

# Dungeon of Doom

CM50109 Coursework 2

Document One

## [TEAM MEMBERS]

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

# 1. Contents

1. Contents	1
2. Project Overview	4
3. Reader's Guide	4
3.1 Customer Requirements	4
3.1.1. Types of Reader	4
3.1.2. Technical Background Required	4
3.1.3. Scope of the Project	4
3.2 Requirement Analysis	4
4. Project Timescale v1	5
5. Requirement Analysis	5
5.1 Meeting Notes with Customer	5
5.1.1. Target and Budget of the Project	5
5.1.2. General Constraints	5
5.1.3. Submission and Deadline	5
5.2 Initial Functional Requirements	5
5.3 Development process plan	6
6. User Stories	6
6.1 User Story 1 – Log in	6
6.2 User Story 2 – Registration	7
6.2.1. Test Cases	7
6.3 User Story 3 – Main Menu	7
6.3.1. Test Cases	8
6.4 User Story 4 – World Creation	8
6.4.1. Test Cases	8
6.5 User Story 5 – Movement	8
6.5.1. Test Cases	9
6.6 User Story 6 – Coins	9
6.6.1. Test Cases	9
6.7 User Story 7 – Map visibility	9
6.7.1. Test Cases	9
6.8 User Story 8 – Exit	9
6.8.1. Test Cases	10
6.9 User Story 9 – Winning Condition	10
6.9.1. Test Cases	10
6.10 Process of the User Story Creation	10
7. CRC Cards	12

8.	Process of Creating CRC Cards	13
9.	Uses Cases v2	13
9.1	Use Case Reading Guide	13
9.2	Use Case 1 – Login with System and Client Relationship	14
9.3	Use Case 2 – Registration with Server and Client Relationship	15
9.4	Use Case 3 – Main Menu with Server and Client Relationship	16
9.4.1.	Single/Multiplayer	16
9.4.2.	Score	17
9.4.3.	Tutorial	17
9.5	Use Case 4 – World Creation with Server and Client Relationship	17
9.6	Use Case 5 – Movement with Server and Client Relationship	18
9.7	Use Case 6 – Coins with System and Client Relationship	19
9.8	Use Case 7 – Map visibility with System and Client Relationship	21
9.9	Use Case 8 – Exit with System and Client Relationship	22
9.10	Use Case 9 – Winning Condition with System and Client Relationship	22
10.	Product Timescale v2	24
11.	UML Diagrams v1	26
11.1	Client Side	28
11.2	Server-side Controllers	29
11.3	4.3 Server-Side Models & Database	30
12.	Defining Non-Functional Requirements	30
12.1	Questions	30
12.2	Non-Functional Requirements	31
12.3	System Architecture	31
12.3.1.	Architectural Views	31
12.3.2.	Deployment View	31
12.3.3.	Architectural Design Patterns	32
12.3.4.	Architectural Style	32
13.	Layout Design	34
13.1	Layout Mock-up Design	34
13.2	Level Design	35
13.3	Implementation of Design and Interface	35
14.	Functional requirements v2	36
15.	Product Timescale v3	37
16.	Use Case V3- Multiplayer Use Case	38
17.	UML v2	39
17.1	Domain package	39
17.2	Service package	41

17.3	Bot package	42
18.	User Guide	42
18.1	System Overview	42
18.2	Login and Register	43
18.3	Lobby	44
18.4	How to play	44
18.5	Game Screen	46
18.6	Score Table	48
19.	Acknowledgements	48
20.	References	50

## **2. Project Overview**

The project is about creating a multi-player online game with agile development practices. The goal is to complete the game until 16 December 2016 with 3-tier architecture, MVC design pattern and Test-Driven Development while working in pairs. During the first meeting, we chose a leader to coordinate the group. Selin was elected as a leader unanimously. Team worked together in requirements analysis. For coding, the group was divided into three teams. Experienced programmers, Mattsi and Tasos, lead the team in technical area. They work both in front and in back end. Selin and Arya work on front-end. Qian and Xiaoxiao had an experience on graphical design they were responsible for the map design. After sprint 5, pair groups were rearranged to one experienced programmer and one inexperienced programmer. By doing so, experienced programmer could share his/her experience with other team members and inexperienced programmer could check the code if there was any mistake. Sprint and customer meetings happened weekly.

## **3. Reader's Guide**

### **3.1 Customer Requirements**

This document contains the final version of requirements for Dungeon of Doom project. These requirements have been derived from the coursework specifications and meetings with the customer.

#### **3.1.1. Types of Reader**

Our readers are Senior Lecturer Dr. Julian Padget as a customer and Phd student Charlie Ann Page.

#### **3.1.2. Technical Background Required**

To understand code's structure and development phases, readers should have knowledge on computer science.

#### **3.1.3. Scope of the Project**

To analyse, specify, design and implement a multi-player on-line game.

### **3.2 Requirement Analysis**

To determine requirements, a team was gathered. Scrum meetings are done for weekly. For agile development phases, Trello is installed in every team member's computer. GitHub accounts are created for each team member.

#### 4. Project Timescale v1

Table 1 represents the initial timescale for the requirement analysis, Use cases, CRC cards and UML Diagram.

Sprint	Effort points	Sprint 0	Sprint 1	Sprint 2	Sprint 3
Date	(1 = 0.5 day)	31.10.2016	7.11.2016	14.11.2016	21.11.2016
Milestones		Start			Ready for Development
Requirement analysis	4	x			
Use case	8		x		
CRC card	4			x	
UML Diagram	6				x
Total Sprint Effort	22				

Table 1. Project Timescale v1

#### 5. Requirement Analysis

##### 5.1 Meeting Notes with Customer

###### 5.1.1. Target and Budget of the Project

According to customer meeting notes; the target group of the product is all ages. It is an online game and it does not require any technical skills, training, or education. There is no specific budget for the project according to the customer.

###### 5.1.2. General Constraints

The customer didn't specify any specific programming language or technology. Subsequently, the team decided to use Java for the back-end and JavaScript for the front-end. For map design, we decided to use JSON files to represent the data. Additionally, we decided to use MySQL to store persistent user data. The game will be displayed on a web browser with the use of HTML5&CSS3. IntelliJ IDE will be used as a development tool.

###### 5.1.3. Submission and Deadline

Finished product will be published on the Internet. No need for specific hardware to play the game. Project should be submitted by 16 December 2016.

##### 5.2 Initial Functional Requirements

**R1.** The user must log in to play the game.

R1.1. If the user is not registered to the game, he can register by giving a username, a password.

**R2.** The user chooses options from a menu (single, multiplayer, score or tutorial)

- R2.1. If the user selects single player game, he enters the level selection screen.
- R2.2. If the user selects multiplayer game, he enters the level selection screen.
- R2.3. If the user selects score, he sees the scoreboards.
- R2.4. If the user selects tutorial, the player sees the instructions of the game.
- R2.5. If the user can select exit or closes the tab.

**R3.** The world is loaded on the server's memory.

**R4.** The user can see a part of the dungeon (explored tiles), other players (human or bots), gold coins, passages and exit.

R4.1. A dungeon is a collection of rooms that are connected with passages. A room may have gold coins.

R4.2. A dungeon can be arbitrary size.

R4.3. Dungeon must contain the minimum gold coins so the player can win.

**R5.** The user interacts with the dungeon:

R5.1. Indicating which way to move (UP, DOWN, LEFT, RIGHT)

R5.2. Picking up gold.

R5.3. The user can move around to reveal the room.

R5.4. Leaving the game.

**R6.** Winning Condition:

R6.1. If the user collects all the gold coins, which are needed to win, and find the exit, the user wins.

R6.2. Else if another user wins first or the user gives up, the user loses.

**R7.** After winning condition, the user is prompted to result screen.

### 5.3 Development process plan

- During our first meetings, we started thinking about user stories, CRC cards and use cases, all of which were derived from our requirement analysis.
- We used tools such as Trello and GitHub to organize our project.
- We were keeping track of our progress by updating our project diary.
- We kept track of our entire project meeting minutes.
- We created an initial project timescale based on these requirements.

## 6. User Stories

### 6.1 User Story 1 – Log in

As a user, I want to log in to main menu

- The user can view the login page.
- The user can enter a username and password for logging in to the game.
- The user can view fail screen if an error is occurred.

Actor Actions
1. Open the game page.
2. Type the username and password.
3. Press "OK" button.

Table 2. Actor Actions in Login

System Actions
1. System displays the login page

- |  |
|--|
| 2. System displays the error message if there is an error. |
|--|

Table 3. System Actions in Login

## 6.2 User Story 2 – Registration

As a user, I want to register to play the game.

- The user can view the registration page.
- The user can enter username and password to create an account.
- The user can view error messages if s(he) types wrong username/password.

Actor Actions
1. Open the registration page.
2. Type the username and password.
3. Press “OK” button.

Table 4. Actor Actions in Registration

System Actions
1. System displays the registration page.
2. System displays the username and hides the password.
3. System displays the error message if credentials are wrong.

Table 5. System Actions in Registration

### 6.2.1. Test Cases

- Test an email address as username (pass)
- Test empty fields as username and password (fail)
- Test long username and passwords (up to 255 chars) (pass)

## 6.3 User Story 3 – Main Menu

As a user, I want to see the main menu to choose game type (single or multiplayer).

- The user can enter the main menu.
- The user can select between two options single and multiplayer for playing the game.
- The user can choose level for a match.
- The user can view dungeon after selection of level.
- The user can start playing the game after dungeon map is loaded.
- The user can select score for seeing the score table.
- The user can select tutorial to see instructions on how to play the game.
- The user can exit or log out from the system by returning to login page.



Actor Actions
1. Begins when the user choose an action from the menu
2. The user is prompted to appropriate screen.

**Table 6. Actor Actions in Main Menu**

System Actions
1. System displays the main menu.
2. System displays the level selection.
3. System displays the score table or tutorial or exit options.

**Table 7. System Actions in Main Menu**

#### 6.3.1. Test Cases

- Test if main menu pops after a successful log in (pass).

### 6.4 User Story 4 – World Creation

As a user, I want to see the world which consists of rooms, passages and golds.

- The user can view the dungeon map which consists of rooms.
- The user can view the rooms. Passages connect the rooms.
- The user can view the gold coins in the rooms.
- The user can see a message with the minimum number of gold coins he needs to collect.

Actor Actions
1. Begins when the user selects a new match.

**Table 8. Actor Actions in the World**

System Actions
1. System displays the dungeon map.
2. System displays the rooms, passages and coins.
3. System displays a message with minimum number of gold coins needed to win the game.

**Table 9. System Actions in the World**

#### 6.4.1. Test Cases

- Test if numbers of coins are enough to finish the game (pass).

### 6.5 User Story 5 – Movement

As a user, I want to move around the dungeon to collect coins.

- The user can move between rooms through passages.
- The user can view the walls which block the way.

Actor Actions
1. Begins when the user presses movement key.

**Table 10. Actor Actions in Movement**

System Actions
1. System displays the movement of the player in the dungeon.

**Table 11. System Actions in Movement**

### 6.5.1. Test Cases

- Test if the player can move to a wall (fail)
- Test if the player can move to exit without having the minimum required coins (fail)
- Test if the player can move to exit with having the minimum required coins (pass)
- Test if the player can move to a coin and collect it (pass)

## 6.6 User Story 6 – Coins

As a user, I want to pick up gold coins to win the game.

- The user can move to a tile and collect the coins in the room.

Actor Actions
1. The user chooses to move to a gold coin.

Table 12. Actor Actions in Coins

System Actions
1. System displays the coins in the rooms.

Table 13. System Actions in Coins

### 6.6.1. Test Cases

- Test if score is updated when a coin is collected (pass)

## 6.7 User Story 7 – Map visibility

As a user, I want to see the map.

- The user can reveal the map as they move around the dungeon.

System Actions
1. System displays the tiles of the map.

Table 14. System Actions in Map

### 6.7.1. Test Cases

- Test if other players or bots are visible when they are close (pass)
- Test if coins are visible when player is close to it (pass)
- Test if exit is visible when player is close to it (pass)

## 6.8 User Story 8 – Exit

As a user, I want to leave the game.

- The user can view the exit button in the game interface.
- The user can return to the main menu by pressing the exit button.

Actor Actions
1. The user presses the exit button.

**Table 15. Actor Actions in Exit**

System Actions
1. System displays the exit option.

**Table 16. System Actions in Exit**

#### **6.8.1. Test Cases**

- Test if both players exit the game when a winning condition is triggered (pass)

### **6.9 User Story 9 – Winning Condition**

As a user, I want to win by getting the minimum amount of coins.

- The user can view the winning status on the screen.
- The user can return to match selection to pick a different level.
- The user can win the game by collecting the minimum amount of coins and pass through the exit.
- The user can view the scoreboard and unlock a new level in the result screen with details about coins that the user collected.
- The user can view “You win” message in the result screen if they exit first.
- The user can view “You lose” message in the result screen if another user exits first.

Actor Actions
1. The user won the game.
2. The user lose the game.

**Table 17. Actor Actions in Winning Condition**

System Actions
1. System displays the result of the game.

**Table 18. System Actions in Winning Condition**

#### **6.9.1. Test Cases**

- Test if both players return to result screen when a winning condition is triggered (pass)
- Test if a winning condition is triggered when a player disconnects (pass)
- Test if a winning condition is triggered when a player exits the map with the minimum amount of gold coins (pass)

### **6.10 Process of the User Story Creation**

- User stories are great for understanding of what the client’s requirements are. Development teams can inquire to the clients about specifications on the requirements. It should be written in a non-technical format. By creating user stories, target goals can be reached more efficiently (Visual Paradigm, 2016). According to Cockburn (2000, p. 42) regarding user stories, an actor begins an interaction with a particular goal in mind in the specified system. Scenarios have a result with respect to each goal.
- For Dungeon of Doom game, user stories were discussed in the first sprint meetings.
- Use stories were written on the whiteboard.

- The user stories were derived from the requirement analysis.
  - As an example, in our first discussion, we decided that the user should see the login and registration layout before getting into the game. For making the process simpler, only username and password are required. If the user is not registered, then they should be redirected to the registration page. We decided to ask only for a username and password from the user.
- All the team members contributed to separate user stories.

## 7. CRC Cards

Responsibilities	Collaboration
<b>Class: Player</b>	
Username Password Level Type	Character Score
<b>Class: Score Table</b>	
Score Calculate Max ID (time-stamp)	Player
<b>Class: Level</b>	
Load from file	Dungeon Match
<b>Class: Dungeon</b>	
Size Name Number of coins	Character File Level
<b>Class: Character</b>	
Position Gold coins collected Movement Track collected coins	Player Score
<b>Class: Tile</b>	
Type Visibility	Dungeon
<b>Class: Match</b>	
Check Victory Condition	Score

## 8. Process of Creating CRC Cards

- CRC cards are useful in Object Oriented Programming. Cards represent classes, their responsibilities and relationships. There is no universal syntax for CRC cards (OpenLoop Technologies). According to Alshehri and Benedicenti from University of Regina creating CRC cards “can be a way of measuring the quality of the software and ensuring the simplicity of the design.” (2013).
- After development of the use stories, in the sprint 1 meeting, the team built CRC cards.
- Each CRC card is derived from one of the user stories.
- Classes are created with the idea of "what happens when..." like “What is going to be happen when all the coins are collected? Should new level will unlock?”
- The team built the CRC cards during a brainstorming session in a sprint meeting.
- For example,
  - The team decided that the Player class should have four different responsibilities: Username, Password, Level, and Type (human or bot). Each player should have a relationship with character and the score.
  - We decided that the Level class should collaborate with Match and Dungeon.
- By using CRC cards we identified the key components of our system and their responsibilities and relationships.

## 9. Uses Cases v2

### 9.1 Use Case Reading Guide

We defined our use cases in a specified format, which start by an identifier followed by a step number. For instance, UC1-2 denotes the second property of use case one which is Author. The use cases are comprised of following properties(where X is the number of Use case):

- Title: It represents the use case name. (UCX-1)
- Author: The team members who created the use case. (UCX-2)
- Date: Creation date (UCX-3)
- Purpose: A brief description of the goal (UCX-4)
- Overview: An outline of processes, which include the interaction between actor and system.(UCX-5)
- Cross-references: It relates to the associated requirements by requirement IDs that are defined in section 3. For example, cross-reference R1.1 represents the functional requirement: ‘R1.1 If the player is not registered to the game, he can register by giving a username, a password. (UCX-6)
- Actors: An external entity which interact with the system (UCX-7)
- Pre-conditions: The condition that needs to be true before execution. (UCX-8)
- Post-conditions: The condition that the system should hold after execution.(UCX-9)

- Alternative flow of events: An unexpected sequence of flow, which might interrupt the normal flow. (UCX-10)
- Exceptional flow of events: The unusual event can occur (UCX-11)

## 9.2 Use Case 1 – Login with System and Client Relationship

UC1-1: Use Case: Log In

UC1-2: Author: QZ, XF

UC1-3: Date: Sprint 3

UC1-4: Purpose: Log in to the play the game

UC1-5: Overview: System requests username and password for login. System validates username and password (if the username and password already exists in the player database) then player enters the Main Menu Screen.

UC1-6: Cross References: R1

UC1-7: Actors: Player

UC1-8: Pre-Conditions:

UC1-Pre-1: The user must be at the log in page.

UC1-9: Post-Conditions:

UC1-Post-1: The user must be prompted to main menu after a successful log in.

Actor Actions	Client System Actions	Server System Actions
1. Open the game page.		
2. Input the username and password.		
3. Press “OK” button.		
	4. Sends the username and password to server.	
		5. Check if the username already exists in the membership database and validates the password.
		6. Sends response.
	7. Receives and parses the response.	
	8. Jump to the main menu interface.	

UC1-10: Alternative flow of events:

Step 5: Username and password is not in the player database. Display an error message, and ask player to reenter their credentials.

UC1-11: Exceptional flow of events:

Steps 4, 6, 7: If the connection with the server is not established return an error message.

### 9.3 Use Case 2 – Registration with Server and Client Relationship

UC2-1: Registration

UC2-2: Author: AG

UC2-3: Date: Sprint 3

UC2-4: Purpose: Player registers to the game

UC2-5: Overview: The user fills a form with his desired username and desired password. If the username and password does not exist on the system's database, then the system saves these details and returns to log in screen.

**Alternative 1:** if username and password exists in database, then an appropriate message is returned to the user and the user must pick a different username or password.

**Alternative 2:** if the email does not contain "@" and "." characters the client must give an appropriate message.

UC2-6: Cross Reference: Functional Requirements R1.1, R1

UC2-7: Actors: Player

UC2-8: Pre-Conditions:

UC2-Pre-1: The registration web page must be loaded.

UC2-Pre-2: The player must not have an account.

UC2-9: Post-Conditions:

UC2-Post-1: The player returns to log in screen.

UC2-Post-2: The player's information is stored into the server's database.

Actor Actions	Client System Actions	Server System Actions
1. Open the registration page.		
2. Input the username, the email and the password.		
3. Press "OK" button.		
	4. Sends the username, and password to server.	
		5. Stores the username and the password to system's database.
		6. Sends response.
	7. Receives and parses the response.	
	8. Jump to the log in page.	



UC2-9: Alternative flow of events:

Step 5: Username or password already exists on the system's database. Appropriate message is returned to the player so the player can pick different username or password.

UC2-9: Exceptional flow of events:

Step 4, 6, and 7: If the connection with the server is not established return an error message.

## 9.4 Use Case 3 – Main Menu with Server and Client Relationship

UC3-1 Use Case: Player Choose Menu

UC3-2 Author: XF, QZ

UC3-3: Date: Sprint 3

UC3-4: Purpose: The user chooses options from the main menu.

UC3-5: Overview: The player must select a match level he wants to play in, the client sends a request to server and the server sends dungeon data to the client. The user is prompted into the dungeon map and starts playing the game.

**Alternative 1:** if the player selects score, the game client sends a request to the server to retrieve the score table from the database. The server sends the data back to client.

**Alternative 2:** if the player selects the exit button, the player logs out of the system and returns to log in screen.

UC3-6: Cross References: [R2](#), [R2.1](#), [R2.2](#), [R2.3](#), [R2.4](#), [R2.5](#)

UC3-7: Actors: Player

UC3-8: Pre-Condition:

UC3-Pre-1: The player must be in menu (i.e. not the dungeon).

UC3-Pre-2: The player must already be logged in.

UC3-9: Post Condition:

UC3-Post-1: The player is sent to appropriate web page based on his/her action.

### 9.4.1. Single/Multiplayer

Actor Actions	Client System Actions	Server System Actions
1. Begins when player click single player of multiplayer button	2. Sends button request to the server	3. The server initiates the player's session.

5. The user selects a match to play in.
4. Jumps to level selection web page.
6. The client sends a request to the server
7. The server responds with dungeon data.
8. Client receives the response and creates a graphical representation of the data.

#### 9.4.2. Score

##### Actor Actions

1. Begins when player click score button

##### Client System Actions

2. Sends button request to server
5. The user is prompted to score table web page and shows the data.

##### Server System Actions

3. Retrieve the scoreboard from database.
4. Send response

#### 9.4.3. Tutorial

##### Actor Actions

1. Begins when player click Tutorial button

##### Client System Actions

2. The user is prompted to tutorial page.

##### Server System Actions

UC3-9: Alternative flow of events:

UC3-10: Exception flow of events:

In steps where the client sends a request or the server sends a response, if the client does not receive any response in time, appropriate message should be displayed.

## 9.5 Use Case 4 – World Creation with Server and Client Relationship

UC4-1 Use Case: World Creation

UC4-2 Author: SK

UC4-3: Date: Sprint 3

UC4-4: Purpose: To create map, dungeon, coins, passages

UC4-5: Overview: The client sends a request to server to retrieve graphical representation of the dungeon.

UC4-6: Cross Reference: R3, R4, R4.1, R4.2, R4.3

UC4-7: Actors: Player

UC4-8: Pre-Condition:

UC4-Pre-1: The player must select a match.

UC4-Pre-2: The player must already log on.

UC4-9: Post Condition:

UC4-Post-1: Player enters the game map and sees a graphical representation of the dungeon.

#### **Actor Actions**

- 1. Begins when player selects a match.**

#### **Client System Actions**

- 2. Sends button request to server**

#### **Server System Actions**

- 3. Dungeon is created with rooms, passages and coins. Server saves this information on its memory.**
- 4. Sends response with visible area by the player**

- 5. Receives and parses response**
- 6. The client draws the graphical representation of the dungeon.**

### **9.6 Use Case 5 – Movement with Server and Client Relationship**

UC5-1 Use Case: Player Moves Character

UC5-2 Author: AG, MJ

UC5-3: Sprint3

UC5-4: Purpose: Move the player in one of the four directions

UC5-5: Overview: The player character's location is a particular point on the map. The Player presses one of the keys W, A, S or D and arrow keys. These keys map to up, left, down, right respectively. The game client sends a request to the server with the details on the action. The server validates this input and decides whether or not the player character can move in that direction. If the character can be moved the server updates the character's position in-memory. The server responds with the current location of the character. The client updates the dungeon graphical representation.

**Alternative 1:** A wall is blocking the direction that the player wishes to move their character in. The server does not update the player's position and responds with the current player location the same as it was.

UC5-6: Cross References: R5, R5.1

UC5-7: Actors: Player

UC5-8: Pre-Condition:

UC5-Pre-1: The player must be in a dungeon (i.e. not the menu).

UC5-Pre-2: The map has been loaded.

UC5-9: Post Condition:

UC5-Post-1: The player's position is the player's previous position moved one unit in the chosen direction.

UC5-Post-2: The client's graphical representation has updated.

Actor Actions	Client System Actions	Server System Actions
1. Begins when player presses movement key		
	2. Sends movement request to server	
		3. Checks whether or not the player can move in that direction.
		4. Sends response
	5. Receives and parses response	
	6. Updates dungeon graphical representation	

UC5-10: Alternative flow of events:

Step 3: The movement is illegal. Server doesn't update player location, responds with player in the same location.

UC5-11: Exception flow of events:

Steps 2, 4, 5: The request or response network packets are dropped or corrupted. Sender sends a request for the current state of the system. If that request fails, the client displays an appropriate message regarding network connectivity problems to the player.

## **9.7 Use Case 6 – Coins with System and Client Relationship**

UC6-1: Use Case: Picking up Gold coins

UC6-2: Authors: AG, MJ

UC6-3: Date: Sprint 3

UC6-4: Purpose: Moves to gold coin to collect it.

UC6-5: Overview: The player moves to a tile with a gold coin on it. The server responds that the gold coin is no longer in its previous position, it increments player's gold coin collection and moves the player to the location of the coin. The client updates the dungeon graphical representation.

UC6-6: Cross References: R5, R5.2

UC6-7: Actors: Player

UC6-8: Pre-condition:

UC-6-Pre-1: The player must be in a dungeon (i.e. not in the menu)

UC-6-Pre-2: The player should be one unit away from the gold coin.

UC6-9: Post-condition:

UC-6-Post-1: The player's gold coin collection is incremented.

UC-6-Post-2: The player moves to the updated location.

UC-6-Post-3: The client updates the dungeon graphical representation.

#### **Actor Actions**

- 1. Player begins to move to a gold coin.**

#### **Client System Actions**

- 2. The client sends a request to the server.**
- 8. Receives and parses the response**
- 9. Updates the graphical representation.**

#### **Server System Actions**

- 3. The server receives the request.**
- 4. The server validates the action.**
- 5. The server increments player's gold coin collection.**
- 6. The server updates gold coin location.**
- 7. Sends response**

## 9.8 Use Case 7 – Map visibility with System and Client Relationship

UC7-1 Use Case: Player movement reveals the map

UC7-2 Author: AG, MJ

UC7-3: Date: Sprint 3

UC7-4: Purpose: To discover the layout the dungeon

UC7-5: Overview: The player character is moving from one tile to another. The server decides which tiles are visible to the player character. The server response includes the current state of the tiles now visible to the player character. The client adds these tile states to its memory. The client remains aware of previously discovered tiles but may not be aware of their current state, i.e. whether another player has moved their character to that location. The client updates the graphical representation including the newly visible tiles.

UC7-6: Cross References: R5, R5.3

UC7-7: Actors: Player

UC7-8: Pre-Condition:

UC7-Pre-1: The player must be in a dungeon (i.e. not the menu)

UC7-Pre-2: The player is in a state of moving from one tile to the next

UC7-9: Post Condition:

UC7-Post-1: The client updates its memory with new tile states.

UC7-Post-2: The client now displays additional tiles that may not have been previously visible.

Actor Actions	Client System Actions	Server System Actions
		1. Begins when the server interprets a move command
		2. Decides which tiles are visible to the character
		3. Responds to move request, including the current state of tiles now visible to the player character
	4. Receives and parses the response	
	5. Adds the updated and/or new tiles to its memory	
	6. Updates the graphical representation	

## 9.9 Use Case 8 – Exit with System and Client Relationship

UC8-1: Use Case: Leaving the Game

UC8-2: Authors: AG, MJ

UC8-3: Date: Sprint 3

UC8-4: Purpose: To exit the game.

UC8-5: Overview: The player presses a button to return to the main menu. The client sends a request to server to terminate the session. The server removes the player's character.

UC8-6: Cross References: R5, R5.4

UC8-7: Actors: Player

UC8-8: Pre-condition:

UC-8-Pre-1: The player must be in a dungeon (i.e. not in the menu)

UC-8-Pre-2: The player presses the leave button

UC8-9: Post-condition:

UC-8-Post-1: The player returns to main menu

UC-8-Post-2: The server removes the player from the game.

Actor Actions	Client System Actions	Server System Actions
1. The player presses the leave button.	2. Sends request to the server.	3. Receives the request.
		4. Removes the player's character.
	5. Returns to main menu.	

UC8-11: Exception flow of events:

Step1: The player closes the browser to exit the game. The server notices that the client has not sent any request within a time limit and removes the player's character.

## 9.10 Use Case 9 – Winning Condition with System and Client Relationship

UC9-1: Use Case: Winning condition

UC9-2: Authors: AN

UC9-3: Date: Sprint3

UC9-4: Purpose: To show the winning status and Main map with levels.

UC9-5: Overview: The player won the game by passing through exit. The client sends the request to update the score. Server starts processing request. Server updates the results in scoreboard, and sends a response back to client

to show the result screen with details which include time, coins or other entities player collected. Client displays the main map to player, which shows the levels with an option to return to main menu.

UC9-6: Cross References: R6, R6.1, R6.2, R7

UC9-7: Actors: Player

<b>Actor Actions</b>	<b>Client System Actions</b>	<b>Server System Actions</b>
<b>1. The player won the game.</b>	<b>2. Sends the updated result to the server.</b>	<b>3. Receives the request.</b>
		<b>4. Update the scoreboard.</b>
		<b>5. Sends the result screen.</b>
	<b>6. Shows the result screen</b>	
	<b>7. Shows the main map with levels</b>	
	<b>8. Returns to main menu.</b>	

UC9-11: Exception flow of events:

Step1: If player loses the connection to the network after winning the game, client waits for a specified amount of time. If player comes back, client sends request to server and steps 2-8 will carry on. If the server hasn't got any request within the time limit, then server will remove the player's character.



## 10. Product Timescale v2

Sprint	Effort points	Sprint 0	Sprint 1	Sprint 2	Sprint 3	Sprint 4	Sprint 5	Sprint 6
<b>Date</b>	<b>1 = 0.5 day</b>	<b>31/10/16</b>	<b>7/11/16</b>	<b>14/11/16</b>	<b>21/11/16</b>	<b>28/11/16</b>	<b>5/12/16</b>	<b>12/12/16</b>
<b>Milestones</b>		<b>Start</b>			<b>Start Coding</b>	<b>Basic Functionality</b>		
<b>Login Functionality</b>	<b>3</b>					<b>X</b>		
<b>Registration Functionality</b>	<b>3</b>					<b>X</b>		
<b>Login/Registration Layout</b>	<b>1</b>				<b>X</b>			
<b>Menu Design</b>	<b>2</b>				<b>X</b>			
<b>Menu Tutorial</b>	<b>2</b>				<b>X</b>			<b>X</b>
<b>Score Layout</b>	<b>2</b>				<b>X</b>			
<b>Leaving (exit by menu)</b>	<b>4</b>							<b>X</b>
<b>Timeout (exit by absence)</b>	<b>2</b>							<b>X</b>
<b>Design map JSON</b>	<b>4</b>				<b>X</b>			
<b>Parse map JSON</b>	<b>4</b>					<b>X</b>		
<b>Model map in code</b>	<b>2</b>					<b>X</b>		
<b>Render the map in client</b>	<b>3</b>					<b>X</b>		
<b>Create player model in server</b>	<b>2</b>					<b>X</b>		
<b>Communication Between server/client</b>	<b>4</b>						<b>X</b>	
<b>Visibility-server</b>	<b>4</b>						<b>X</b>	
<b>Visibility-client</b>	<b>4</b>							<b>X</b>
<b>Movement</b>	<b>2</b>						<b>X</b>	
<b>Score- model</b>	<b>4</b>						<b>X</b>	
<b>Score-calculate</b>	<b>1</b>						<b>X</b>	
<b>Score- add coin</b>	<b>2</b>							<b>X</b>

<b>Setup database</b>	<b>2</b>				<b>X</b>			
<b>Add winning condition</b>	<b>4</b>							<b>X</b>
<b>Total Sprint Effort</b>	<b>61</b>				<b>12</b>	<b>16</b>	<b>16</b>	<b>17</b>

Table 19. Project Timescale v2

## 11. UML Diagrams v1

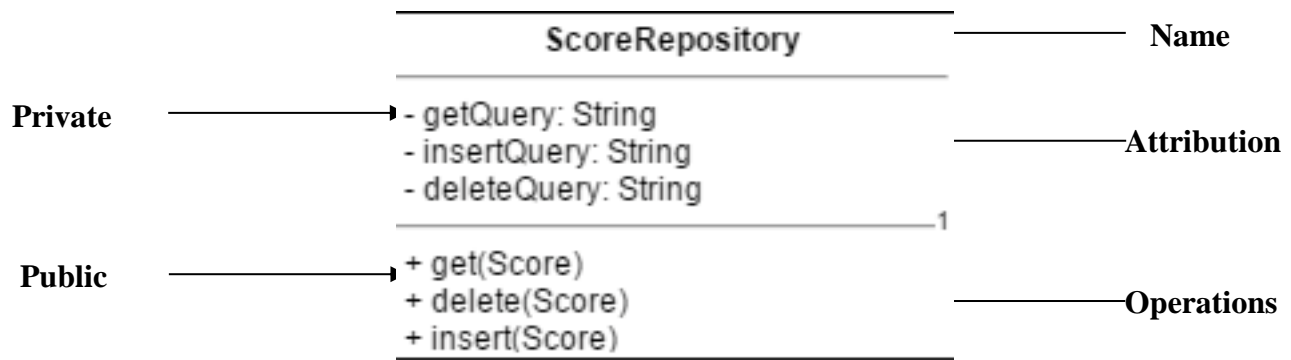
UML is Unified Modeling Language. The UML is an international industry standard graphical notation for describing software analysis and design (Thomas, 1997). We use UML to create Class Diagrams. Class diagrams are used both in the process of analysis and the design. During the analysis and design phase, we detailed implementation information, including class name, the methods and attribute of the classes, and the relationships among classes.

1. The goal of UML is using a picture to replace thousands of words. The most important goal is defining some general-purpose modelling language which all designers can use so as to be simpler to understand and use. (TutorialsPoint, n.d.).
2. A conceptual model can be defined as a model which consists of concepts and their relationships. The conceptual model is the first step before drawing a UML diagram. It helps to understand entities in the real world and how they interact with each other. Because UML describes real-time systems, it is important to develop a conceptual model and then proceed systematically. The UML conceptual model can be learned by the following three main elements: UML building blocks, rules for connecting building blocks, UML Common Mechanism (TutorialsPoint, n.d.).
3. The UML plays an important role in defining different perspective of a system, including design, implementation, process, and deployment. The UML Diagram connects all these four. A UML Diagram shows the functionality of the system.
4. Designing a system consists of classes, interfaces and collaborations. Implementation defines the components together to show a complete system. The process defines the flow of the system and deployment represents the physical nodes of the system that forms the hardware.
5. According to use cases definition, there should be two sides in the system: server and client side. Client side is separated with the model and controller. The model has main attributes of the uses cases. View side is the screen and results that are shown to the users of the game. For the server side, controllers are used for keep track of the movement, the state, the position and the session of a match. In addition to these, databases should keep the records of the game.
6. The top compartment shows the class's name. The middle compartment lists the class's attributes. The bottom compartment lists the class's operations. When drawing a class element on a class diagram, you must use the top compartment, and the bottom two compartments are optional. (The bottom two would be unnecessary on a diagram depicting a higher level of detail in which the purpose is to show only the relationship between the classifiers.) (Bell, 2004).

The UML class is represented by the diagram shown below:

- The top section is used to name the class
- The second one is used to show the attributes of the class
- The third section is used to describe the operations performed by the class.

## Class



We decided to architect two fundamentally separate components: server and client. The server uses the MVC design pattern, where JSON encoded results are the view. Models on the client have attributes from the CRC cards. A database keeps track of data that needs to persist between sessions and the diagrams show how we envisioned using the Repository pattern (“Repo”) to access the database.

The diagram also shows our intention to implement our game logic in services, such that they are reusable and separated from the controllers. This way the controllers can be very simple and game logic is reusable between controllers.

## 11.1 Client Side

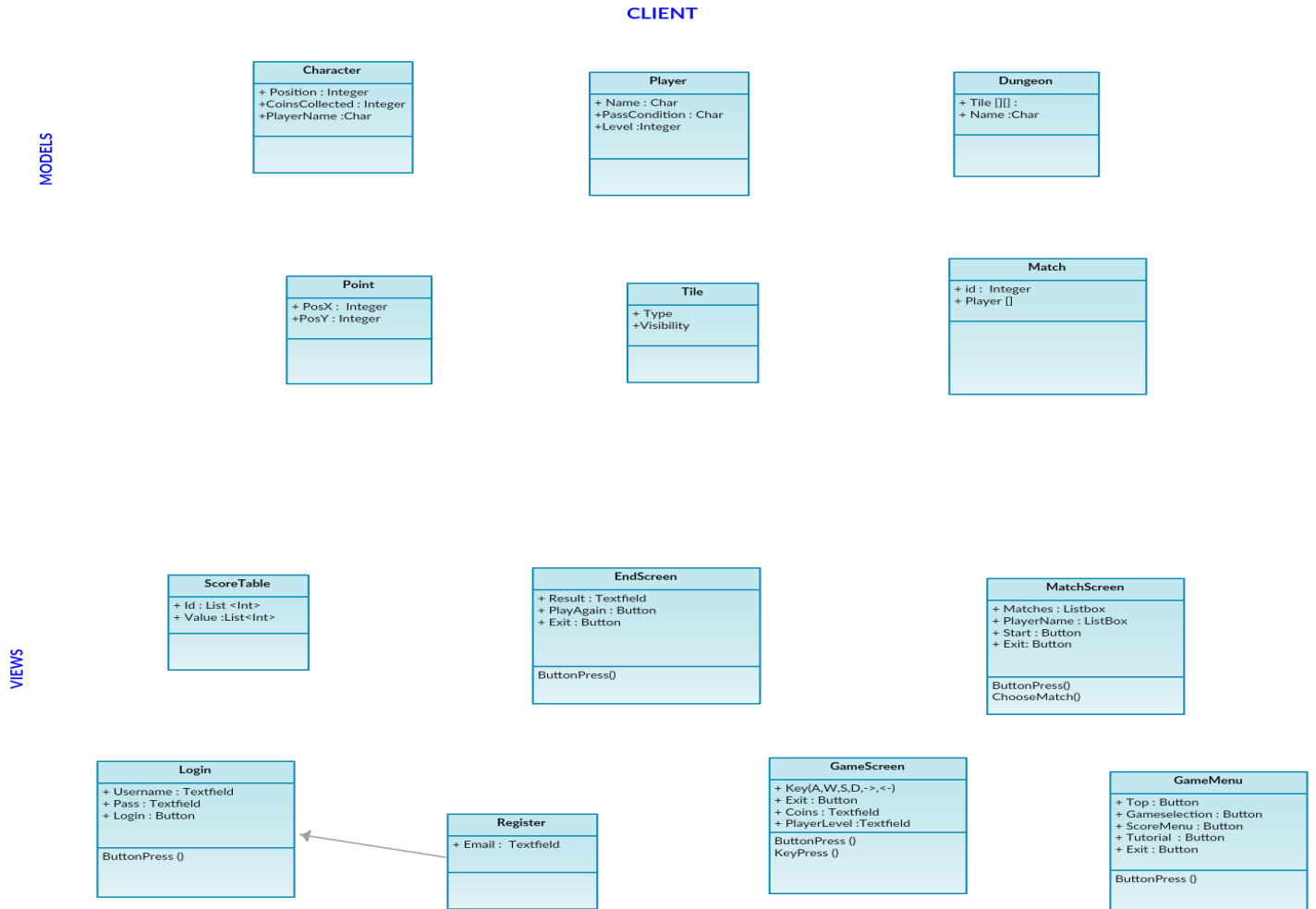


Figure 1. UML for client v1

## 11.2 Server-side Controllers

### CONTROLLER

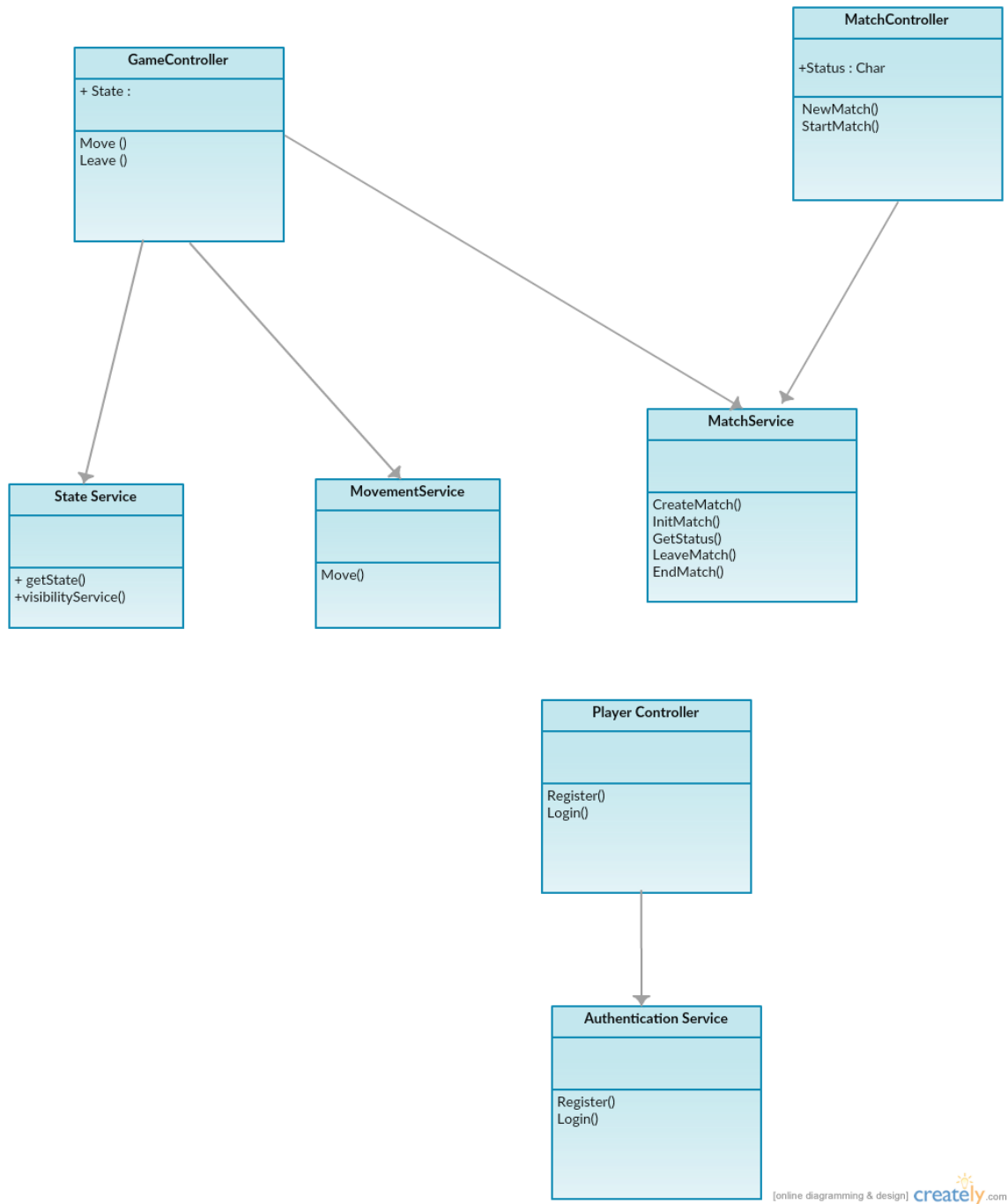


Figure 2. UML for server-side controllers and their services v1

## 11.3 4.3 Server-Side Models & Database

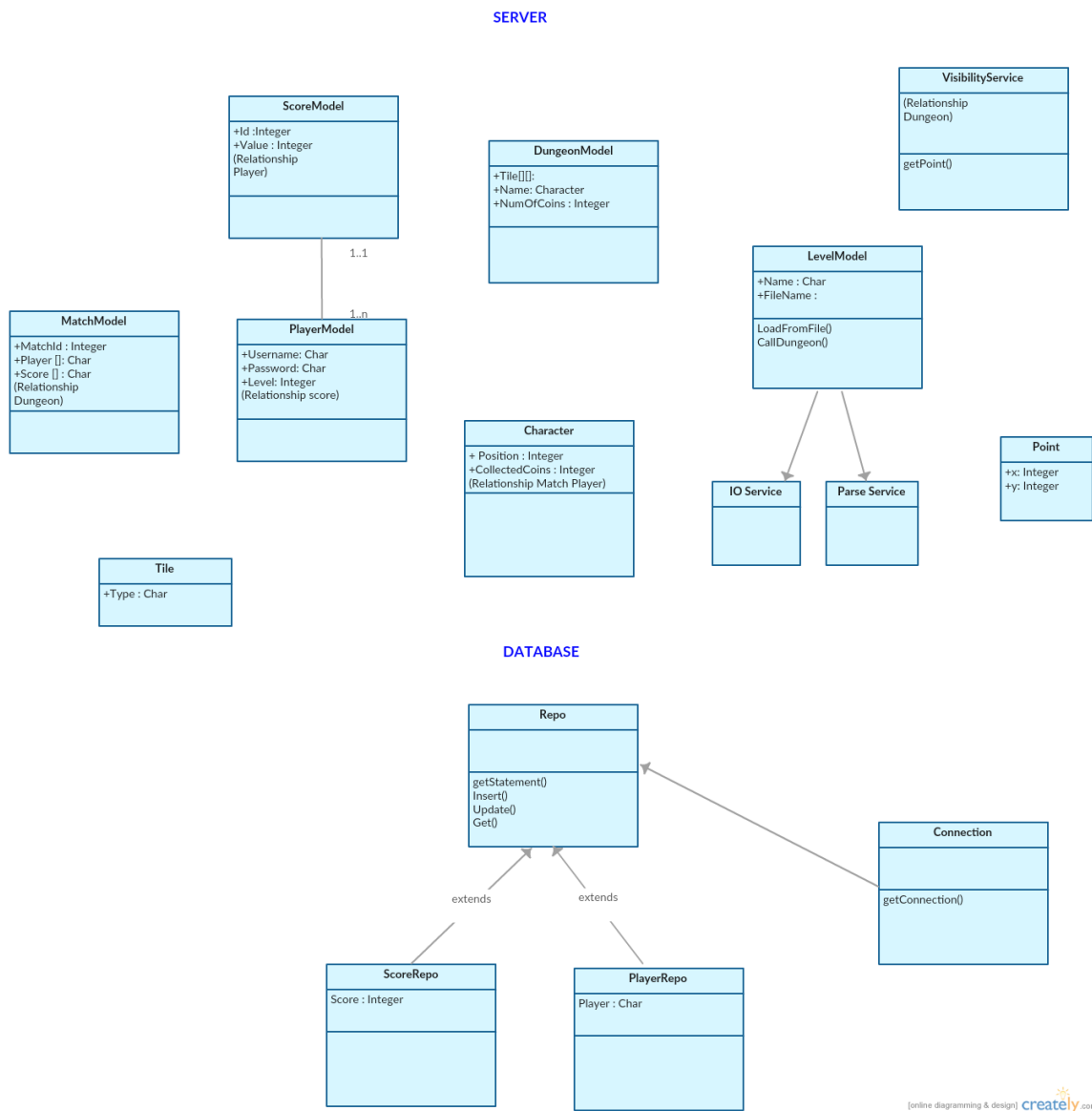


Figure 3. UML for server-side models & database layer v1

After the design of UML diagrams, the system is ready for development.

## 12. Defining Non-Functional Requirements

### 12.1 Questions

In the third sprint, the team discussed:

- Which programming languages and technologies could be used for both front-end and back-end?
- How should MVC be implemented?
- Which database is more suitable?

## 12.2 Non-Functional Requirements

During our meetings, we tried to answer the above questions and we came up with the following non-functional requirements:

- The system is designed with 3-tier architecture. It consists of 3 layers: data layer, logic layer and presentation layer.
- The system uses MVC design pattern. It consists of 3 components: view, model and controllers which interact with each other.
- The database is done by using MySQL server.
- The server side is written in Java.
- The client side is written in HTML5&CSS3&JavaScript.
- The graphical representation will use HTML5 canvas and pixiJS.

## 12.3 System Architecture

This section provides an overview of the architecture of the system. The major architectural goal is to reduce redundancy and code reusability. This section includes some of the architectural decisions about how the use cases converted into the system components and how these components are communicating.

### 12.3.1. Architectural Views

To design this system, we looked at it from different perspectives. Hence, the architecture of Dungeon of Doom can be viewed in three ways:

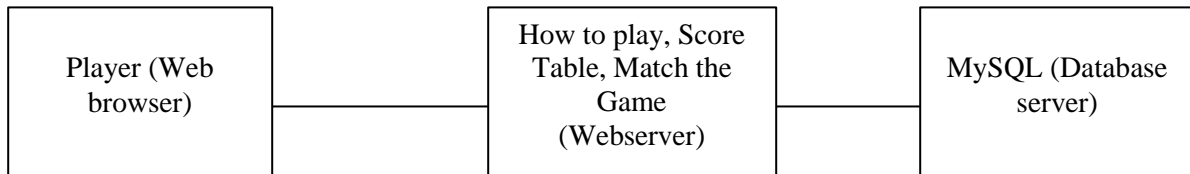
- Use Case View: the primary focus of the use case view is to specify the important drivers of the game i.e.; requirements
- Design View: the design view of the Dungeon of Doom is done in the form of class diagram to represent the functionalities of the game to the user or player.
- Deployment View: This view represents all the physical nodes that shows how the online multiplayer works all together.

Along with these views, CRC cards should also be used during the design of system architecture.

### 12.3.2. Deployment View

The deployment view of Dungeon of Doom represents the physical nodes that connect at the time of execution. The Dungeon of Doom contains three physical nodes, which are the browser, the web server, and the database server. The connections between these physical nodes are illustrated in the following diagram.





**Figure 4. Main components based on architectural view.**

### 12.3.3. Architectural Design Patterns

The design pattern that is implemented in the server side application code of Dungeon of Doom is MVC (Model View Controller). In compliance with the use cases, each requirement is designed in accordance MVC by separating view, model and controller.

The Models are Java classes used to represent data. These model classes (Eckstein, 2007) are independent of their controller and view.

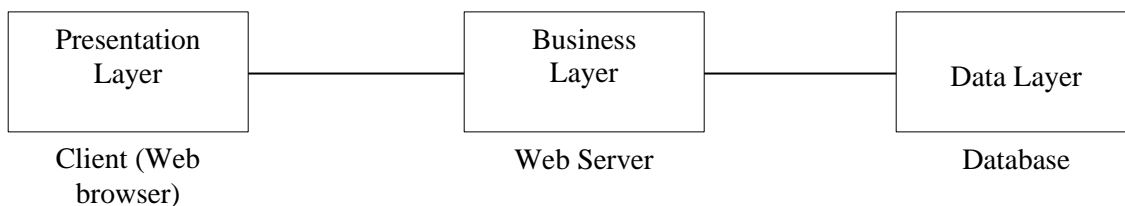
The view or presentation of Dungeon of Doom's server is implemented through JAXB annotated models, which the Jersey framework converts into JSON files.

The controller (Eckstein, 2007) performs the interconnection between MVC components. The controllers are Java classes which use annotations of Jersey framework. With these annotations, the Jersey framework identifies the appropriate method from the controller class.

The main advantages of using MVC design pattern in the system are that it encourages separation of concerns allowing us to write reusable functionality and reduce dependencies between components.

### 12.3.4. Architectural Style

We decided to use Three-Tier Architecture. The system will consist of 3 components: Presentation layer, Business Logic Layer, and Database Layer:



- Data Layer:
  - A MySQL database that contains information about Players and Score, and the Repository pattern in the Java code to interact with that database.
  - The JSON-encoded map assets in the file system.
- Logic Layer:
  - Services that contain game logic
  - Models of the game's components in Java code.

- These components will be encoded in JSON using the Jersey API framework's support for MOXy JSON parser with JAXB annotations.
- Presentation Layer:
  - Consists of the JavaScript client
    - The client interacts by sending requests to the web service (back-end).
    - The display will be rendered using HTML5 Canvas through the PIXIJS framework
    - The GUI will be formatted in HTML and CSS3

A layer needs to communicate with different layers to complete the requested service. For instance, the user needs to log in to the application before starting playing game. Once the user types the username and password on the client side, it invokes a service to logic tier and from there it connects to data layer to check if the username and password are valid. If the user record exists, a response will be sent to the user.

In our project, we decided to use 3-tier architecture because it allowed the team members to work efficiently and independently on each compartmentalized component. It reduces the dependencies into the system across different layers, thus, allowing us to test different components with ease.

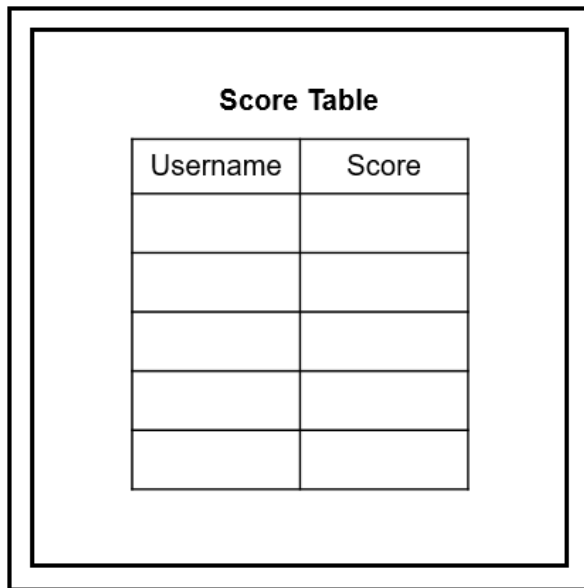
## 13. Layout Design

### 13.1 Layout Mock-up Design

In the third sprint meeting we discussed designs. Firstly, simple drafts of design were drawn to paper. Then, in the fourth sprint, we converted the designs into simple graphics.

Figure 5 shows our design for the score table. It would be populated by scores that belong to particular players. Figure 6 shows our design for the login/registration page. You would be able to login and register from the same form, which simplifies the requirement and reduces redundancy.

**Login/Registration Form**

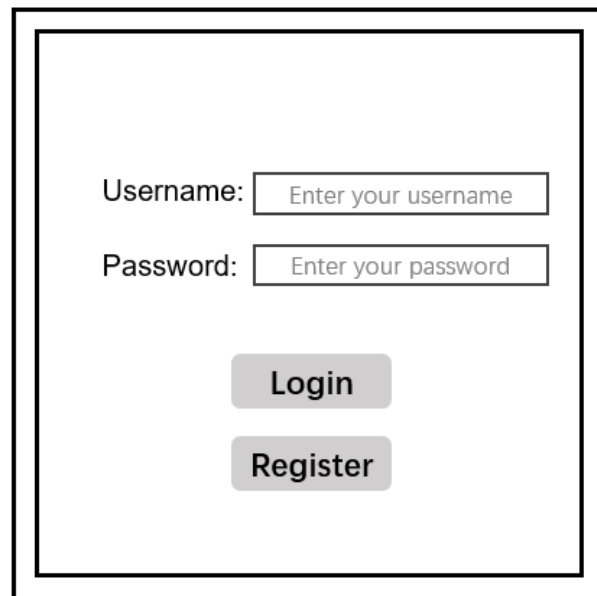


The mockup shows a rectangular frame containing a table titled "Score Table". The table has two columns: "Username" and "Score". There are six rows in total, with the first row being the header and the remaining five rows being empty for data entry.

Username	Score

**Figure 5. Login and Registration Layout**

**Score Layout**



The mockup shows a rectangular frame containing a form. It has two input fields: "Username:" followed by a text box with the placeholder "Enter your username", and "Password:" followed by a text box with the placeholder "Enter your password". Below these fields are two buttons: "Login" and "Register".

Username:

Password:

Login

Register

**Figure 6. Score Table Layout**

### 13.2 Level Design

The layouts of the dungeon are shown in Figure 7 below. There are three levels in the dungeon in our game. Each dungeon consists of four rooms; each room includes walls, paths. The number of coins in the map is defined in the beginning and coins are placed in the path randomly. The amount of coins required to win may vary between levels.

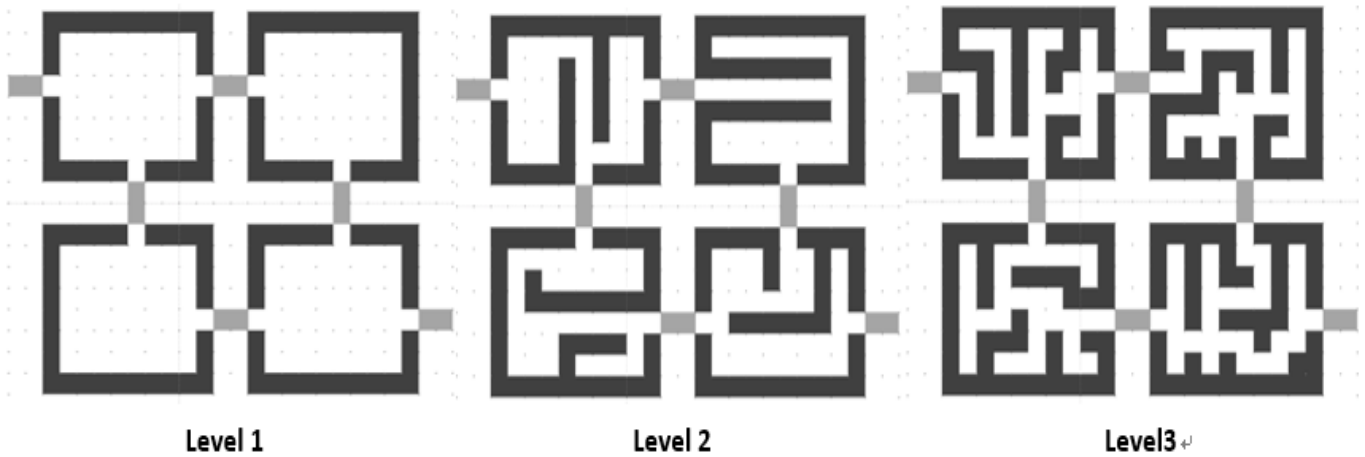


Figure 7. Levels of the Game

### 13.3 Implementation of Design and Interface

In the process of designing the user interface, we followed the subsequent design principles:

- **Unity:** all the pages in our game have a united background.
- **Simplicity:** we minimize the number of buttons which makes it easier to operate and reduces redundancy. This also reduces the cognitive load of the user. The rooms in the map are arranged orderly and the paths in the dungeon are straight which will make it easier for users to enter another room and find the exit.
- **Intuitive:** In the game map, we use simple symbols to represent different types of tiles in order that users can distinguish them well. There should be some text and word areas to show the game state for users including the number of coins they have collected and the amount of coins required to win.
- **User-friendly:** While playing the game, the user's character should always stand at the centre of the screen, the different areas of the map will be shown based on the player's position.

#### **14. Functional requirements v2**

During Sprint 5, we decided to work on the multiplayer feature of our game that was vaguely discussed on previous weeks. Firstly, we reviewed functional requirements of our game and we introduced new requirements for our new features.

**R1.** The user must log in to play the game.

R1.1. If the player is not registered to the game, he can register by giving a username, a password.

**R2.** The user can play single player or multiplayer against another human player or bot.

R2.1. The user can wait for other players or join a previously created match.

R2.2. The user can choose to play against computer (bot).

R2.3. The user can choose to play single player game.

**R3.** The user chooses options from a menu (new match, start a match, join a match, tutorial, or score)

R3.1. The user can either create a new match or join a match.

R3.2. If the user selects new match, the user can either wait for another player or start a single player game.

R3.3. If the user selects join a match, they start playing another human player.

R3.4. If the user selects score, they can see the scoreboards.

R3.5. If the user selects tutorial, they see the instructions of the game.

R3.6. If the user selects to exit or closes the tab, they leave the game.

**R4.** The world is loaded on the server's memory.

**R5.** The player can see a part of the dungeon (explored tiles), other players (human or bots), gold coins, passages, and exit.

R5.1. A dungeon is a collection of rooms that are connected by passages. A room may have gold coins.

R5.2. A dungeon can be arbitrary size.

R5.3. Dungeon must contain the minimum gold coins so the player can win.

**R6.** Player interacts with the dungeon:

R6.1. Indicating which way to move (UP, DOWN, LEFT, RIGHT)

R6.2. Picking up gold.

R6.3. Player can move around to reveal the room.

R6.4. Leaving the game.

**R7.** Winning Condition:

R7.1. If the player collects all the gold coins, needed to win, and find the exit before the other player, the player wins.

R7.2. Else if another player wins first or the player gives up, the player loses.

R8. After winning condition, the player goes to result screen.

### 15. Product Timescale v3

Sprint	Effort points	Sprint 0	Sprint 1	Sprint 2	Sprint 3	Sprint 4	Sprint 5	Sprint 6
Date	1 = 0.5 day	31/10/16	7/11/16	14/11/16	21/11/16	28/11/16	5/12/16	12/12/16
Milestones		Start			Start Coding	Basic Functionality		
Login Functionality	3					X		
Registration Functionality	3					X		
Login/Registration Layout	1				X			
Menu Design	2				X			
Menu Tutorial	2				X			X
Score Layout	2				X			
Leaving (exit by menu)	4							X
Timeout (exit by absence)	2							X
Design map JSON	4				X			
Parse map JSON	4					X		
Model map in code	2					X		
Render the map in client	3					X		
Create player model in server	2					X		
Communication Between server/client	4						X	
Visibility-server	4						X	
Visibility-client	4							X
Movement	2						X	
Score- model	4						X	
Score-calculate	1						X	

Score- add coin	2							X
Setup database	2				X			
Add winning condition	4							X
Match Functionality	4						X	
Implement Score Table	2						X	
User Guide	1							X
Add bot	4							X
Total Sprint Effort	73				12	16	16	17

Table 20. Project Timescale v3

## 16. Use Case V3- Multiplayer Use Case

Use Case 10:

UC10-1: Use Case: Player start a new match

UC10-2: Author: AG

UC10-3: Date: Sprint 5

UC10-4: Purpose: The user initiates a new match.

UC10-5: Overview: The player selects to start a new match. A request is sent from client to server to initialize a match. The server initializes the match and sends dungeon map representation back to the client. Alternative 1: The player waits at the lobby screen for another player. When a new player joins, the server notifies the first player. Either player can start the match. A request is sent to the server to start the game. The server sends the dungeon map representation to both players.

UC10-6: Cross Reference: R2.1, R2.2, R2.3, R3.1, R3.2, R3.3

UC10-7: Actors: Players

UC10-8: Pre-Condition:

UC10-Pre-1: The player must have selected a level and wait at the lobby screen.

UC10-Pre-2: The player must be already logged on.

UC10-9: Post-Condition:

UC10-Post-1: The player(s) is/are inside the dungeon.

Actor Actions

**1. Begins when player starts a match.**

Client System Actions

**2. Sends button request to server**

Server System Actions

		3. Sends response with visible area by the player
	4. Receives and parses response	
	5. The client draws the graphical representation of the dungeon.	
Actor Actions	Client System Actions	Server System Actions
1. The player waits for another player to join the match.		
2. Player 2 joins the match.		
		3. The server sends a response to the first player
	4. Client receives and response and update number of players on screen.	
5. One of 2 players begins the match.	6. The client sends a request to start the match.	
		7. Server gets the request and sends a response to both players with the visible area by the players.
	8. Receives and parses response	
	9. The client draws the graphical representation of the dungeon for each player.	

UC10-11: Exceptional flow of events:

Steps 3, 4, 6, 7, 8: The request or response network packets are dropped or corrupted. Sender sends a request for the current state of the system. If that request fails, the client displays an appropriate message regarding network connectivity problems to the player.

## 17. UML v2

As the project progressed and we further refined the design we updated the UML to reflect the finer details we'd come to design.

### 17.1 Domain package



This package contains our domain models and database layer. You can see how we used interfaces to abstract the repositories for testing and created a `DatabaseRepository<T>` base class to generify the logic used to generate database statements. We also have a static `DatabaseConnection` class to ensure that the same connection is reused across different repositories and different connections.

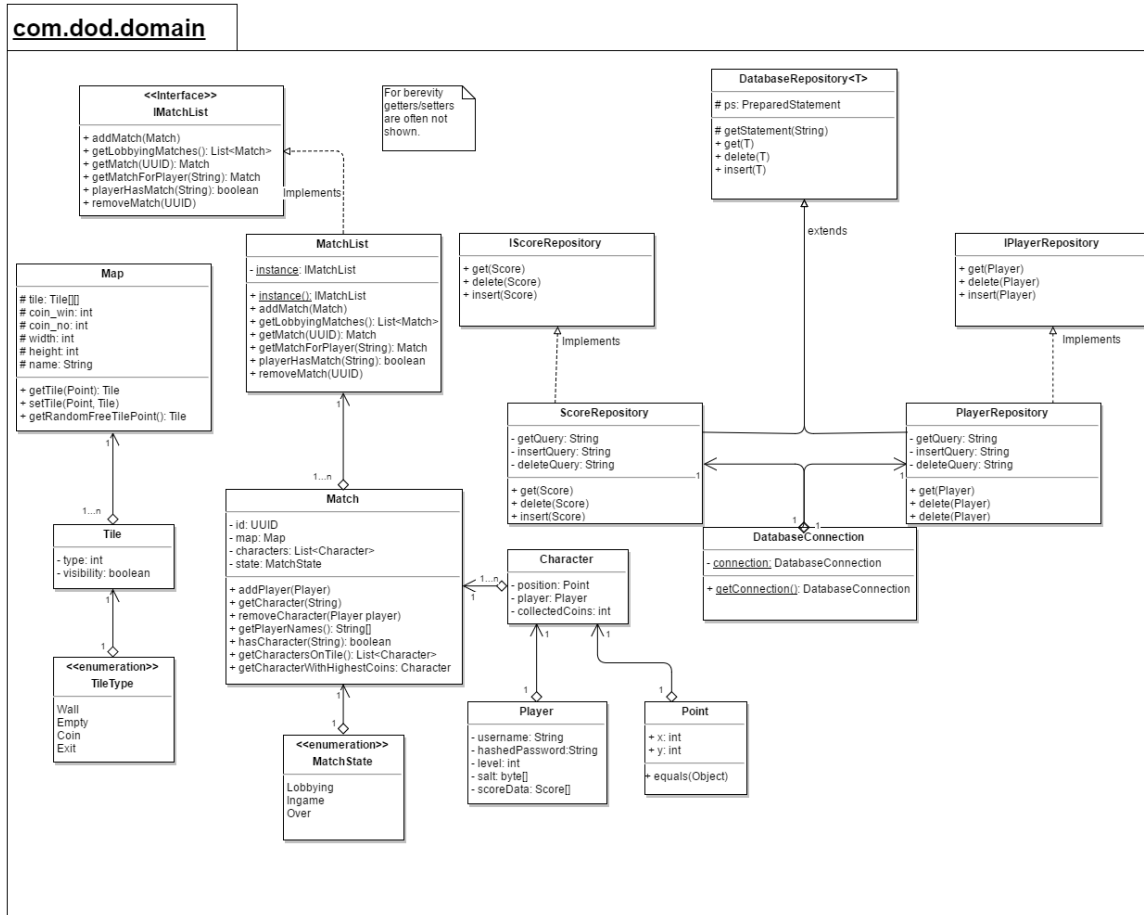


Figure 8. Version 2 UML for Java Domain Package

## 17.2 Service package

The service package controls the web server. The controllers define the API paths and how an endpoint will respond, while models define the structure of our JSON responses and services contain generic and reusable game logic.

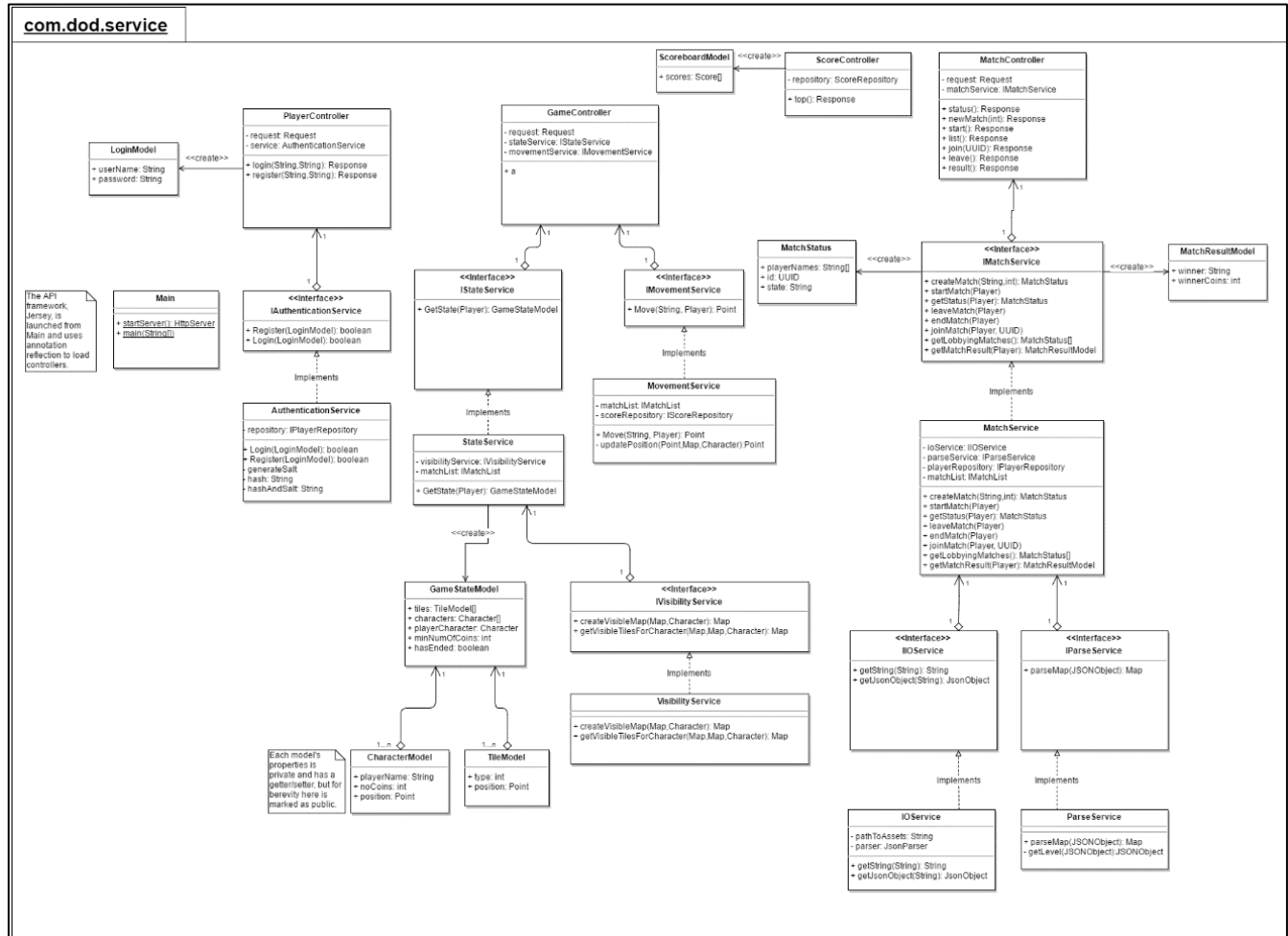


Figure 9. Version 2 UML for Java Service Package

### 17.3 Bot package

Our last package is our *bot* package, where the source code of our bot resides. CommunicatorBase manages generic communication between the bot and the server, and the specialized communicators send specific messages for joining a match, moving in a direction etc.

Map is used to construct an abstract map that models the game state based on the response from the server- given that the server only returns the small number of tiles immediately nearby the character's position.

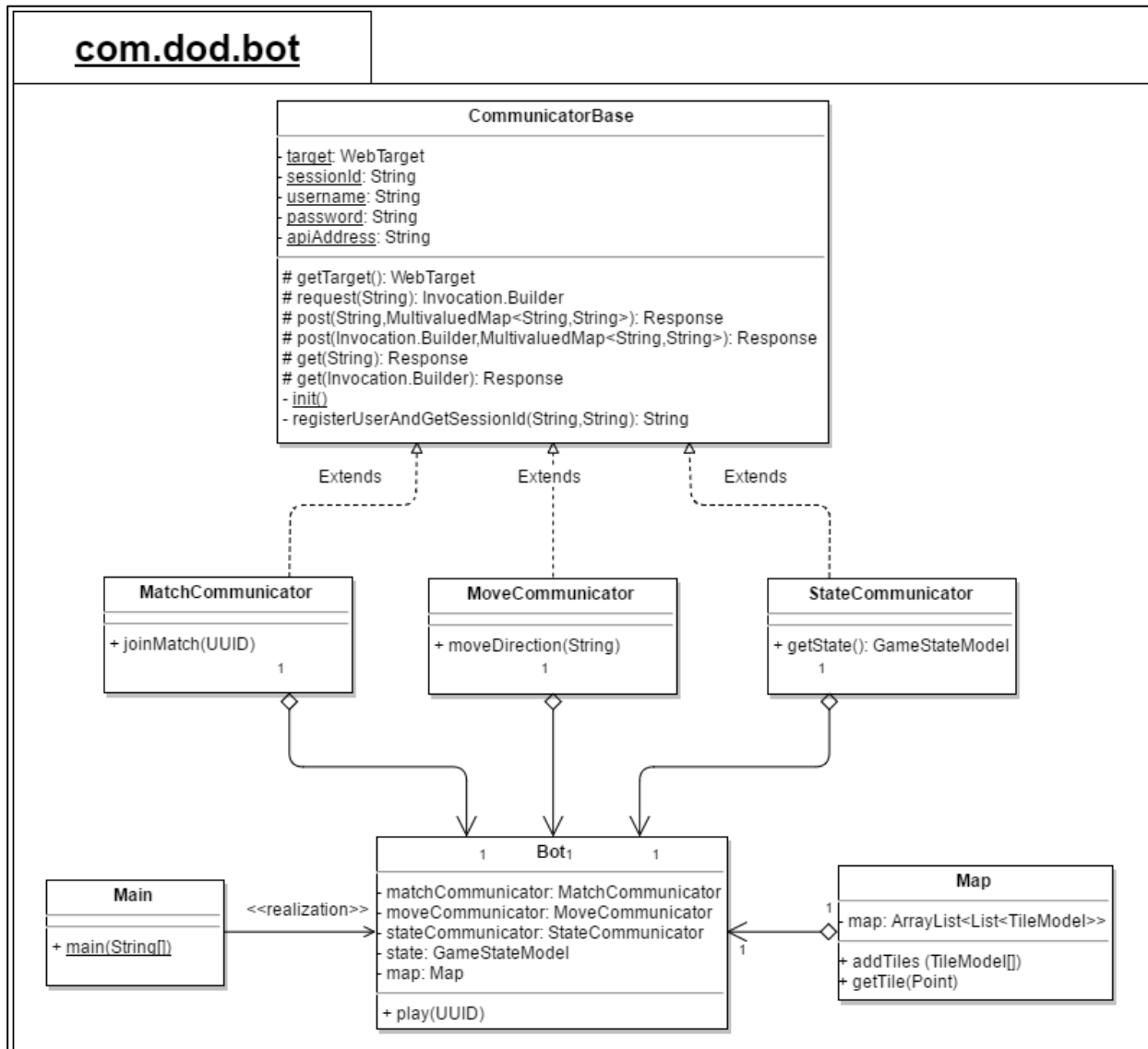


Figure 10. Version 2 UML for Java Bot Package

## 18. User Guide

### 18.1 System Overview

The dungeon of doom is an online multiplayer game with three levels. The objective of the game is to collect the specified amount of gold in the dungeon and get to the exit before another player.

## 18.2 Login and Register

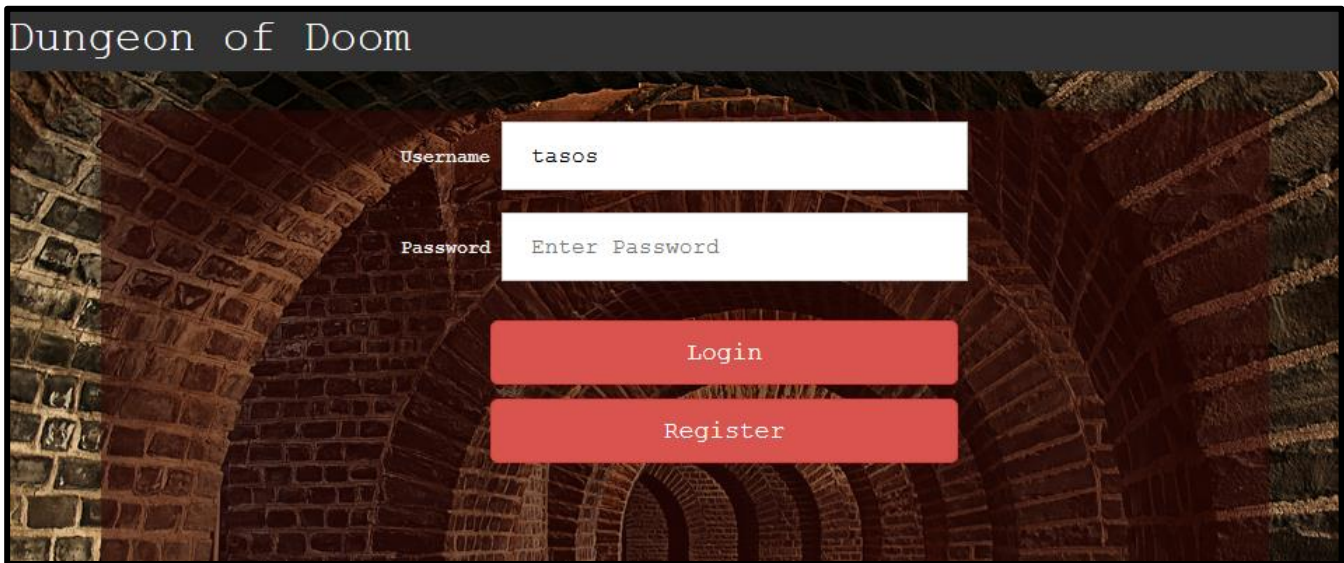


Figure 11. Login and Registration screen

To get access to the Dungeon of Doom, the user need to log in first simply by typing the username and password followed by clicking the login button. It needs to note that both the password and username are case-sensitive which means the system considers “F” and “f” separately.

Once the user entered successfully, it will show the Lobby screen where the user can choose the appropriate option for further use. In case of wrong login details, the system shows a message saying incorrect username or password.

The new user can register on to system simply by typing new username and password followed by clicking on register button.

## 18.3 Lobby

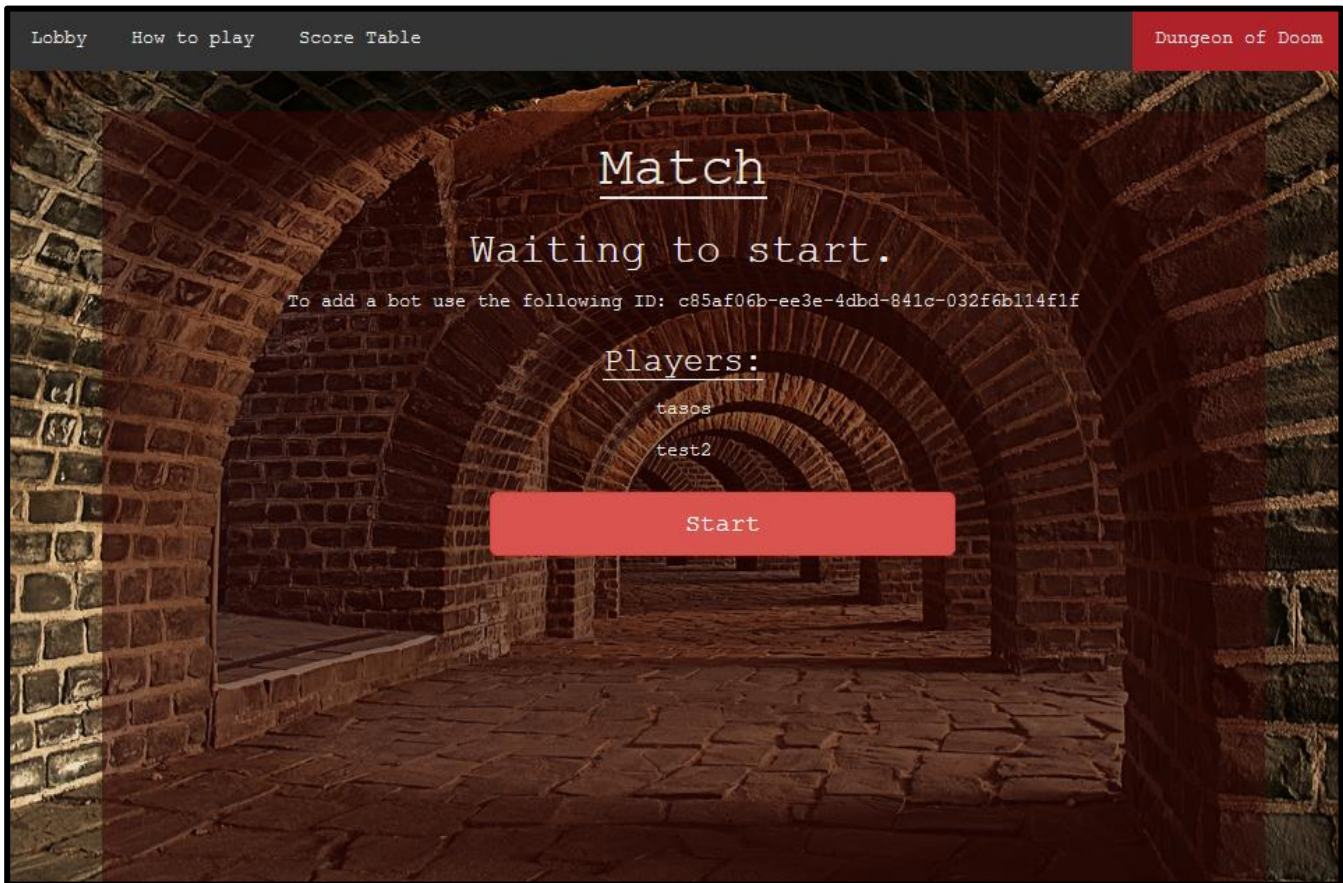


Figure 12. Lobby

Once the user successfully logged onto the system, the next screen appears is Lobby screen. On Lobby there are some tabs are provided for users choice in top left navigation bar.

Lobby: by clicking on Lobby tab, the system goes to lobby or refreshes the lobby screen.

How to Play: by clicking on this tab directs to the detailed instructions on how to play the game Dungeon of Doom.

Score Table: If the user wants to see scores achieved in the completed previous game, it can view by clicking on score table tab.

Game selection: The player can choose the level and add other players from game selection area of lobby.

## 18.4 How to play

How to play section provides the detailed instructions on how the game works and the instructions player need to follow. It also included the mockups explaining the game screen.





Figure 13. Selecting Level

From the lobby, there is an option for selecting the game level. For multiplayer option, the player needs to click on Join link to add another player. Or else player can add a bot by using an ID specified on the screen. Followed by clicking on Start button, the game starts on both sides.

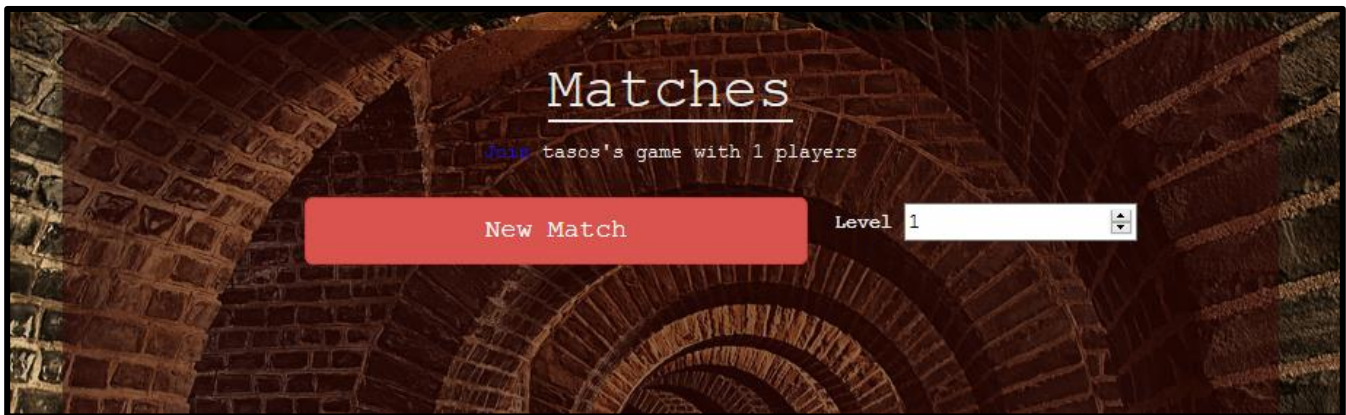


Figure 14. Join link for Multiplayer Game

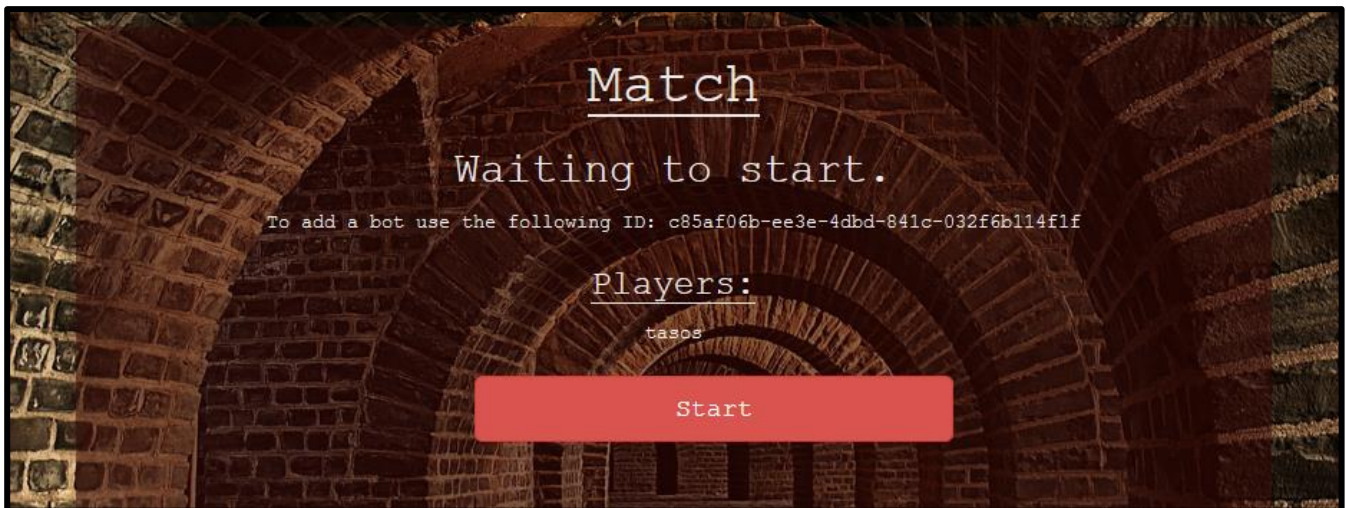
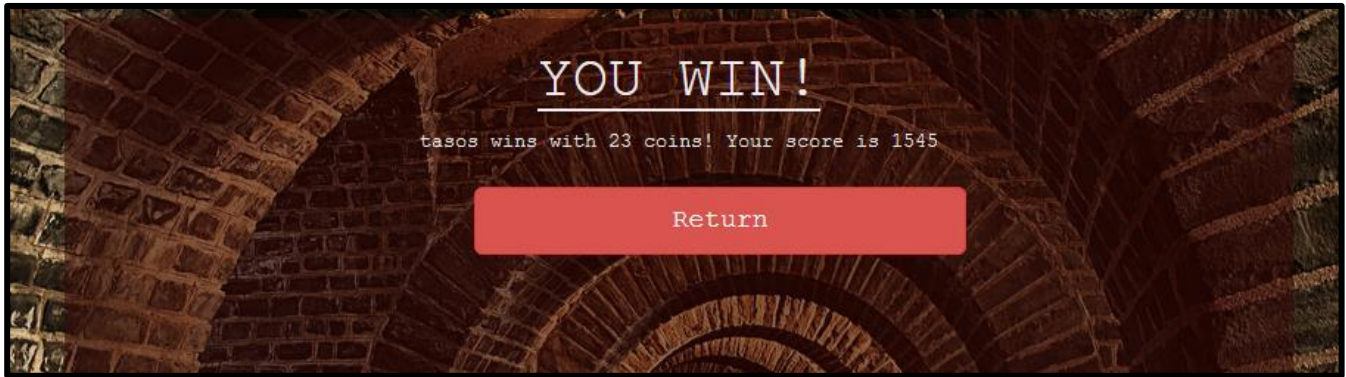


Figure 15. Option for adding bot

Once the game starts by entering the dungeon, the player need to collect all the gold coins by moving through the location where coins are placed. During the game, the player has to pass through the passages to enter into different rooms. There are some walls placed around the dungeon where player cannot enter into it.

Once the player collected the minimum gold coins in the dungeon, the exit will be opened and the player need to pass through it before the other player. Upon going through the exit, the game will finish and result screen will appear.



**Figure 16. Result screen**

The movements of the player are controlled by W, A, S and D and arrow keys and there is a leave button also if player needs to leave from the game.

### **18.5 Game Screen**

In the game screen, the yellow colored dots indicates the gold coins in the dungeon the player is denoted by a character in blue color circle. Stairs icon in the dungeon denotes the exit.

A message will be displayed on top the screen about the coins already collected and coins need to collect.



Figure 17. Gold Coins and Player Character



Figure 18. Exit from Dungeon



## 18.6 Score Table



Username	Score
test2	123249
test2	97001
test2	26287
ilias	12527
wert	12328
wert	11587
wert	11300
tasos	10995
tasos	10677
test2	9887

Figure 19. Score Table

The player can view the scores of completed previous games by clicking on score table tab from Lobby.

## 19. Acknowledgements

We made use of a number of third-party libraries, resources and tools:

- Jersey RESTful Web Service Framework
  - Dual licensed
    - Common Development and Distribution License (CDDL - Version 1.1)
    - GNU General Public License (GPL - Version 2, June 1991) with the “Classpath Exception”
  - Built by Oracle
  - <https://jersey.java.net/index.html>
- PixiJS
  - Javascript HTML5 Canvas graphics engine
  - MIT Open License
  - Main author Mat Groves
  - <http://www.pixijs.com/>

- JUnit
  - Eclipse Public License 1.0
  - Authored by JUnit
  - <http://junit.org/junit4/>
- “Crawl” tileset graphics
  - Autor Chris Hamons
  - Public Domain
  - <http://opengameart.org/content/dungeon-crawl-32x32-tiles>
- JQuery
  - Authored by the JS Foundation <https://js.foundation/>
  - <https://jquery.com/>
- “Vaulted Cellar” Photograph
  - Used in the background of the client layout
  - Public Domain
  - Author is a user named “132369”
  - <https://pixabay.com/en/vaulted-cellar-tunnel-arches-keller-247391/>
- “VT232” Font
  - Used in the client
  - Designed by Peter Hull
  - Open Font License
  - <https://fonts.google.com/specimen/VT323?selection.family=VT323>
- Creately.com
  - Tool used to generate the first version of UML diagrams
  - <http://www.creately.com>
- Draw.io
  - Tool used to generate the second version of UML diagrams
  - <http://www.draw.io>
- Mockito Java mocking framework
  - MIT Open License
  - <http://site.mockito.org/>

## 20. References

- Bageri, S., 2013. *Create and Implement 3-Tier Architecture in ASP.Net* [Online]. Mumbai, India: C# Corner. Available from: <http://www.c-sharpcorner.com/uploadfile/4d9083/create-and-implement-3-tier-architecture-in-asp-net/> [Accessed 20/11/2016]
- Bell, D., 2004. *The Class Diagram* [Online]. Durham, NC: IBM. Available from: <https://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell/> [Accessed 20/11/2016]
- Cockburn, A., 2000. *Writing Effective Usecases*. Boston, U.S.: Addison Wesley.
- Eckstein, R., 2007. *Java SE Application Design With MVC* [Online]. London, UK: Oracle. Available from: <http://www.oracle.com/technetwork/articles/javase/index-142890.html> [Accessed 20/11/2016]
- OpenLoop Technologies, n.d.. *Introduction to CRC Cards* [Online]. San Jose, CA, U.S.: Openloop Computing. Available from: [http://www.openloop.com/education/classes/umn\\_98/umn\\_cpp/crc.htm](http://www.openloop.com/education/classes/umn_98/umn_cpp/crc.htm) [Accessed 10/12/2016]
- Alshehri, S., L. B., 2013. Prioritizing CRC cards as a simple design tool in Extreme Programming. *2013 26th IEEE Canadian Conference Of Electrical And Computer Engineering (CCECE)*, 5-8 May 2013, Regina, Canada. Regina: IEEE, pp. 1-4.
- Thomas, R., 1997. Introduction to the Unified Modeling Language. *Proceedings. Technology of Object-Oriented Languages and Systems, 25 (97TB100239)*, pp. 354-354.
- TutorialsPoint., n.d.. *UML – Overview* [Online]. India: TutorialsPoint. Available from: [https://www.tutorialspoint.com/uml/uml\\_overview.htm](https://www.tutorialspoint.com/uml/uml_overview.htm) [Accessed 20/11/2016]
- Visual Paradigm, 2016. *Writing Effective Usecase*. Kln, Hong Kong: Visual Paradigm. Available from: <https://www.visual-paradigm.com/tutorials/writingeffectiveusecase.jsp> [Accessed 10/12/2016]