.service;

import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;

import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Paths;

/**
 * Handles IO within the Service
 */
public class IOService implements IIOService {

    private String pathToAssets = "..//..//assets";
    private JSONParser parser;

    public IOService(String pathToAssets) {
        this.pathToAssets = pathToAssets;
        parser = new JSONParser();
    }

    public IOService() {
        parser= new JSONParser();
    }

    /**
     * Fetches an asset as a String
     * @param path String the path to the asset we are to fetch
     * @return String the contents of the asset
     * @throws IOException if the file is missing
     */
    @Override
```

```
    public String getString(String path) throws IOException {
        byte[] encoded = Files.readAllBytes(Paths.get(pathToAssets + path));
        return new String(encoded, StandardCharsets.UTF_8);
    }

    /**
     * Fetches an asset as parsed JSON
     * @param path String the path to the asset we are to fetch
     * @return JSONObject the parsed content of the asset
     * @throws IOException if the file is missing
     * @throws ParseException if the file isn't encoded in valid JSON
     */
    @Override
    public JSONObject getJsonObject(String path) throws IOException,
ParseException {
        String input = getString(path);
        return (JSONObject) parser.parse(input);
    }
}
```

4.17.7 IParseService.java

```
package com.dod.service.service;

import com.dod.models.Map;
import org.json.simple.JSONObject;

/**
 * Parses JSON objects- namely the Map
 */
public interface IParseService {
    /**
     * Parses a Map object from it's JSON encoding
     * @param input JSONObject a JSON encoding of the Map
     * @return Map an initialised Map parsed from JSON
     * @throws NullPointerException may be thrown by SimpleJson while parsing
     */
    Map parseMap(JSONObject input) throws NullPointerException;
}
```

4.17.8 IStateService.java

```
package com.dod.service.service;

import com.dod.models.Player;
import com.dod.service.model.GameStateModel;

/**
 * Generates a representation of the current game state form the perspective
of a particular character
 */
public interface IStateService {

    /**
     * Generates and returns a representation of the current game state form
the perspective of a particular character
     * @param player Player the Player a GameStateModel will be generated for
     * @return GameStateModel a model of the current game state
     */
    GameStateModel GetState(Player player);

}
```

4.17.9 IVisibilityService.java

```
package com.dod.service.service;

import com.dod.models.Map;
import com.dod.models.Character;

/**
 * Calculates the visible tiles from the perspective of a particular Character
 */
public interface IVisibilityService {

    /**
     * Generates a copy of a Map with the correct isVisible flags set for
the perspective of a particular Character
     * @param deungeonMap the Map  pchar resides in
     * @param pchar the Character the perspective of which we're generating
visibility with
     * @return a copy of dungeonMap with correct isVisible flags set for the
perspective of pchar
     */
    Map createVisibleMap(Map deungeonMap, Character pchar);

}
```

4.17.10MatchService.java

```
package com.dod.service.service;

import com.dod.db.repositories.IPlayerRepository;
import com.dod.game.IMatchList;
import com.dod.models.*;
import com.dod.models.Character;
import com.dod.service.constant.Assets;
import com.dod.service.model.MatchResultModel;
import com.dod.service.model.MatchStatus;

import java.sql.SQLException;
import java.util.Date;
import java.util.List;
import java.util.UUID;

/**
 * <pre>
 * Manages joining/starting/ending matches.
 * Makes heavy use of MatchList to store matches in memory.
 * Uses PlayerRepository to fetch Player data.
 * Uses IOService and ParseService to load levels when starting a new Match.
 * </pre>
 */
public class MatchService implements IMatchService {

    private IIOService ioService;
    private IParseService parseService;
    private IPlayerRepository playerRepository;
    private IMatchList matchList;

    public MatchService(IIOService ioService, IParseService parseService,
IPlayerRepository playerRepository, IMatchList matchList) {
        this.ioService = ioService;
        this.parseService = parseService;
        this.playerRepository = playerRepository;
        this.matchList = matchList;
```

```
    }

    /**
     * Creates a new Match
     * @param userName String username of the Player who is starting the
Match
     * @param level int the number of the level to load for this Match
     * @return MatchStatus of the newly created Match
     */
    @Override
    public MatchStatus createMatch(String userName, int level) {
        Map map = null;
        Player player;

        try {
            String     path     =     String.format(Assets.MapLevelFormat,
Integer.toString(level));
            map = parseService.parseMap(ioService.getJsonObject(path));
            player = playerRepository.get(new Player(userName));
        }
        catch(Exception e) {
            e.printStackTrace();
            return null;
        }

        Match match = new Match(map);
        match.addCharacter(player, map.getRandomFreeTilePoint());
        for(int i = 0; i < map.getCoinNo(); i++) {

map.getTile(map.getRandomFreeTilePoint()).setType(TileType.Coin.getValue())
;
        }
        matchList.addMatch(match);

        return new MatchStatus(match);
    }

    /**
     * Changes a Match's state to InGame
     * @param player Player whose ongoing Match will be modified
     */
    @Override
    public void startMatch(Player player) {
        Match match = matchList.getMatchForPlayer(player.getUsername());
        Date temp = new Date();
        match.setTimer(temp.getTime());
        match.setState(MatchState.Ingame);
    }

    /**
     * Returns the MatchStatus for a particular Player's Match
     * @param player Player whose ongoing Match will be fetched
     * @return
     */
    @Override
    public MatchStatus getStatus(Player player) {
        if(!matchList.playerHasMatch(player.getUsername())) {
            return null;
        } else {
            Match match = matchList.getMatchForPlayer(player.getUsername());
            return new MatchStatus(match);
```

```
        }
    }

    /**
     * Removes a Player from their current ongoing Match
     * @param player Player the Player whom will be removed from their ongoing
Match
     */
    @Override
    public void leaveMatch(Player player) {
        Match match = matchList.getMatchForPlayer(player.getUsername());

        match.removeCharacter(player);
    }

    /**
     * Changes a Match's state to Over
     * @param player Player whose ongoing Match will be modified
     */
    @Override
    public void endMatch(Player player) {
        Match match = matchList.getMatchForPlayer(player.getUsername());
        matchList.removeMatch(match.getId());
    }

    /**
     * Adds the Player to a particular Match
     * @param player Player whom will be added
     * @param matchID UUID of the Match that player will be addd to
     *  @throws SQLException thrown if Player doesn't exist or a SQL
connectivity issue occurs
     */
    @Override
    public void joinMatch(Player player, UUID matchId) throws SQLException {
        Match match = matchList.getMatch(matchId);
        player = playerRepository.get(player);
        match.addCharacter(player,
match.getMap().getRandomFreeTilePoint());
    }

    /**
     * Get all Matches currently in the Lobbying state
     * @return MatchStatus[] array of all Matches in the Lobbying state
     */
    @Override
    public MatchStatus[] getLobbyingMatches() {
        List<Match> matches = matchList.getLobbyingMatches();
        MatchStatus[] matchStatuses = new MatchStatus[matches.size()];

        for(int i = 0; i < matches.size(); i++) {
            matchStatuses[i] = new MatchStatus(matches.get(i));
        }

        return matchStatuses;
    }

    /**
     * Gets the MatchResultModel for a finished Match
     * todo why not remove the Player from the Match at this point rather
than send another request?
     * @param player Player the Player that has a finished Match
```

```
     * @return MatchResultModel pertaining to the player's Match
     */
    @Override
    public MatchResultModel getMatchResult(Player player) {
        Match match = matchList.getMatchForPlayer(player.getUsername());
        Character winner = match.getCharacterWithHighestCoins();

        return    new    MatchResultModel(winner.getPlayer().getUsername(),
winner.getCollectedCoins(), match.getScore());
    }
}
```

4.17.11MovementService.java

```
package com.dod.service.service;

import com.dod.db.repositories.IScoreRepository;
import com.dod.db.repositories.ScoreRepository;
import com.dod.game.IMatchList;
import com.dod.game.MatchList;
import com.dod.models.*;
import com.dod.models.Character;

import java.sql.SQLException;
import java.util.Date;

/**
 * Implementation of IMovementService
 */
public class MovementService implements IMovementService {

    IMatchList matchList;
    IScoreRepository scoreRepository;

    public MovementService() {
        this.matchList = MatchList.instance();
        this.scoreRepository = (IScoreRepository)new ScoreRepository();
    }

    /**
     * Moves the Player in a particular direction. Will increment player's
gold if interacting with gold coins, can
     * trigger end of the Match when player interacts with Exit.
     * @param direction String a char from {W,S,A,D} pertaining to a
particular direction in the WASD layout
     * @param player Player whom's Character will be moved
     * @return Point that the Player has moved to
     * @throws SQLException if the database cannot be reached or statement
fails while inserting new score
     */
    @Override
    public Point Move(String direction, Player player) throws SQLException {
        Match match = matchList.getMatchForPlayer(player.getUsername());
        Character pChar = match.getCharacter(player.getUsername());
        Map dungeonMap = match.getMap();
        Point newPoint;

        switch (direction) {
            case "W":
                // check if movement valid
                newPoint      =      new      Point(pChar.getPosition().x,
pChar.getPosition().y - 1);
```

```
                return updatePosition(newPoint, dungeonMap, pChar);
            case "D":
                newPoint    =    new    Point(pChar.getPosition().x    +    1,
pChar.getPosition().y);
                return updatePosition(newPoint, dungeonMap, pChar);
            case "S":
                newPoint      =      new      Point(pChar.getPosition().x,
pChar.getPosition().y + 1);
                return updatePosition(newPoint, dungeonMap, pChar);
            case "A":
                newPoint    =    new    Point(pChar.getPosition().x    -    1,
pChar.getPosition().y);
                return updatePosition(newPoint, dungeonMap, pChar);
            default:
                return pChar.getPosition();
        }
    }

    /**
     * Decides whether or not to update the Player's Position and interacts
with special Tiles.
     * @param newPoint Point the Point the Character wishes to move to
     * @param dungeonMap Map that the Character is moving in
     * @param pChar Character that is moving
     * @return Point the Point that the Character is now in
     * @throws SQLException if the database cannot be reached or statement
fails while inserting new score
     */
    private Point updatePosition(Point newPoint, Map dungeonMap, Character
pChar) throws SQLException {
        if            (dungeonMap.getTile(newPoint).getType()            ==
TileType.Empty.getValue()) {
            pChar.setPosition(newPoint);
        }     else     if     (dungeonMap.getTile(newPoint).getType()     ==
TileType.Coin.getValue()){
            pChar.setPosition(newPoint);
            if (!pChar.getCollectedCoinsPos().contains(newPoint)) {
                pChar.setCollectedCoins(pChar.getCollectedCoins() + 1);
                pChar.addCollectedCoinsPos(newPoint);
            }
        }
        else          if(dungeonMap.getTile(newPoint).getType()          ==
TileType.Exit.getValue()) {
            if(pChar.getCollectedCoins() > dungeonMap.getCoinWin()) {
                pChar.setPosition(newPoint);
                Match                          match                          =
matchList.getMatchForPlayer(pChar.getPlayer().getUsername());
                match.setState(MatchState.Over);

                Date date = new Date();
                match.setTimer(date.getTime() - match.getTimer());
                int  score  =  ((int)  ((double)pChar.getCollectedCoins()  /
(double)match.getTimer() * 10000000));

                match.setScore(score);
                scoreRepository.insert(new
Score(pChar.getPlayer().getUsername(), score));
            }
        }
        return pChar.getPosition();
    }
```

```
}
```

4.17.12ParseService.java

```
package com.dod.service.service;

import com.dod.models.Map;
import com.dod.models.Point;
import com.dod.models.Tile;
import org.json.simple.JSONArray;
import org.json.simple.JSONObject;

import java.util.Iterator;

/**
 * Implementation of IParseService.
 */
public class ParseService implements IParseService {

    /**
     * Parses a Map object from it's JSON encoding
     * @param input JSONObject a JSON encoding of the Map
     * @return Map an initialised Map parsed from JSON
     * @throws NullPointerException may be thrown by SimpleJson while parsing
     */
    @Override
    public Map parseMap(JSONObject input) throws NullPointerException {
        JSONObject level = getLevel(input);
        JSONArray rowsOfTiles = (JSONArray) level.get("map");
        int xSize = ((JSONArray) rowsOfTiles.get(0)).size();
        int ySize = rowsOfTiles.size();

        Map map = new Map(
                (String) level.get("name"),
                ((Long) (level.get("coin_num"))).intValue(),
                ((Long) (level.get("coin_win"))).intValue(),
                ((Long) (level.get("Width"))).intValue(),
                ((Long) (level.get("Height"))).intValue(),
                new Point(xSize, ySize));

        for (int y = 0; y < ySize; y++) {
            JSONArray row = (JSONArray) rowsOfTiles.get(y);
            for (int x = 0; x < xSize; x++) {
                JSONObject tile = (JSONObject) row.get(x);
                map.setTile(new     Point(x,     y),     new     Tile(((Long)
tile.get("type")).intValue()));
            }
        }

        return map;
    }

    /**
     * Figures out the level name based on the number of the level and returns
the initial element
     * @param input the level numer
     * @return JSONObject of the Map object
     */
    private JSONObject getLevel(JSONObject input) {
        Iterator<String> keys = input.keySet().iterator();
        String levelKey = keys.hasNext() ? keys.next() : "";
```

```
        return (JSONObject)input.get(levelKey);
    }
}
```

4.17.13StateService.java

```
package com.dod.service.service;

import com.dod.game.IMatchList;
import com.dod.models.Character;
import com.dod.models.*;
import com.dod.service.model.CharacterModel;
import com.dod.service.model.GameStateModel;
import com.dod.service.model.TileModel;

import java.util.ArrayList;
import java.util.List;

/**
 * Generates a representation of the current game state form the perspective
of a particular character
 */
public class StateService implements IStateService {

    IVisibilityService visibilityService;
    IMatchList matchList;

    public  StateService(IVisibilityService  visibilityService,  IMatchList
matchList) {
        this.visibilityService = visibilityService;
        this.matchList = matchList;
    }

    /**
     * Generates and returns a representation of the current game state form
the perspective of a particular character
     * @param player Player the Player a GameStateModel will be generated for
     * @return GameStateModel a model of the current game state
     */
    @Override
    public GameStateModel GetState(Player player) {
        Match match = matchList.getMatchForPlayer(player.getUsername());
        Map   map   =   visibilityService.createVisibleMap(match.getMap(),
match.getCharacter(player.getUsername()));

        List<TileModel> tiles = new ArrayList();
        List<CharacterModel> characters = new ArrayList();

        for(int x = 0; x < map.getWidth(); x++) {
            for(int y = 0; y < map.getHeight(); y++) {
                Point point = new Point(x,y);
                Tile tile = map.getTile(point);

                if(tile.isVisible()) {
                    tiles.add(new TileModel(tile.getType(), point));
                    List<Character>           charactersOnTile           =
match.getCharactersOnTile(point);
                    for(Character character : charactersOnTile) {
                        characters.add(new CharacterModel(
                                character.getPlayer().getUsername(),
                                character.getCollectedCoins(),
                                character.getPosition()));
```

```
                    }
                }
            }
        }

        Character character = match.getCharacter(player.getUsername());
        return new GameStateModel(tiles.toArray(
                new TileModel[tiles.size()]),
                characters.toArray(new CharacterModel[characters.size()]),
                new CharacterModel(
                        character.getPlayer().getUsername(),
                        character.getCollectedCoins(),
                        character.getPosition()),
                match.getState() == MatchState.Over,
                match.getMap().getCoinWin());
    }
}
```

4.17.14VisibilityService.java

```
package com.dod.service.service;

import com.dod.models.Map;
import com.dod.models.Character;
import com.dod.models.Point;
import com.dod.models.Tile;

/**
 * Calculates the visible tiles from the perspective of a particular Character
 */
public class VisibilityService implements IVisibilityService {

    /**
     * Generates a copy of a Map with the correct isVisible flags set for
the perspective of a particular Character
     * @param deungeonMap the Map  pchar resides in
     * @param pchar the Character the perspective of which we're generating
visibility with
     * @return a copy of dungeonMap with correct isVisible flags set for the
perspective of pchar
     */
    @Override
    public Map createVisibleMap(Map dungeonMap, Character pchar) {
        Map      returnValue      =      new      Map(dungeonMap.getName(),
dungeonMap.getCoinNo(),   dungeonMap.getCoinWin(),   dungeonMap.getWidth(),
dungeonMap.getHeight(),          new          Point(dungeonMap.getWidth(),
dungeonMap.getHeight()));
        for (int i = 0; i < dungeonMap.getWidth(); i++) {
            for (int j = 0; j < dungeonMap.getHeight(); j++) {
                if (pchar.getCollectedCoinsPos().contains(new Point(i, j)))
                    returnValue.setTile(new Point(i, j), new Tile(1, true));
                else
                    returnValue.setTile(new                      Point(i,j),
dungeonMap.getTile(new Point(i, j)));
                if (pchar.getPosition().x - 2 > i || pchar.getPosition().x +
2 < i||pchar.getPosition().y -2 > j||pchar.getPosition().y+2 < j)
                    returnValue.getTile(new                          Point(i,
j)).setVisibility(false);
                else
                    returnValue.getTile(new                          Point(i,
j)).setVisibility(true);
            }
```

```
        }
        return returnValue;
    }
}
```

4.18 DungeonOfDooom-master\Sourcecode\project\src\service\src\main\java\com\dod\service

4.18.1 Main.java

```
package com.dod.service;

import com.dod.service.filters.corsFilter;
import org.glassfish.grizzly.http.server.HttpServer;
import org.glassfish.jersey.grizzly2.httpserver.GrizzlyHttpServerFactory;
import org.glassfish.jersey.server.ResourceConfig;

import java.io.IOException;
import java.net.URI;

/**
 * Main class.
 *
 */
public class Main {
    // Base URI the Grizzly HTTP server will listen on
    public static final String BASE_URI = "http://localhost:8080/";

    /**
     * Starts Grizzly HTTP server exposing JAX-RS resources defined in this
application.
     * @return Grizzly HTTP server.
     */
    public static HttpServer startServer() {
        // create a resource config that scans for JAX-RS resources and
providers
        // in com.dod.service package
        final ResourceConfig rc = new
ResourceConfig().packages("com.dod.service");
        rc.register(new corsFilter());

        // create and start a new instance of grizzly http server
        // exposing the Jersey application at BASE_URI
        return
GrizzlyHttpServerFactory.createHttpServer(URI.create(BASE_URI), rc);
    }

    /**
     * Main method.
     * @param args
     * @throws IOException
     */
    public static void main(String[] args) throws IOException {
        final HttpServer server = startServer();
        System.out.println(String.format("Jersey app started with WADL
available at "
                + "%sapplication.wadl\nHit enter to stop it...", BASE_URI));
        System.in.read();
        server.stop();
    }
}
```

4.19 DungeonOfDooom-master\Sourcecode\project\src\service

4.19.1 pom.xml

```
<project                              xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <groupId>dungeon-of-doom</groupId>
    <artifactId>dungeon-of-doom-service</artifactId>
    <packaging>jar</packaging>
    <version>1.0</version>
    <name>dungeon-of-doom-service</name>

    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>org.glassfish.jersey</groupId>
                <artifactId>jersey-bom</artifactId>
                <version>${jersey.version}</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>

    <dependencies>
        <dependency>
            <groupId>org.glassfish.jersey.containers</groupId>
            <artifactId>jersey-container-grizzly2-http</artifactId>
        </dependency>

         <dependency>
            <groupId>org.glassfish.jersey.media</groupId>
            <artifactId>jersey-media-moxy</artifactId>
        </dependency>

        <dependency>
            <groupId>com.googlecode.json-simple</groupId>
            <artifactId>json-simple</artifactId>
            <version>1.1.1</version>
        </dependency>

        <!--  https://mvnrepository.com/artifact/commons-codec/commons-codec
-->
        <dependency>
            <groupId>commons-codec</groupId>
            <artifactId>commons-codec</artifactId>
            <version>1.10</version>
        </dependency>

        <!--
https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>javax.servlet-api</artifactId>
            <version>3.1.0</version>
        </dependency>

        <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -
```

```
->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.40</version>
        </dependency>

        <dependency>
            <groupId>org.glassfish.jersey.ext</groupId>
            <artifactId>jersey-bean-validation</artifactId>
            <version>2.24.1</version>
        </dependency>

        <dependency>
            <groupId>com.owlike</groupId>
            <artifactId>genson</artifactId>
            <version>1.4</version>
        </dependency>

    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>2.5.1</version>
                <inherited>true</inherited>
                <configuration>
                    <source>1.7</source>
                    <target>1.7</target>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.codehaus.mojo</groupId>
                <artifactId>exec-maven-plugin</artifactId>
                <version>1.2.1</version>
                <executions>
                    <execution>
                        <goals>
                            <goal>java</goal>
                        </goals>
                    </execution>
                </executions>
                <configuration>
                    <mainClass>com.dod.service.Main</mainClass>
                </configuration>
            </plugin>
        </plugins>
    </build>

    <properties>
        <jersey.version>2.24.1</jersey.version>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>
</project>
```

4.20 DungeonOfDooom-master\Sourcecode\project\src\tests\com\dod\test\integration\db

4.20.1 DatabaseConnectionTests.java

```
package dod.test.integration.db;
```

```
import com.dod.db.DatabaseConnection;
import org.junit.Assert;
import org.junit.Test;

import java.sql.Connection;
import java.sql.SQLException;

/**
 * Tests database integration
 */
public class DatabaseConnectionTests {

    @Test
    public void ShouldConnectToDatabase() {
        Connection connection = null;

        try {
            connection = DatabaseConnection.getConnection();
            Assert.assertFalse(connection.isClosed());
        }
        catch(SQLException e) {
            Assert.fail(e.getMessage());
        }
        DatabaseConnection.Close();
    }

    @Test
    public void ShouldCloseDatabase() {
        Connection connection = null;

        try {
            connection = DatabaseConnection.getConnection();
        }
        catch(SQLException e) {
            Assert.fail(e.getMessage());
        }

        DatabaseConnection.Close();

        try {
            Assert.assertTrue(connection.isClosed());
        }
        catch(SQLException e) {
            Assert.fail(e.getMessage());
        }
    }

}
```

4.20.2 DatabaseQueryTests.java

```
package dod.test.integration.db;

import com.dod.db.repositories.PlayerRepository;
import com.dod.db.repositories.ScoreRepository;
import com.dod.models.Player;
import com.dod.models.Score;
import org.apache.commons.codec.digest.DigestUtils;
import org.junit.Assert;
import org.junit.Test;
```

```
import java.sql.SQLException;

/**
 * Unit tests for Database
 */
public class DatabaseQueryTests {

    @Test
    public void shouldReturnTrueIfNewPlayerValueIsAddedInDatabase() {
        PlayerRepository pr = new PlayerRepository();
        String pass = DigestUtils.sha1Hex("1234");
        Player pl = new Player("test", pass, new byte[0]);
        try {
            Assert.assertTrue(pr.insert(pl));
        } catch (SQLException e) {
            Assert.fail(e.toString());
            e.printStackTrace();
        }
    }

    @Test
    public void shouldReturnTrueIfPlayerValueExistsInDatabase() {

        PlayerRepository pr = new PlayerRepository();
        String pass = DigestUtils.sha1Hex("1234");
        Player pl = new Player("test", pass, new byte[0]);
        try {

Assert.assertTrue(pl.getUsername().equals(pr.get(pl).getUsername())       &&
pl.getHashedPassword().equals(pr.get(pl).getHashedPassword()));
        } catch (SQLException e) {
            Assert.fail(e.toString());
            e.printStackTrace();
        }
    }

    @Test
    public void shouldReturnTrueIfPlayerValueIsDeleted() {
        PlayerRepository pr = new PlayerRepository();
        String pass = DigestUtils.sha1Hex("1234");
        Player pl = new Player("test", pass, new byte[0]);
        try {
            Assert.assertTrue(pr.delete(pl));
        } catch (SQLException e) {
            Assert.fail(e.toString());
            e.printStackTrace();
        }
    }

    @Test
    public void shouldReturnTrueIfNewScoreValueIsAdded() {
        ScoreRepository pr = new ScoreRepository();
        Player nPlayer = new Player("test", "1234", new byte[0]);
        Score temp = new Score(nPlayer.getUsername(), 20);
        try {
            Assert.assertTrue(pr.insert(temp));
        } catch (SQLException e) {
            Assert.fail(e.toString());
            e.printStackTrace();
        }
    }
```

```
    @Test
    public void shouldReturnTrueIfScoreValueExistsInDatabase() {
        ScoreRepository pr = new ScoreRepository();
        Score temp = new Score(1, "test", 20);
        try {
            Assert.assertTrue(temp.getId()   ==   pr.get(temp).getId()   &&
temp.getValue()          ==          pr.get(temp).getValue()          &&
temp.getUsername().equals(pr.get(temp).getUsername()));
        } catch (SQLException e) {
            Assert.fail(e.toString());
            e.printStackTrace();
        }
    }

    @Test
    public void shouldReturnTrueIfScoreValueIsDeleted() {
        ScoreRepository pr = new ScoreRepository();
        Score temp = new Score(1, "test", 20);
        try {
            Assert.assertTrue(pr.delete(temp));
        } catch (SQLException e) {
            Assert.fail(e.toString());
            e.printStackTrace();
        }
    }

}
```

4.21 DungeonOfDooom-master\Sourcecode\project\src\tests\com\dod\test\integration\service

4.21.1 AuthenticatedClientTestBase.java

```
package dod.test.integration.service;

import com.dod.game.MatchList;
import com.dod.service.Main;
import com.dod.service.model.MatchStatus;
import org.glassfish.grizzly.http.server.HttpServer;
import org.glassfish.jersey.moxy.json.MoxyJsonConfig;
import org.glassfish.jersey.moxy.json.MoxyJsonFeature;
import org.junit.After;
import org.junit.Before;

import javax.ws.rs.client.Client;
import javax.ws.rs.client.ClientBuilder;
import javax.ws.rs.client.Entity;
import javax.ws.rs.client.WebTarget;
import javax.ws.rs.core.MultivaluedHashMap;
import javax.ws.rs.core.MultivaluedMap;
import javax.ws.rs.core.Response;
import javax.ws.rs.ext.ContextResolver;
import java.util.*;

import static org.junit.Assert.assertEquals;

/**
 * A base class for testing endpoints with sessions
 */
public class AuthenticatedClientTestBase {
    protected WebTarget target;
    protected String testUsername;
```

```
    protected String sessionId;
    protected List<UUID> matchesToRemove;
    private HttpServer server;

    @Before
    public void setUp() {
        server = Main.startServer();

        //Setup JSON client
        Map<String, String> namespacePrefixMapper = new HashMap<String,
String>();
        namespacePrefixMapper.put("http://www.w3.org/2001/XMLSchema-
instance", "xsi");
        MoxyJsonConfig moxyJsonConfig = new MoxyJsonConfig()
                .setNamespacePrefixMapper(namespacePrefixMapper)
                .setNamespaceSeparator(':');

        final ContextResolver<MoxyJsonConfig> jsonConfigResolver =
moxyJsonConfig.resolver();
        Client c = ClientBuilder.newBuilder()
                .register(MoxyJsonFeature.class)
                .register(jsonConfigResolver)
                .build();

        target = c.target(Main.BASE_URI);

        //Generate random user/pass for testing
        testUsername = UUID.randomUUID().toString();

        //Register user/pass so we have a guarunteed user that exists
        sessionId = registerUserAndGetSessionId(testUsername);
        //For cleaning up the static MatchList
        matchesToRemove = new ArrayList();
    }

    @After
    public void tearDown() throws Exception {
        server.stop();

        //Cleanup static data before next turn
        for(UUID id : matchesToRemove) {
            MatchList.instance().removeMatch(id);
        }
    }

    protected String registerUserAndGetSessionId(String identifier) {
        MultivaluedMap<String, String> formData = new
MultivaluedHashMap<String, String>();
        formData.add("username", identifier);
        formData.add("password", identifier);
        Response registerResponse =
target.path("player/register").request().post(Entity.form(formData));

        //get the sessionId so we can send authorised session cookies with
requests
        return registerResponse.getCookies().get("JSESSIONID").getValue();
    }

    protected MatchStatus startNewMatch() {
        MultivaluedMap<String, String> formData = new
MultivaluedHashMap<String, String>();
```

```
        formData.add("level", "1");

        javax.ws.rs.client.Invocation.Builder           request           =
target.path("match/new").request();
        request.cookie("JSESSIONID",sessionId);
        Response result = request.post(Entity.form(formData));
        assertEquals(200, result.getStatus());

        return result.readEntity(MatchStatus.class);
    }
}
```

4.21.2 GameControllerTests.java

```
package dod.test.integration.service;

import com.dod.game.MatchList;
import com.dod.models.Player;
import com.dod.models.Point;
import com.dod.service.model.GameStateModel;
import com.dod.service.model.MatchStatus;
import org.junit.Assert;
import org.junit.Test;

import javax.ws.rs.client.Invocation;
import javax.ws.rs.core.Response;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertNotNull;

/**
 * Tests the GameController
 */
public class GameControllerTests extends AuthenticatedClientTestBase {

    @Test
    public void shouldRespondToStatus() {
        MatchStatus matchStatus = startNewMatch();
        matchesToRemove.add(matchStatus.getId());

        Invocation.Builder request = target.path("game/status").request();
        request.cookie("JSESSIONID",sessionId);

        Response response = request.buildGet().invoke();

        Assert.assertEquals(200, response.getStatus());
        GameStateModel result = response.readEntity(GameStateModel.class);
        assertNotNull(result);
        assertEquals(1, result.getCharacters().length);
        assertEquals(468, result.getTiles().length);
        assertNotNull(result.getTiles()[0].getPosition());
    }

    @Test
    public void shouldRespondToMove() {
        String                          responseMsg                          =
target.path("game/move").request().post(null).readEntity(String.class);
        assertEquals("unimplemented", responseMsg);
    }
}
```

4.21.3 MatchControllerTests.java

```
package dod.test.integration.service;

import com.dod.game.MatchList;
import com.dod.models.Match;
import com.dod.models.MatchState;
import com.dod.models.Player;
import com.dod.models.Point;
import com.dod.service.model.MatchStatus;
import org.junit.Assert;
import org.junit.Test;

import javax.ws.rs.client.*;
import javax.ws.rs.core.MultivaluedHashMap;
import javax.ws.rs.core.MultivaluedMap;
import javax.ws.rs.core.Response;
import java.util.*;

import static junit.framework.Assert.assertNotNull;
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertNull;

/**
 * Tests for MatchController
 * !NOTE! : As of right now these tests will ONLY work if you add a Symboolic
Link directory (mklink /d in Win)
 * to the git root pointing to the Assets folder
 * This is becaues of project config issues... It's a crap solution I know.
 * Potential future solutions:
 *      Add a run parameter that can override the assets folder path
 *     Place a static variable somewhere that can be overridden by the Tests
project, to hold the assets folder path.
 *      Find a way to pass a variable into the HttpServer object that can be
fed to the IOService
 */
public class MatchControllerTests extends AuthenticatedClientTestBase {

    @Test
    public void shouldGiveCurrentMatchStatus() {
        MatchStatus matchStatus = startNewMatch();

        Invocation.Builder request = target.path("match/status").request();
        request.cookie("JSESSIONID",sessionId);
        MatchStatus response = request.get(MatchStatus.class);

        assertNotNull(response);
        matchesToRemove.add(matchStatus.getId());
        assertEquals(matchStatus.getId(), response.getId());
        assertEquals(testUsername, response.getPlayerNames()[0]);
    }

    @Test
    public void whenPlayerHasNoOngoingMatchStatusShouldReturnNull() {
        Invocation.Builder request = target.path("match/status").request();
        request.cookie("JSESSIONID",sessionId);
        MatchStatus response = request.get(MatchStatus.class);

        assertNull(response);
    }

    @Test
    public void shouldCreateNewMatch() {
```

```
        MatchStatus matchStatus = startNewMatch();
        matchesToRemove.add(matchStatus.getId());

        assertNotNull(matchStatus.getId());
        matchesToRemove.add(matchStatus.getId());

        assertEquals(testUsername, matchStatus.getPlayerNames()[0]);
        assertNotNull(MatchList.instance().getMatch(matchStatus.getId()));
    }

    @Test
    public void shouldStartMatch() {
        MatchStatus matchStatus = startNewMatch();
        matchesToRemove.add(matchStatus.getId());

        Invocation.Builder request = target.path("match/start").request();
        request.cookie("JSESSIONID",sessionId);
        Response result = request.post(null);

        assertEquals(200, result.getStatus());
        assertEquals(MatchState.Ingame,
MatchList.instance().getMatch(matchStatus.getId()).getState());
    }

    @Test
    public void joinShouldAddUserToMatch() {
        //Add a match with the original test user
        MatchStatus matchStatus = startNewMatch();
        matchesToRemove.add(matchStatus.getId());
        //Register another user that isn't already a member of the new match
        String newTestUsername = UUID.randomUUID().toString();
        String                        newUserSession                        =
registerUserAndGetSessionId(newTestUsername);

        Invocation.Builder request = target.path("match/join").request();
        request.cookie("JSESSIONID",newUserSession);

        MultivaluedMap<String,       String>       formData       =       new
MultivaluedHashMap<String, String>();
        formData.add("matchId", matchStatus.getId().toString());
        Response response = request.post(Entity.form(formData));

        assertEquals(200, response.getStatus());

Assert.assertTrue(MatchList.instance().getMatch(matchStatus.getId()).hasCha
racter(newTestUsername));

        MatchStatus result = response.readEntity(MatchStatus.class);
        assertNotNull(result);
        assertEquals(matchStatus.getId(), result.getId());
    }

    //todo improve this test so that it doesn't break other tests if it fails
    @Test
    public void listShouldListAllLobbyingMatches() {
        MatchList.instance().addMatch(new Match(null));
        MatchList.instance().addMatch(new Match(null));
        MatchList.instance().addMatch(new Match(null));

        Invocation.Builder request = target.path("match/list").request();
        request.cookie("JSESSIONID",sessionId);
```

```
        Response result = request.get();
        MatchStatus[] response = result.readEntity(MatchStatus[].class);

        assertEquals(200, result.getStatus());
        assertEquals(3, response.length);

        matchesToRemove.add(response[0].getId());
        matchesToRemove.add(response[1].getId());
        matchesToRemove.add(response[2].getId());
    }

    @Test
    public void leaveShouldRemovePlayerFromMatch() {
        MatchStatus matchStatus = startNewMatch();
        matchesToRemove.add(matchStatus.getId());

        Invocation.Builder request = target.path("match/leave").request();
        request.cookie("JSESSIONID",sessionId);
        Response result = request.post(null);

        assertEquals(200, result.getStatus());
        assertEquals(false,
MatchList.instance().getMatch(matchStatus.getId()).hasCharacter(testUsernam
e));
    }
}
```

4.21.4 MyResourceTest.java

```
package dod.test.integration.service;

import javax.ws.rs.client.Client;
import javax.ws.rs.client.ClientBuilder;
import javax.ws.rs.client.WebTarget;

import com.dod.service.Main;
import org.glassfish.grizzly.http.server.HttpServer;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import static org.junit.Assert.assertEquals;

public class MyResourceTest {

    private HttpServer server;
    private WebTarget target;

    @Before
    public void setUp() throws Exception {
        // start the server
        server = Main.startServer();
        // create the client
        Client c = ClientBuilder.newClient();

        // uncomment the following line if you want to enable
        // support for JSON in the client (you also have to uncomment
        //   dependency   on   jersey-media-json   module   in   pom.xml   and
Main.startServer())
        // --
        //                                        c.configuration().enable(new
org.glassfish.jersey.media.json.JsonJaxbFeature());
```

```
        target = c.target(Main.BASE_URI);
    }

    @After
    public void tearDown() throws Exception {
        server.stop();
    }

    /**
     * Test to see that the message "Got it!" is sent in the response.
     */
    @Test
    public void testGetIt() {
        String                          responseMsg                          =
target.path("myresource").request().get(String.class);
        assertEquals("Got it!", responseMsg);
    }
}
```

4.21.5 PlayerControllerTests.java

```
package dod.test.integration.service;

import com.dod.db.repositories.PlayerRepository;
import com.dod.models.Player;
import com.dod.service.Main;
import org.glassfish.grizzly.http.server.HttpServer;
import org.junit.After;
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;

import javax.ws.rs.client.Client;
import javax.ws.rs.client.ClientBuilder;
import javax.ws.rs.client.Entity;
import javax.ws.rs.client.WebTarget;
import javax.ws.rs.core.MultivaluedHashMap;
import javax.ws.rs.core.MultivaluedMap;
import javax.ws.rs.core.Response;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertNotNull;

/**
 * Tests the PlayerController
 */
public class PlayerControllerTests {

    private HttpServer server;
    private WebTarget target;
    private PlayerRepository repository;

    private final String testUsername = "testUsername";
    private       final       String       testNonExistantusername       =
"testNonexistantUsername";
    private final String testPassword = "testPassword";

    @Before
    public void setUp() {
        server = Main.startServer();
        Client c = ClientBuilder.newClient();
```

```
        repository = new PlayerRepository();

        target = c.target(Main.BASE_URI);
    }

    @After
    public void tearDown() throws Exception {
        server.stop();

        try {
            repository.delete(new Player(testUsername));
        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }

    @Test
    public void whenDetailsAreValidShouldRegisterPlayer() throws Exception {
        MultivaluedMap<String,       String>       formData       =       new
MultivaluedHashMap<String, String>();
        formData.add("username", testUsername);
        formData.add("password", testPassword);

        Response response = target.path("player/register")
                .request()
                .post(Entity.form(formData));

        assertEquals("", response.readEntity(String.class));
        assertEquals(200, response.getStatus());
        assertNotNull(repository.get(new Player(testUsername)));
    }

    @Test
    public void whenUsernameEmptyRegisterShouldReturnValidationError() {
        MultivaluedMap<String,       String>       formData       =       new
MultivaluedHashMap<String, String>();
        formData.add("username", "");
        formData.add("password", testPassword);

        Response response = target.path("player/register")
                .request()
                .post(Entity.form(formData));

        assertEquals(400, response.getStatus());
    }

    @Test
    public void whenPasswordEmptyRegisterShouldReturnValidationError() {
        MultivaluedMap<String,       String>       formData       =       new
MultivaluedHashMap<String, String>();
        formData.add("username", testUsername);
        formData.add("password", "");

        Response response = target.path("player/register")
                .request()
                .post(Entity.form(formData));

        assertEquals(400, response.getStatus());
    }
```

```
    @Test
    public void whenPasswordTooLongRegisterShouldReturnValidationError() {
        MultivaluedMap<String,       String>       formData       =       new
MultivaluedHashMap<String, String>();
        formData.add("username", testUsername);
        formData.add("password", generateStringOfSize(257));

        Response response = target.path("player/register")
                .request()
                .post(Entity.form(formData));

        assertEquals(400, response.getStatus());
    }

    @Test
    public void whenUsernameTooLongRegisterShouldReturnValidationError() {
        MultivaluedMap<String,       String>       formData       =       new
MultivaluedHashMap<String, String>();
        formData.add("username", generateStringOfSize(256));
        formData.add("password", testPassword);

        Response response = target.path("player/register")
                .request()
                .post(Entity.form(formData));

        assertEquals(400, response.getStatus());
    }

    @Test
    public                                                                  void
whenUsernameAlreadyTakenRegisterShouldReturnValidationError() {
        MultivaluedMap<String,       String>       formData       =       new
MultivaluedHashMap<String, String>();
        formData.add("username", testUsername);
        formData.add("password", testPassword);

        Response response = target.path("player/register")
                .request()
                .post(Entity.form(formData));

        assertEquals(200, response.getStatus());

        response = target.path("player/register")
                .request()
                .post(Entity.form(formData));

        assertEquals(400, response.getStatus());
    }

    @Test
    public void whenDetailsValidLoginShouldReturnBlankOkStatus() {
        MultivaluedMap<String,       String>       formData       =       new
MultivaluedHashMap<String, String>();
        formData.add("username", testUsername);
        formData.add("password", testPassword);

        //Create player before trying to login
        Response response = target.path("player/register")
                .request()
                .post(Entity.form(formData));
```

```
        assertEquals(200, response.getStatus());

        response = target.path("player/login")
                .request()
                .post(Entity.form(formData));

        Assert.assertEquals(200, response.getStatus());
        assertEquals("", response.readEntity(String.class));
    }

    @Test
    public void whenUsernameEmptyLoginShouldReturnValidationError() {
        MultivaluedMap<String,       String>       formData       =       new
MultivaluedHashMap<String, String>();
        formData.add("username", "");
        formData.add("password", testPassword);

        Response response = target.path("player/login")
                .request()
                .post(Entity.form(formData));

        assertEquals(400, response.getStatus());
    }

    @Test
    public void whenPasswordEmptyLoginShouldReturnValidationError() {
        MultivaluedMap<String,       String>       formData       =       new
MultivaluedHashMap<String, String>();
        formData.add("username", testUsername);
        formData.add("password", "");

        Response response = target.path("player/login")
                .request()
                .post(Entity.form(formData));

        assertEquals(400, response.getStatus());
    }

    @Test
    public void whenPasswordTooLongLoginShouldReturnValidationError() {
        MultivaluedMap<String,       String>       formData       =       new
MultivaluedHashMap<String, String>();
        formData.add("username", testUsername);
        formData.add("password", generateStringOfSize(256));

        Response response = target.path("player/login")
                .request()
                .post(Entity.form(formData));

        assertEquals(400, response.getStatus());
    }

    @Test
    public void whenUsernameTooLongLoginShouldReturnValidationError() {
        MultivaluedMap<String,       String>       formData       =       new
MultivaluedHashMap<String, String>();
        formData.add("username", generateStringOfSize(256));
        formData.add("password", testPassword);

        Response response = target.path("player/login")
                .request()
```

```
                .post(Entity.form(formData));

        assertEquals(400, response.getStatus());
    }

    /**
     * We don't want to return validation here- we don't want to inform a
malicious user
     * when they do or don't randomly guess a correct username
     */
    @Test
    public                                                              void
whenUsernameDoesNotExistLoginShouldReturnBlankAuthorisationError() {
        MultivaluedMap<String,      String>      formData      =       new
MultivaluedHashMap<String, String>();
        formData.add("username", testNonExistantusername);
        formData.add("password", testPassword);

        Response response = target.path("player/login")
                .request()
                .post(Entity.form(formData));

        assertEquals(403, response.getStatus());
        assertEquals("", response.readEntity(String.class));
    }

    private String generateStringOfSize(int size) {
        String result = "";

        for(int i = 0; i < size; i++) {
            result += "z";
        }

        return result;
    }
}
```

4.22 DungeonOfDooom-master\Sourcecode\project\src\tests\com\dod\test\unit\domain\game

4.22.1 MatchListTests.java

```
package dod.test.unit.domain.game;

import com.dod.game.MatchList;
import com.dod.models.Match;
import com.dod.models.MatchState;
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;

import java.util.List;
import java.util.UUID;

import static org.mockito.Mockito.*;

/**
 * Tests for MatchList
 */
public class MatchListTests {

    private final String testUsername = "testUsername";
```

```
    private MatchList matchList;

    @Before
    public void Setup() {
        matchList = new MatchList();
    }

    @Test
    public void shouldGetLobbyingMatches() {
        Match lobbyingMatch = mock(Match.class);
        Match ingameMatch = mock(Match.class);
        Match anotherIngameMatch = mock(Match.class);

        when(lobbyingMatch.getState()).thenReturn(MatchState.Lobbying);
        when(ingameMatch.getState()).thenReturn(MatchState.Ingame);
        when(anotherIngameMatch.getState()).thenReturn(MatchState.Ingame);

        matchList.addMatch(lobbyingMatch);
        matchList.addMatch(ingameMatch);
        matchList.addMatch(anotherIngameMatch);

        List<Match> result = matchList.getLobbyingMatches();

        Assert.assertEquals(1, result.size());
        Assert.assertEquals(lobbyingMatch, result.get(0));
    }

    @Test
    public void shouldGetMatchById() {
        Match matchOne = mock(Match.class);
        Match matchTwo = mock(Match.class);
        UUID idOne = UUID.randomUUID();
        UUID idTwo = UUID.randomUUID();

        when(matchOne.getId()).thenReturn(idOne);
        when(matchTwo.getId()).thenReturn(idTwo);

        matchList.addMatch(matchOne);
        matchList.addMatch(matchTwo);

        Match result = matchList.getMatch(idOne);

        Assert.assertEquals(matchOne, result);
    }

    @Test
    public void shouldGetMatchForPlayer() {
        Match match = mock(Match.class);
        when(match.hasCharacter(testUsername)).thenReturn(true);

        matchList.addMatch(match);

        Match result = matchList.getMatchForPlayer(testUsername);

        Assert.assertEquals(match,result);
    }
}
```

4.23 DungeonOfDooom-master\Sourcecode\project\src\tests\com\dod\test\unit\domain\model

4.23.1 MatchTests.java

```
package dod.test.unit.domain.model;

import com.dod.models.*;
import com.dod.models.Character;
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;

import static org.mockito.Mockito.*;

/**
 * Tests some of the non-trivial Match functions
 */
public class MatchTests {

    Map map;
    Match match;
    Player player;

    private final String testUsername = "testUsername";

    @Before
    public void Setup() {
        map = mock(Map.class);
        match = new Match(map);
        player = mock(Player.class);
        when(player.getUsername()).thenReturn(testUsername);
    }

    @Test
    public void shouldAddCharacter() {
        match.addCharacter(player, new Point(0,0));
        Assert.assertTrue(match.hasCharacter(testUsername));
    }

    @Test
    public void whenThereAreMultipleCharactersShouldGetCorrectCharacter() {
        Player anotherPlayer = mock(Player.class);
        Player anotherAnotherPlayer = mock(Player.class);

when(anotherPlayer.getUsername()).thenReturn("anotherTestUsername");

when(anotherAnotherPlayer.getUsername()).thenReturn("anotherAnotherTestUser
name");

        match.addCharacter(player, new Point(0,0));
        match.addCharacter(anotherPlayer, new Point(0,0));
        match.addCharacter(anotherAnotherPlayer, new Point(0,0));

        Assert.assertEquals(player,
match.getCharacter(testUsername).getPlayer());
    }
}
```

4.24 DungeonOfDooom-master\Sourcecode\project\src\tests\com\dod\test\unit\service

4.24.1 AuthenticationServiceTests.java

```
package dod.test.unit.service;

import com.dod.db.repositories.IPlayerRepository;
import com.dod.models.Player;
```

```
import com.dod.service.model.LoginModel;
import com.dod.service.service.AuthenticationService;
import com.dod.service.service.IAuthenticationService;
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;

import static org.mockito.Mockito.*;

/**
 * Tests for the AuthenticationService
 */
public class AuthenticationServiceTests {

    private IAuthenticationService service;
    private IPlayerRepository repository;

    private final String testPlayername = "test";
    private final String testPassword = "testPassword";
    private final String incorrectTestPassword = "incorrectTestPassword";

    //These two are calculated by the hashing algorithm from testPassword so
should always work
    private final byte[] testSalt = new byte[] {-77,14,44,-103,-37,0,60,-
41,54,60,-24,-69,-10,-14,101,-17,101,
            95,16,50,60,81,34,-90,-85,123,88,88,-18,71,80,93};
    private final String testHashedPassword =

"sw4smdsAPNc2POi79vJl72VfEDI8USKmq3tYWO5HUF0=vNgKzsYRou5lhm4l8i7pFYsYeqeicv
/5O5KeplB2rLY=";

    @Before
    public void Setup() throws Exception {
        repository = mock(IPlayerRepository.class);
        service = new AuthenticationService(repository);
    }

    @Test
    public                                                              void
whenUsernameDoesNotExistRegisterShouldCreatePlayerAndReturnTrue()     throws
Exception {
        when(repository.get(any(Player.class))).thenReturn(null);

        boolean  result  =  service.Register(new  LoginModel(testPlayername,
testPassword));

        verify(repository).insert(any(Player.class));
        Assert.assertEquals(true, result);
    }

    @Test
    public  void  whenUsernameDoesExistRegisterShouldReturnFalse()  throws
Exception {
        when(repository.get(any(Player.class))).thenReturn(new
Player(testPlayername, testPassword, new byte[0]));

        boolean  result  =  service.Register(new  LoginModel(testPlayername,
testPassword));

        Assert.assertEquals(false, result);
    }
```

```
    @Test
    public void whenDetailsAreValidLoginShouldReturnTrue() throws Exception
{
        when(repository.get(any(Player.class))).thenReturn(new
Player(testPlayername, testHashedPassword, testSalt));

        boolean result = service.Login(new LoginModel(testPlayername,
testPassword));

        Assert.assertEquals(true, result);
    }

    @Test
    public void whenPlayerDoesNotExistLoginShouldReturnFalse() throws
Exception {
        when(repository.get(any(Player.class))).thenReturn(null);

        boolean result = service.Login(new LoginModel(testPlayername,
testPassword));

        Assert.assertEquals(false, result);
    }

    @Test
    public void whenPasswordIsWrongLoginShouldReturnFalse() throws Exception
{
        when(repository.get(any(Player.class))).thenReturn(new
Player(testPlayername, testHashedPassword, testSalt));

        boolean result = service.Login(new LoginModel(testPlayername,
incorrectTestPassword));

        Assert.assertEquals(false, result);
    }
}
```

4.24.2 IOServiceTests.java

```
package dod.test.unit.service;

import com.dod.service.constant.Assets;
import com.dod.service.service.IIOService;
import com.dod.service.service.IOService;
import org.json.simple.JSONObject;
import org.json.simple.parser.ParseException;
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;

import java.io.IOException;

/**
 * Unit tests for the IOService
 */
public class IOServiceTests {

    IIOService service;

    private String testAssetPath = "\\test\\test.asset";
    private String expectedTestAssetResult = "testasset :)";
    private String nonExistantTestAssetPath = "nonexistant.asset";
```

```
    private String testJsonPath = "\\test\\test.json";

    @Before
    public void Setup() {
        service = new IOService(".\\assets");
    }

    @Test
    public void shouldGetAssetAtPath() {
        try {
            String result = service.getString(testAssetPath);
            Assert.assertEquals(expectedTestAssetResult, result);
        }
        catch(IOException e) {
            Assert.fail("Unexepected exception thrown by service:" +
e.toString());
            e.printStackTrace();
        }
    }

    @Test
    public void whenPathIsInvalidShouldThrowException() {
        try {
            String result = service.getString(nonExistantTestAssetPath);
            Assert.fail("Service did not throw exception when expected.");
        }
        catch(IOException e) {
            //Pass!
        }
    }

    @Test
    public void shouldParseJsonFile() {
        try {
            JSONObject result = service.getJsonObject(testJsonPath);
            Assert.assertTrue(result.containsKey("id"));
        }
        catch(Exception e) {
            Assert.fail("Unexepected exception thrown by service:" +
e.toString());
            e.printStackTrace();
        }
    }

    @Test
    public void whenJsonIsInvalidShouldThrownParseException() {
        try {
            JSONObject result = service.getJsonObject(testAssetPath);
            Assert.fail("Service did not throw exception when expected.");
        }
        catch(ParseException e) {
            //Pass!
        }
        catch(Exception e) {
            Assert.fail("Unexepected exception thrown by service:" +
e.toString());
            e.printStackTrace();
        }
    }
}
```

4.24.3 MatchServiceTests.java

```
package dod.test.unit.service;

import com.dod.db.repositories.IPlayerRepository;
import com.dod.game.IMatchList;
import com.dod.game.MatchList;
import com.dod.models.*;
import com.dod.service.model.MatchStatus;
import com.dod.service.service.IIOService;
import com.dod.service.service.IOService;
import com.dod.service.service.IParseService;
import com.dod.service.service.MatchService;
import org.json.simple.JSONObject;
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;
import org.mockito.Mock;
import org.mockito.Mockito;

import java.sql.SQLException;
import java.util.UUID;

import static org.junit.Assert.fail;
import static org.mockito.Mockito.*;

/**
 * Tests for MatchService
 */
public class MatchServiceTests {

    MatchService service;

    IIOService ioServiceMock;
    IParseService parseServiceMock;
    IPlayerRepository playerRepositoryMock;
    IMatchList matchListSpy;
    Map mapMock;
    Player playerMock;

    private final int testLevelNo = 0;
    private final String testLevelPath = "/maps/level0.json";
    private final String testUsername = "testUsername";
    private final Point testPoint = new Point(0,0);
    private final int testNumberOfCoins = 10;

    @Before
    public void setup() {
        ioServiceMock = mock(IOService.class);
        parseServiceMock = mock(IParseService.class);
        playerRepositoryMock = mock(IPlayerRepository.class);
        matchListSpy = spy(new MatchList());
        mapMock = mock(Map.class);
        playerMock = mock(Player.class);
        when(playerMock.getUsername()).thenReturn(testUsername);

        service = new MatchService(ioServiceMock, parseServiceMock,
playerRepositoryMock, matchListSpy);
    }

    @Test
    public void shouldCreateMatch() throws Exception {
```

```
        when(ioServiceMock.getJsonObject(any(String.class))).thenReturn(new
JSONObject());

when(parseServiceMock.parseMap(any(JSONObject.class))).thenReturn(mapMock);
        when(playerRepositoryMock.get(any(Player.class))).thenReturn(new
Player(testUsername));
        when(mapMock.getRandomFreeTilePoint()).thenReturn(testPoint);
        when(mapMock.getCoinNo()).thenReturn(testNumberOfCoins);
        when(mapMock.getTile(any(Point.class))).thenReturn(new Tile(0));


        MatchStatus result = service.createMatch(testUsername,testLevelNo);

        verify(matchListSpy).addMatch(any(Match.class));
        Assert.assertTrue(matchListSpy.playerHasMatch(testUsername));
        Assert.assertEquals(result.getId(),
matchListSpy.getMatchForPlayer(testUsername).getId());
        Assert.assertEquals(testPoint,

matchListSpy.getMatchForPlayer(testUsername).getCharacter(testUsername).get
Position());
    }

    @Test
    public                                                              void
WhenCreatingMatchShouldAssignRandomCharacterAndCoinPositions()       throws
Exception {
        when(ioServiceMock.getJsonObject(any(String.class))).thenReturn(new
JSONObject());

when(parseServiceMock.parseMap(any(JSONObject.class))).thenReturn(mapMock);
        when(playerRepositoryMock.get(any(Player.class))).thenReturn(new
Player(testUsername));
        when(mapMock.getRandomFreeTilePoint()).thenReturn(testPoint);
        when(mapMock.getCoinNo()).thenReturn(testNumberOfCoins);
        when(mapMock.getTile(any(Point.class))).thenReturn(new Tile(0));


        MatchStatus result = service.createMatch(testUsername,testLevelNo);

        verify(mapMock,              times(testNumberOfCoins              +
1)).getRandomFreeTilePoint();
        Assert.assertEquals(testPoint,

matchListSpy.getMatchForPlayer(testUsername).getCharacter(testUsername).get
Position());
    }

    @Test
    public void shouldStartMatch() {
        Match matchSpy = spy(new Match(null));
        matchListSpy.addMatch(matchSpy);
        matchSpy.addCharacter(playerMock, testPoint);

        service.startMatch(playerMock);

        verify(matchListSpy, times(1)).getMatchForPlayer(testUsername);
        verify(matchSpy, times(1)).setState(MatchState.Ingame);
        Assert.assertEquals(MatchState.Ingame, matchSpy.getState());
    }
```

```
    @Test
    public void shouldGetMatchStatus() {
        Match matchSpy = spy(new Match(null));
        matchSpy.addCharacter(playerMock, testPoint);

        matchListSpy.addMatch(matchSpy);

        MatchStatus result = service.getStatus(playerMock);

        verify(playerMock, atLeastOnce()).getUsername();
        verify(matchListSpy, times(1)).playerHasMatch(testUsername);
        Assert.assertEquals(matchSpy.getId(), result.getId());
    }

    @Test
    public void whenPlayerHasNoMatchGetStatusShouldReturnNull() {
        MatchStatus result = service.getStatus(playerMock);

        Assert.assertNull(result);
    }

    @Test
    public void leaveMatchShouldRemoveCharacterFromMatch() {
        Match matchSpy = spy(new Match(null));
        matchSpy.addCharacter(playerMock, testPoint);

        matchListSpy.addMatch(matchSpy);

        service.leaveMatch(playerMock);

        Assert.assertNull(matchSpy.getCharacter(testUsername));
    }

    @Test
    public void endMatchShouldRemoveMatchFromMatchList() {
        Match matchSpy = spy(new Match(null));
        matchSpy.addCharacter(playerMock, testPoint);

        matchListSpy.addMatch(matchSpy);

        service.endMatch(playerMock);

        verify(matchListSpy, times(1)).removeMatch(matchSpy.getId());
        Assert.assertNull(matchListSpy.getMatchForPlayer(testUsername));
    }

    @Test
    public void joinMatchShoulAddPlayerToMatch() throws Exception {
        when(mapMock.getRandomFreeTilePoint()).thenReturn(testPoint);

when(playerRepositoryMock.get(any(Player.class))).thenReturn(playerMock);
        Match matchSpy = spy(new Match(mapMock));
        matchListSpy.addMatch(matchSpy);

        service.joinMatch(playerMock, matchSpy.getId());

        Assert.assertTrue(matchSpy.hasCharacter(testUsername));
    }

    @Test
    public void whenSqlExceptionoccursJoinMatchShouldThrowException() throws
```

```
Exception {
        when(mapMock.getRandomFreeTilePoint()).thenReturn(testPoint);
        when(playerRepositoryMock.get(any(Player.class))).thenThrow(new
SQLException());
        Match matchSpy = spy(new Match(mapMock));
        matchListSpy.addMatch(matchSpy);

        try {
            service.joinMatch(playerMock, matchSpy.getId());
            fail();
        }
        catch(SQLException e) {
            //success!
        }
        catch(Exception e) {
            fail();
        }
    }

    @Test
    public void getLobbyingMatchesShouldOnlyReturnMatchesInLobbyState() {
        Match lobbyingMatchMock = mock(Match.class);
        when(lobbyingMatchMock.getState()).thenReturn(MatchState.Lobbying);
        Match inGameMatchMock = mock(Match.class);
        when(inGameMatchMock.getState()).thenReturn(MatchState.Ingame);

        UUID testId = UUID.randomUUID();
        when(lobbyingMatchMock.getId()).thenReturn(testId);

        matchListSpy.addMatch(lobbyingMatchMock);
        matchListSpy.addMatch(inGameMatchMock);

        MatchStatus[] result = service.getLobbyingMatches();

        Assert.assertEquals(1, result.length);
        Assert.assertEquals(testId, result[0].getId());
    }

    @Test
    public                                                                  void
whenNoMatchesInLobbyStateGetLobbyingMatchesShouldReturnEmptyArray() {
        MatchStatus[] result = service.getLobbyingMatches();

        Assert.assertNotNull(result);
        Assert.assertEquals(0, result.length);
    }
}
```

4.24.4 MovementTests.java

```
package dod.test.unit.service;

import com.dod.models.Character;
import com.dod.models.Map;
import com.dod.models.Player;
import com.dod.models.Point;
import com.dod.service.service.IOService;
import com.dod.service.service.MovementService;
import com.dod.service.service.ParseService;
import org.json.simple.JSONObject;
import org.json.simple.parser.ParseException;
import org.junit.Assert;
```

```
import org.junit.Before;
import org.junit.Test;

import java.io.IOException;

/**
 * Created by tasos on 11/12/2016.
 */
public class MovementTests {

    private IOService ioService;
    private ParseService parService;
    private Map dungeonMap;
    private Character pChar, pChar2;
    private JSONObject jobject;

    @Before
    public void Setup() {
        ioService = new IOService(".\\assets");
        try {
            jobject = ioService.getJsonObject("\\maps\\Level1.json");
        } catch (IOException e) {
            e.printStackTrace();
        } catch (ParseException e) {
            e.printStackTrace();
        }
        parService = new ParseService();
        dungeonMap = parService.parseMap(jobject);
        pChar = new Character(new Point(4, 4), new Player("test"));
        pChar2 = new Character(new Point(3, 1), new Player("dadasda"));
    }

    @Test
    public void shouldReturnTrueIfPlayerMovedToRightTile() throws Exception
{
        MovementService moveService = new MovementService();
        Assert.assertTrue(moveService.Move("D",                          new
Player("test")).equals(new Point(5,4)));
    }

    @Test
    public void shouldReturnFalseIfPlayerMovedToRightTile() throws Exception
{
        MovementService moveService = new MovementService();
        Assert.assertFalse(moveService.Move("D",                         new
Player("test")).equals(new Point(4,4)));
    }

    @Test
    public void shouldReturnFalseIfPlayerMovesToWall() throws Exception {
        MovementService moveService = new MovementService();
        Assert.assertFalse(moveService.Move("D",                         new
Player("test")).equals(new Point(3,0)));
    }

    @Test
    public void shouldReturnTrueIfPlayerCantMoveToWall() throws Exception {
        MovementService moveService = new MovementService();
        Assert.assertFalse(moveService.Move("D",                         new
Player("test")).equals(new Point(3,1)));
    }
```

```
}
```

4.24.5 ParseServiceTests.java

```
package dod.test.unit.service;

import com.dod.models.Map;
import com.dod.models.Point;
import com.dod.service.service.IParseService;
import com.dod.service.service.ParseService;
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;

/**
 * Tests for the ParseService
 */
public class ParseServiceTests {

    private IParseService service;
    private JSONParser parser;

    private String validJson =
            "{           \"testLev\":           {           \"id\":\"test\",
\"name\":\"test\",\"coin_num\":6,    \"coin_win\":5,    \"Width\"    :    26,
\"Height\" : 18, \"tiles\": [" +
            "{\"id\":
\"tile_wall\",\"name\":\"wall\",\"type\":0,\"visibility\":true,\"touchable\
":false}," +
            "{\"id\":
\"tile_path\",\"name\":\"path\",\"type\":1,\"visibility\":true,\"touchable\
":true}," +
            "{\"id\":
\"tile_path2\",\"name\":\"path2\",\"type\":2,\"visibility\":true,\"touchabl
e\":true}" +
                    "],
\"map\":[ [{\"type\":0},{\"type\":0},{\"type\":0},{\"type\":0}], " +
            "[{\"type\":0},{\"type\":0},{\"type\":0},{\"type\":0}]] } }";

    private String invalidJson =
            "{ \"testLev\": { \"id\":\"test\", \"tiles\": [" +
            "{\"id\":
\"tile_wall\",\"name\":\"wall\",\"type\":0,\"visibility\":true,\"touchable\
":false}," +
            "{\"id\":
\"tile_path\",\"name\":\"path\",\"type\":1,\"visibility\":true,\"touchable\
":true}," +
            "{\"id\":
\"tile_path2\",\"name\":\"path2\",\"type\":2,\"visibility\":true,\"touchabl
e\":true}" +
            "],
\"map\":[ [{\"type\":0},{\"type\":0},{\"type\":0},{\"type\":0}], " +
            "[{\"type\":0},{\"type\":0},{\"type\":0},{\"type\":0}]] } }";

    @Before
    public void Setup() {
        service = new ParseService();
        parser = new JSONParser();
    }
```

```
    @Test
    public void shouldGenerateMapFromJson() {
        try {
            JSONObject input = (JSONObject) parser.parse(validJson);
            Map result = service.parseMap(input);

            Assert.assertEquals(5, result.getCoinWin());
            Assert.assertEquals(6, result.getCoinNo());
            Assert.assertEquals("test", result.getName());

            for(int x = 0; x < 4; x++) {
                for(int y = 0; y < 2; y++) {
                    Assert.assertEquals(0,               result.getTile(new
Point(x,y)).getType());
                }
            }
        }
        catch(Exception e) {
            e.printStackTrace();
            Assert.fail("Unexepected   exception   thrown   by   service:"   +
e.toString());
        }
    }

    @Test
    public void whenJsonIsInvalidShouldThrowException() {
        try {
            JSONObject input = (JSONObject) parser.parse(invalidJson);
            Map result = service.parseMap(input);
            Assert.fail("Test did not throw expected exception.");
        }
        catch(NullPointerException e) {
            //Passed!
        }
        catch(Exception e) {
            Assert.fail("Unexepected   exception   thrown   by   service:"   +
e.toString());
            e.printStackTrace();
        }
    }
}
```

4.24.6 StateServiceTests.java

```
package dod.test.unit.service;

import com.dod.game.MatchList;
import com.dod.models.*;
import com.dod.service.model.GameStateModel;
import com.dod.service.service.IOService;
import com.dod.service.service.IVisibilityService;
import com.dod.service.service.ParseService;
import com.dod.service.service.StateService;
import org.json.simple.JSONObject;
import org.json.simple.parser.ParseException;
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;

import java.io.IOException;
```

```
import static org.junit.Assert.fail;
import static org.mockito.Mockito.*;

/**
 * Tests for the StateService.
 */
public class StateServiceTests {

    private IVisibilityService visibilityServiceMock;
    private MatchList matchListMock;
    private StateService stateService;
    private Map map;
    private Point testPoint;
    private Match match;

    private String testUsername = "testUsername";

    @Before
    public void Setup() {
        visibilityServiceMock = mock(IVisibilityService.class);
        matchListMock = mock(MatchList.class);

        stateService     =     new     StateService(visibilityServiceMock,
matchListMock);

        IOService ioService = new IOService();
        JSONObject jobject = null;
        try {
            jobject = ioService.getJsonObject("\\maps\\level1.json");
        } catch (IOException e) {
            e.printStackTrace();
        } catch (ParseException e) {
            e.printStackTrace();
        }
        ParseService parService = new ParseService();
        map = parService.parseMap(jobject);

        match = new Match(map);
        testPoint = map.getRandomFreeTilePoint();
        match.addCharacter(new Player(testUsername), testPoint);


when(matchListMock.getMatchForPlayer(testUsername)).thenReturn(match);
    }

//    @Test
//    public void shouldGetCurrentStateOfGame() {
//                    GameStateModel   result   =   stateService.GetState(new
Player(testUsername));
//               when(visibilityServiceMock.getVisibleTilesForCharacter(map,
match.getCharacter(testUsername))).thenReturn(map);
//
//                                                Assert.assertEquals(testPoint,
result.getCharacters()[0].getPosition());
//                  Assert.assertEquals(map.getWidth()   *   map.getHeight(),
result.getTiles().length);
//        Assert.assertEquals(1, result.getCharacters().length);
//    }
//
//    @Test
//    public void shouldOnlyReturnVisibleTiles() {
```

```
//              when(visibilityServiceMock.getVisibleTilesForCharacter(map,
match.getCharacter(testUsername))).thenReturn(null);
//        //todo
//        fail();
//    }
}
```

4.24.7 VisibilityServiceTest.java

```
package dod.test.unit.service;

import com.dod.models.Map;
import com.dod.models.Player;
import com.dod.models.Point;
import com.dod.models.Character;
import com.dod.service.service.IOService;
import com.dod.service.service.ParseService;
import com.dod.service.service.VisibilityService;
import org.json.simple.JSONObject;
import org.json.simple.parser.ParseException;
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;

import java.io.IOException;

/**
 * Created by tasos on 7/12/2016.
 */
public class VisibilityServiceTest {

    private IOService ioService;
    private ParseService parService;
    private Map dungeonMap;
    private JSONObject jobject;
    private Character pChar;
    private Map visibleMap;

    @Before
    public void Setup() {
        ioService = new IOService(".\\assets");
        visibleMap = new Map("level1", 30, 20, 26, 18, new Point(26, 18));
        try {
            jobject = ioService.getJsonObject("\\maps\\Level1.json");
        } catch (IOException e) {
            e.printStackTrace();
        } catch (ParseException e) {
            e.printStackTrace();
        }
        parService = new ParseService();
        dungeonMap = parService.parseMap(jobject);
        pChar = new Character(new Point(4, 4), new Player("test"));
    }

    @Test
    public void shouldReturnTrueIfTheTile34IsVisible() {
        VisibilityService vService = new VisibilityService();
        visibleMap = vService.createVisibleMap(dungeonMap, pChar);

        Assert.assertTrue(visibleMap.getTile(new Point(3, 4)).isVisible());
    }
```

```
    @Test
    public void shouldReturnFalseIfTheTile77IsNotVisible() {
        VisibilityService vService = new VisibilityService();
        visibleMap = vService.createVisibleMap(dungeonMap, pChar);

        Assert.assertFalse(visibleMap.getTile(new Point(7,7)).isVisible());
    }
}
```

4.25 DungeonOfDooom-master\Sourcecode\project\src\tests

4.25.1 pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>dungeon-of-doom</groupId>
    <artifactId>dungoen-of-doom-tests</artifactId>
    <version>1.0</version>

    <dependencies>
        <!-- https://mvnrepository.com/artifact/org.mockito/mockito-all -->
        <dependency>
            <groupId>org.mockito</groupId>
            <artifactId>mockito-all</artifactId>
            <version>1.9.5</version>
        </dependency>
        <dependency>
            <groupId>dungeon-of-doom</groupId>
            <artifactId>dungeon-of-doom-service</artifactId>
            <version>1.0</version>
        </dependency>
    </dependencies>
</project>
```

5 Project Diaries

5.1 Mattsi Jansky:

Date	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
31s Oct	Attended first group meeting		Attended second group meeting			Reviewed lecture slides/notes and spec in prep for Monday meet	Reviewed lecture slides/notes and spec in prep for Monday meet
7th Nov	Attended group meeting Gave presentation on findings Worked with group on requirements analysis Introduced team to Trello & setup backlog/sprint boards Kept meeting notes Proposed choosing leader	Worked with Tassos to produce several usecases	Attended group meeting Worked with group on CRC cards Added some tasks to Trello				
14th Nov	Attended group meeting Worked with group on CRCs and task estimations	Produced timescale document, to be discussed with team Weds.	Attended team meeting Introduced team to timescale document, made final changes to it.		Worked with Tassos on database creation, basic project layout and interfacing with database.		
21st Nov	Attended team meeting Worked on UML		Attended team meeting			Worked on Test Plan and researched Java API frameworks	Worked on Test Plan and researched Java API frameworks
28th Nov	Attended team meeting	Setup Jersey API framework	Worked with Qian to: implement IOservice, add		Worked with Qian to implement JSON service		

	Arranged next Sprint Created all remaining Trello cards and filled in various technical details of cards Had meeting with Julian Tested Java API framework	Refactored project structure Worked with Tassos to setup skeleton of project ie blank endpoints, services and tests	JSON framework to project, start JSON service (implement tests)				
5th Dec	Worked with Selin and Arya to review their work on the login/registration card Picked up the login/registration card and started work on it Set up Sprint 5 in Trello Attended team meeting Attended team meeting with customer	Implemented AuthorisationService Fixed web service setup issues Implemented PlayerController Added validation to web service Started client-server interaction	Continued client-server interaction, implemented login/registration from client side Attended team meeting Implemented state memory architecture (MatchList) Began implementing MatchService/ MatchController	Continued implementing MatchService/ MatchController Improved state memory architecture Implemented JSON support in service Implemented JSON support in Java test client to test endpoints (and later for bot?)	Finished implementing MatchService / MatchController Implemented StateService (except for visibility) Implemented "Game/Status" endpoint Implemented basic client lobbying	Implemented match details, joining a match, starting a new match in client Implemented starting a match in client Implemented basic rendering in client based on Tassos' render code, added players to display	Implemented game loop & communication Implemented automatic refresh for lobby list & match status Improved graphics Added functionality to leave a game, in server and client Added StateService use of VisibilityService Implemented basic use of visible tiles in client Began to implement bot project Began to merge

							client design with prototype
12th Dec	Finished merging functional prototype with client design Improved the client and fixed bugs Added ability to load different levels Fixed a lot of bugs Added endgame state Implemented graphical visibility in the client Implemented scoreboard controller/action in server Implemented scoreboard in client	Fixed a lot of bugs Implemented simultaneous start between players Implemented a very basic bot. Started working UML diagrams version 2	Continued working on and completed UML v2 diagrams. Improved doc1 and added Word styling / auto contents etc. Added Javadoc comments to Java domain library and service	Fixed doc1 merge conflict Improved, fixed and refactored doc1 with Tassos & team Filled team questionnaire w/ team Merged various documentation components into doc1 Added bot Javadoc comments Wrote maintenance guide overviews and compiled javadocs together	Wrote Javascript maintenance guide. Added acknowledgements. Fixed styling for references etc. Worked on doc2 formatting styling etc with Selin and Tassos.		

5.2 Anastasios Gemtos:

Date	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
31s Oct	Team Meeting (sprint 0)		Team Meeting				
7th Nov	Team Meeting (sprint 1)	I worked on use cases with Mattsi about player actions in the dungeon.	Team Meeting (use cases, CRC)				
14th Nov	Team Meeting (CRC, Task estimation)		Team Meeting		Worked with Mattsi on setting up the		

	(sprint 2)		(Created UML classes and discussed about system architecture)		database. We implemented some functionality based on models from UML classes		
21st Nov	Team Meeting (Completed the UML classes both in client and server) (sprint 3)		Team Meeting (Reviewed all the documentation so far)				Updated the requirement analysis and the use cases on our documents.
28th Nov	Team Meeting (worked on a document about document style and code style conventions, created draft overview of our work so far, meeting with customer) (sprint 4)	Worked with Mattsi on the skeleton of web server's framework.	Team Meeting (Worked with Xiao on database)	Completed the database functionality.	Worked on rendering the dungeon's map on web browser.	Completed the rendering of dungeon's map on web browser.	
5th Dec	Team Meeting (Meeting with Julian to discuss our progress) (sprint 5)	Worked on Character model.	Worked with Qian on Visibility on the server-side.	Completed the Visibility service to meet unit tests requirements.			Worked with Arya on Movement service.
12th Dec	Added coin collection functionality on movement service. Worked on movement on client side. Updated documentation. Team Meeting (with Julian to show him our playable demo). (sprint 6)	Worked with Selin on how the score is calculated. Worked with Arya on movement service. Worked on test cases. I did minor changes on the code to fix bugs.	I worked on adding textures to our project to look better. I worked on final version of requirement analysis.	Worked with the team for the final version of our documentation. We completed Group Questionnaire. I worked on Test Plan and on Personal Questionnair.	Deadline.		

5.3 Selin Kutlamis:

Date	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
31s Oct	First group meeting, understand the document of project		Second group meeting, Understand the document of project				Prep for Monday meet
7th Nov	Team meeting Worked with group on requirements analysis Meet Trello Brainstorming, Discussion Add meeting notes Choose leader		Attended group meeting Worked with group on use cases				
14th Nov	Group meeting Worked with group on CRCs and task estimations		Write the timescale Pair Groups are defined			Preparation for Uml diagrams	Investigation on layout design
21st Nov	Uml diagrams creation		Menu/Login/Score Layout is arranged. Worked with Pair Programmer Arya	Create customer Requirements /Reader's Guide documentation	Create customer Requirements /Reader's Guide documentation	Creation of mock up	Creation of uml diagrams and continued to document checking.
28th Nov	Weekly meeting and discussion		Sprint meeting		Investigation on login and registration with MVC Work with pair programmer Arya	Investigation on login and registration restful api	Investigation on login and registration restful api
5th Dec	Working with pair programmers Mattsi,Arya on login and registration Meeting with customer		Sprint meeting discussion Produce documentation layout		Work on documentation	Work on documentation	Work on documentation Score Interaction investigation

12th Dec	Team meeting Writing Document 1 and document2 Meeting with customer	Worked with the Pair Programmer Tasos on calculation of score/timer interaction	Team Meeting Writing Document 1 and document2	Product Timescale v3 is added. Documentation is continied.Final grammer check and control are made.	Final version of the documentation is added.		

5.4 Qian Zhou :

Date	Mondy	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
31s Oct	Team Meeting		Team Meeting				
7th Nov	Team Meeting (sprint0) – worked with group on the requirements analysis.	Worked with Xiao on the use cases about the user interaction before playing the game.	Team Meeting – worked with group on the use cases and CRC cards.				
14th Nov	Team Meeting – worked with group on the CRC cards and task estimations.		Team Meeting - worked with group on the UML classes and system architecture. Discuss about the timescale and divided the tasks for pair programming.		Worked on the design of the map interface.	Worked on the design of the map interface.	
21st Nov	Team Meeting - worked with group on the UML classes both in client and server.		Team Meeting – Worked on the documentation.		Create the Json files for different levels of the dungeon.	Create the Json files for the maps. Write the documentation about the Json files.	
28th Nov	Team Meeting - separate the tasks in next sprint, had a meeting with Julian. Worked with Xiao on determining the final design of our maps and finished the Json file.	Create and add two classes - Map and Tile to the project.	Worked with Mattis to implement the IOservice, add Json Framework to project, start Json service (including tests).		Worked with Mattis to implement Json service (parse map Json, Model map in code).		
5th Dec	Team Meeting – separate the tasks in Sprint 5.		Worked with Tasos on the visibility in server.				
12th Dec	Team Meeting – list the unfinished work both for codes and documentation, and arrange the tasks in the final week.	Worked on the documentation about layout of the map.	Worked on the documentation about the interface design.	Worked on the documentation about the user guide of use cases.	Work on the final version of the documentations.		

5.5 Xiao Fan:

Date	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
31s Oct	Team Meeting		Team Meeting				
7th Nov	Team Meeting (sprint 1)	I worked on a use case with Qian about log in and menu design in the dungeon.	Team Meeting (use cases, CRC) (I took the Meeting Minutes)				
14th Nov	Team Meeting (the rest of CRC) (sprint 2)		Team Meeting (Task estimation)				
21st Nov	Team Meeting (UML, Diagram) (sprint 3)		Team Meeting (Client UML, Diagram, documentation tasks distribution)			Learn JSON	Dungeon interface

28th Nov	Team Meeting Create JSON file about the interface of Dungeon(Qian) (sprint 4)		Creating database Build play model, Connect SQL with JAVA(Tasos), Building web service framework				
5th Dec	Researching web game named Forestry maze						
12th Dec	Team meeting (found the unfinished work and divided to every team member, have discuss with Julian)	Finding the reference about UML Diagram, Writing documentation about UML Diagram	Writing documentation about UML Diagram	Writing time scale about requirement and some other works for programming	Check the details of document and submit		

5.6 Arya Nalinkumar:

Date	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
31s Oct	First group meeting		Second meeting, gone through the documentation of project				
7th Nov		Worked on use cases for winning condition	Group meeting Gone through use cases & CRC cards				
14th Nov	Group meeting (Sprint 2) Continued work on CRC cards		Group meeting Discussed on System architecture				
21st Nov	Group meeting (Sprint 3)		Worked with Selin on template: Login, Menu & Score table section	Completed template in Score table section	Worked on documenting the system architecture		Completed System architecture document
28th Nov	Group meeting (Sprint 4) Checked the documents with team members Customer meeting		Group meeting		Discussed with Selin on login registration functionality.	Worked on login functionality- Done the Sql query to database in login & registration section.	Continued work on login & Register on client side.
5th Dec	(Sprint 5) Checked the login & register functionality with Selin & Mattsi. Discussed the modification to template.	Worked on new template changes. Completed Login/Registration, Lobby & Score table Template section	Group meeting Shown the new template and started work on How to play section & result Screen	Completed new templates work			Worked with Tassos on Player Movement Service.

	Customer Meeting						
12th Dec	(Sprint 6) Group meeting Discussed the task need to complete Customer meeting	Completed the content of user guide. Updated documentation Worked on new tutorial section. (Change in game screen)	Team meeting Checked the documents with team members. Added new screenshots on the user guide & modification on How to play section in template.	Worked on use case description and documenting source code			

Meeting Minutes

Call to order

A meeting of Team B was held at EB0.7 on 09-NOV-2016.

Attendees

Attendees included Arya, Mattsi, Anastasios, Xiao, Qian, Selin.

Members not in attendance

-

Approval of minutes

- **Sprint 0**
- **Selin was elected as team leader unanimously.**
- **Format was set for Project Diary.**
- **Requirement analysis.**
- **Trello Setup.**
- **Github Setup.**
- **Documentation of Use Cases**

Reports

- **Validated Use Cases**
- **Produced layout for login-registration screen and main menu.**
- **Create CRC cards for the first two use cases.**

Unfinished business

- **Create CRC from all Use cases**

New business

- **Create tasks from Use cases**

Announcements

-

Mattsi Jansky
Secretary

09-NOV-2016
Date of approval

Meeting Minutes

Call to order

A meeting of Team B was held at CB5.12 on 14-NOV-2016.

Attendees

Attendees included Mattsi, Anastasios, Xiao, Qian, Selin.

Members not in attendance

Members not in attendance included Arya.

Approval of minutes

- **Layout for login-registration screen and main menu uploaded on Trello.**
- **CRC cards for the first two use cases uploaded on Trello.**

Reports

- **Completed CRC cards for all use cases.**
- **The project was split into tasks.**
- **Time estimation on each task was set.**

Unfinished business

Creating uses cases with server and client side

New business

- **UML design.**

Announcements

-

Anastasios Gemtos
Secretary

14-NOV-2016
Date of approval

Meeting Minutes

Call to order

A meeting of Team B was held at EB0.7 on 16-NOV-2016.

Attendees

Attendees included Mattsi, Anastasios, Xiao, Qian, Selin, Arya.

Members not in attendance

-

Approval of minutes

- **CRC cards were added to project's documentation.**
- **Few changes on time estimation were made and we added tasks on Trello.**

Reports

- **System architecture was discussed (3-tier architecture).**
- **Created UML classes for Server-side.**

Unfinished business

- **UML classes for client-side**

New business

- **Layouts for game screen and maps**
- **Database set up**

Announcements

-

Anastasios Gemtos
Secretary

16-NOV-2016
Date of approval

Meeting Minutes

Call to order

A meeting of Team B was held at CB5.12 on 21-NOV-2016.

Attendees

Attendees included Mattsi, Anastasios, Xiao, Qian, Selin, Arya.

Members not in attendance

-

Approval of minutes

- **UML tool was used to create UML diagrams for server.**

Reports

- **Created UML classes for Client-side and made a few changes on Server-side UMLs.**

Unfinished business

-

New business

- **Review all the documents created so far.**
- **Start coding.**

Announcements

-

Anastasios Gemtos
Secretary

21-NOV-2016
Date of approval

Meeting Minutes

Call to order

A meeting of Team B was held at CB5.8 on 23-NOV-2016.

Attendees

Attendees included Mattsi, Anastasios, Xiao, Qian, Selin, Arya.

Members not in attendance

Approval of minutes

- **UML diagrams were added into our project's documentation.**

Reports

- **Reviewed all the documents about requirement analysis, Use Cases, CRC cards, system architecture and design, UML diagrams and test planning.**

Unfinished business

- **Need to integrate all documents into a single one.**

New business

- **Start coding.**

Announcements

- **The code development of the project begins next Monday**
-

Anastasios Gemtos
Secretary

23-NOV-2016
Date of approval

Meeting Minutes

Call to order

A meeting of Team B was held at CB5.12 on 28-NOV-2016.

Attendees

Attendees included Mattsi, Anastasios, Xiao, Qian, Selin, Arya.

Members not in attendance

-

Approval of minutes

- **Changes were made to all documents to be clearer and less ambiguous.**

Reports

- **Created a single document that explains our system's architecture and design based on requirement analysis.**
- **Created tasks on Trello (Sprint 4).**
- **Created a document for style conventions both for documentation and coding.**
- **Meeting with Julian Padget to demonstrate our progress so far.**

Unfinished business

- **Need to split the Use Cases into user stories and design use cases.**
- **Need to restructure our document.**

New business

-

Announcements

-

Anastasios Gemtos
Secretary

28-NOV-2016
Date of approval

Meeting Minutes

Call to order

A meeting of Team B was held at EB0.7 on 30-NOV-2016.

Attendees

Attendees included Mattsi, Anastasios, Xiao, Qian, Selin, Arya.

Members not in attendance

Approval of minutes

-

Reports

- **We worked in pairs to complete database functionality, login/registration functionality and map parsing by the server.**

Unfinished business

- **Need to split the Use Cases into user stories and design use cases.**
- **Need to restructure our document.**

New business

-

Announcements

-

Anastasios Gemtos
Secretary

30-NOV-2016
Date of approval

Meeting Minutes

Call to order

A meeting of Team B was held at CB5.12 on 5-DEC-2016

Attendees

Attendees included Mattsi, Anastasios, Xiao, Qian, Selin, Arya.

Members not in attendance

Xiao

Approval of minutes

-

Reports

- **We discussed about remaining functionality and tasks are assigned. Meeting with the customer is happened.**

Unfinished business

- **Score Implementation, bot functionality**
- **Need to restructure our document.**
- **Exit and leave condition**
- **User guide and tutorial**

New business

-

Announcements

-

Selin Kutlamis
Secretary

05-DEC-2016
Date of approval

Meeting Minutes

Call to order

A meeting of Team B was held at EB0.7 on 7-DEC-2016.

Attendees

Attendees included Mattsi, Anastasios, Xiao, Qian, Selin, Arya.

Members not in attendance

Xiao

Approval of minutes

-

Reports

- **We worked in pairs to complete movement functionality, visibility functionality and documentation.**

Unfinished business

- **Score Implementation, bot functionality**
- **Need to restructure our document.**
- **Exit and leave condition**
- **User guide and tutorial**

New business

-

Announcements

-

Selin KUTLAMIS
Secretary

07-DEC-2016
Date of approval

Meeting Minutes

Call to order

A meeting of Team B was held at CB.5.12 on 12-DEC-2016.

Attendees

Attendees included Mattsi, Anastasios, Xiao, Qian, Selin, Arya.

Members not in attendance

-

Approval of minutes

-

Reports

- **Discussion for documentation and improvement in the bug fixes.**

Unfinished business

- **Score interaction and exit condition**

New business

-

Announcements

-

Selin KUTLAMIS
Secretary

12-DEC-2016
Date of approval

Meeting Minutes

Call to order

A meeting of Team B was held at EB0.7 on 14-DEC-2016.

Attendees

Attendees included Mattsi, Anastasios, Xiao, Qian, Selin, Arya.

Members not in attendance

Approval of minutes

-

Reports

- **Continued with documentation.**

Unfinished business

-

New business

-

Announcements

-

Selin KUTLAMIS
Secretary

14-DEC-2016
Date of approval

References

Croll, A., 2010. *Namespacing in JavaScript* [Online]. San Francisco, CA, U.S.: Javascript Web Blog. Available from: https://javascriptweblog.wordpress.com/2010/12/07/namespacing-in-javascript/ [Accessed 15/12/2016]

ISTQB Exam Certification. (n.d.). *What is Component testing?* [Online]. ?: Bruseels, Belgium: ITSQB. Available from: http://istqbexamcertification.com/what-is-component-testing/ [Available 26/11/2016]

Kaner, C., 2006. *Exploratory Testing from Florida Institute of Technology, Quality Assurance Institute Worldwide Annual Software Testing Conference, Orlando, FL* [Online]. *Orlando, FL*: Kaner.com. Available from: http://www.kaner.com/pdfs/ETatQAI.pdf [Accessed 26/11/2016]