|  |  |
| --- | --- |
|  |  |
|  | DUNGEOON OF DOOOM  REPORT V01 |

|  |
| --- |
| [TEAM MEMBERS]  MATTSI JANSKY  ARYA ramdev  SELIN KUTLAMIS  TASSOS GEMTOS  QIAN ZHOU  XIAOXIAO FAN |

1. Contents

[1. READER’S GUIDE 3](#_Toc469494978)

[2. Functional Requirements 4](#_Toc469494979)

[3. Initial Project Development Process 4](#_Toc469494980)

[4. Producing Project Timescale 4](#_Toc469494981)

[5. Use Cases v1 6](#_Toc469494982)

[5.1 Use Case 1 – Log in 6](#_Toc469494983)

[5.2 Use Case 2 – Registration 7](#_Toc469494984)

[5.3 Use Case 3 – Main Menu 7](#_Toc469494985)

[5.4 Use Case 4 – World Creation 7](#_Toc469494986)

[5.5 Use Case 5 – Movement 8](#_Toc469494987)

[5.6 Use Case 6 – Coins 8](#_Toc469494988)

[5.7 Use Case 7 – Map visibility 8](#_Toc469494989)

[5.8 Use Case 8 – Exit 8](#_Toc469494990)

[5.9 Use Case 9 – Winning Condition 8](#_Toc469494991)

[5.10 Process of the Use Case Creation 9](#_Toc469494992)

[6. CRC Cards 9](#_Toc469494993)

[7. 2. Process of Creating CRC Cards 10](#_Toc469494994)

[8. Uses Cases v2 11](#_Toc469494995)

[8.1 Use Case 1 – Login with System and Client Relationship 11](#_Toc469494996)

[8.2 Use Case 2 – Registration with Server and Client Relationship 11](#_Toc469494997)

[8.3 Use Case 3 – Main Menu with Server and Client Relationship 12](#_Toc469494998)

[8.4 Use Case 4 – World Creation with Server and Client Relationship 14](#_Toc469494999)

[8.5 Use Case 6 – Coins with System and Client Relationship 16](#_Toc469495000)

[8.6 Use Case 7 – Map visibility with System and Client Relationship 16](#_Toc469495001)

[8.7 Use Case 8 – Exit with System and Client Relationship 17](#_Toc469495002)

[8.8 Use Case 9 – Winning Condition with System and Client Relationship 18](#_Toc469495003)

[9. UML Diagrams v1 20](#_Toc469495004)

[9.1 4.1 Client Side 20](#_Toc469495005)

[9.2 4.2 Server-side Controllers 21](#_Toc469495006)

[9.3 4.3 Server-Side Models & Database 22](#_Toc469495007)

[10. Defining Non-Functional Requirements 23](#_Toc469495008)

[11. Layouts of Design 23](#_Toc469495009)

[11.1 1. Mock up Design 23](#_Toc469495010)

[11.2 2. Implementation of Design and Interface 24](#_Toc469495011)

[12. System Architecture 24](#_Toc469495012)

[12.1 1. Three-Tier Architecture 24](#_Toc469495013)

[12.2 2. Design Pattern 25](#_Toc469495014)

[13. UML v2 25](#_Toc469495015)

[13.1 Domain package 26](#_Toc469495016)

[13.2 Service package 27](#_Toc469495017)

[13.3 Bot package 28](#_Toc469495018)

[14. User Guide 28](#_Toc469495019)

1. READER’S GUIDE

1. Customer Requirements

This document contains the first version requirements for DungeonofDooom. These requirements have been derived from Moodle.

1.1 Types of Reader

Our readers are Senior Lecturer Julian Padget and Charlie Ann Page.

1.2 Technical Background Required

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

1.3 Scope of the Project

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

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.

3. Meeting Notes with Customer

3.1 Target and Budget of the Project

According to customer meeting notes; target is everyone. It is an online game and it is not requiring any technical background, training or education. There is no specific budget according to customer meetings.

3.2 General Constraints

Customer didn’t specify any specific languages. So in the sprint, team decided to use Java. For tile creation and parsing JSON is decided to use. For interface applications html and JavaScript are chosen. IntelliJ platform is installed.

3.3 Assumptions and Dependencies

Finished product can be delivered over the internet. It requires administrator only for adding a new player. No need for specific skills to play the game. Project should be finished until 16 December 2016.

1. Functional Requirements

We started the project by reading the specification in great detail as a group to decide on a list of *functional requirements*.

1. The user must log in to play the game.
   1. If the player is not registered to the game, he can register by giving a username, a password and an email.
2. The user chooses options from a menu (single, multiplayer, score or tutorial)
   1. If player selects single player game, he enters the level selection screen.
   2. If player selects multiplayer game, he enters the level selection screen.
   3. If player selects score, he sees the scoreboards.
   4. If player selects tutorial, the player sees the instructions of the game.
   5. If player can select exit or closes the tab.
3. The world is loaded on the server’s memory.
4. The player can see a part of the dungeon (explored tiles), other players (human or bots), gold coins, passages and exit.
   1. A dungeon is a collection of rooms that are connected with passages. A room may have gold coins.
   2. A dungeon can be arbitrary size.
   3. Dungeon must contain the minimum gold coins so the player can win.
5. Player interacts with the dungeon:
   1. Indicating which way to move (UP, DOWN, LEFT, RIGHT)
   2. Picking up gold.
   3. Player can move around to reveal the room.
   4. Leaving the game.
6. Winning Condition:
   1. If the player collects all the gold coins, needed to win, and find the exit, player wins.
   2. Else if another player wins first or the player gives up, the player loses.
7. After winning condition, the player goes to result screen.
8. Initial Project Development Process

For sprint 0, we started to produce project diaries. We prepared a project diary template. Project meeting minutes were added. For the sprint 1 based on these functional specifications we created initial use cases. Each team member created one or more use cases. In addition to these, we wrote the product timescale document.

1. Producing Project Timescale

Project timescale is built by Windows Excel.

**Graphic 1. Product Timescale**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Sprint | Effort points | Sprint 0 | Sprint 1 | Sprint 2 | Sprint 3 | Sprint 4 | Sprint 5 | Sprint 6 |
| Date | **(1 = 0.5 day)** | **31.10.2016** | **7.11.2016** | **14.1.2016** | **21.11.2016** | **28.11.2016** | **5.12.2016** | **12.12.2016** |
| Milestones |  | **Start** |  |  | **Code start** | **Basic**  **Functioanlity** | |  |
| Login Function. | 3 |  |  |  |  | **X** |  |  |
| Regist. Function. | **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** |  |  |
| Comm. between ser/cli | **4** |  |  |  |  |  | **X** |  |
| Visibility- server | **4** |  |  |  |  |  | **X** |  |
| Visibility- client | **4** |  |  |  |  |  |  | **X** |
| Move.- update playrPos | **2** |  |  |  |  |  | **X** |  |
| Score- model | **4** |  |  |  |  |  | **X** |  |
| Score- calculate | **1** |  |  |  |  |  | **X** |  |
| Score- add coin | **2** |  |  |  |  |  |  | **X** |
| Setup database | **2** |  |  |  | **X** |  |  |  |
| Add winning condi. | **4** |  |  |  |  |  |  | **X** |
| Sprint effort | **61** |  |  |  | **12** | **16** | **16** | **17** |

**Table 1. Effort Table**

1. Use Cases v1
   1. Use Case 1 – Log in

As a player, I want to login for seeing main menu

* User can view the login page.
* User can enter a username and password for login to the game.
* User can view fail screen if error is happened.

|  |
| --- |
| Actor Actions |
| 1. Open the game page. |
| 1. Input the username and password. |
| 1. Press “OK” button. |

Table 2. Actor Actions in Login

* 1. Use Case 2 – Registration

As a player, I want to register for returning login page.

* User can view the register page.
* User can enter username, password and e-mail for playing the game.
* User can view warning messages if s(he) enters wrong information.

|  |
| --- |
| Actor Actions |
| 1. Open the registration page. |
| 1. Input the username, the email and the password. |
| 1. Press “OK” button. |

Table 3. Actor Actions in Registration

* 1. Use Case 3 – Main Menu

As a player, I want to see the main menu to choose player type.

* User can enter the main menu.
* User can select between two options single and multiplayer for playing the game.
* User can view level section after s(he) selects player type for match.
* User can view dungeon after selection of level.
* User can play the game after seeing map.
* User can select score table for seeing the score.
* User can select tutorial to learn how to play.
* User can exit or log out from the system for returning login page.

|  |
| --- |
| Actor Actions |
| 1. Begins when player click single player of multiplayer button |
| 1. The user selects a match to play in. |

**Table 4. Actor Actions in Main Menu**

* 1. Use Case 4 – World Creation

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

* User can view the dungeon which consists of rooms for playing.
* User can view the rooms, passages which connect rooms.
* User can view the golds in the rooms for earning points.
* User can view the arbitrary size of rooms based on his/her level.
* User can see the minimum numbers of gold coins for winning the game.

|  |
| --- |
| Actor Actions |
| 1. Begins when player selects a match. |

**Table 5. Actor Actions in Dungeon**

* 1. Use Case 5 – Movement

As a player, I want to move in the dungeon for entering rooms and collect coins.

* User can move in the dungeon and go to rooms with passages.
* User can view the walls which block the direction.

|  |  |  |
| --- | --- | --- |
| |  | | --- | | Actor Actions | | 1. Begins when player presses movement key | |
| Table 6. Actor Actions in Movement |

* 1. Use Case 6 – Coins

As a player, I want to picking up gold coins for winning the game.

* User can move to a tile and collects the coins in the room for scoring.

|  |
| --- |
| Actor Actions |
| 1. Player begins to move to a gold coin. |

Table 7. Actor Actions in Coins

* 1. Use Case 7 – Map visibility

As a player, I want to see the map.

* User can view the map while moves around the dungeon.
* User can go from one tile to another for discover map layout.

* 1. Use Case 8 – Exit

As a player, I want to leave the game for stopping.

* User can view the exit option in the game.
* User can return to the main menu by using exit option.

|  |
| --- |
| Actor Actions |
| 1. The player presses the exit button. |

Table 8. Actor Actions in Exit

* 1. Use Case 9 – Winning Condition

As a player, I want to win by getting all coins.

* User can view the winning status.
* User can return view Main map with unlocked level to select different level.
* User can win the game by passing through exit.
* User can view the scoreboard; unlock the new level in the result screen with details which include time, coins or other entities player collected.
* User can view “You lose” message and return to result screen if another user exits first.

|  |
| --- |
| Actor Actions |
| 1. The player won the game. |

Table 9. Actor Actions in Winning Condition

* 1. Process of the Use Case Creation

User stories are great for understanding of what actually clients want. Development teams can ask who, what and why questions to the clients to understand their requirements. It should be written in non-technical format. By creating user stories, target goals can be reached more efficiently. [3] According to Cockburn in the use cases, actors have goals under the declared system responsibilities. And scenarios have a particular result with the respect to the goal.

In the light of this information, user stories are discussed in the brain-storming sessions in the sprint meetings. Cases are written on the blackboard. As a result of the first discussion, in an online game, player should see firstly login and registration. For making process simpler, in login page, only username and password should be entered. If player is not registered then player should be linked to register page. Team decided asking only username, email and password from user. For connecting end user and our team, main page includes only single/multiplayer selection. Scenarios are created like what if user wants to exit in the main page. Then exit in the main login is decided, which is connected to login page. In one of the user scenario, new users won’t know how to play the game. Based on this, creating tutorial decision is made. Adding timer to the game is cancelled due to creating extra load. Movement scenario includes which characters in the keyboard will be used and movement is made until player encounter with the wall. Game should be over when all the gold coins are collected by the player. Team decided to add different levels to the game after the single/multiplayer section. Ending of the game is discussed therefore exit use case is created. All of the team members contributed the use cases and came up with the different ideas.

1. CRC Cards

|  |  |
| --- | --- |
| Player | |
| Username  Password  Level  Type | Character  Score |

|  |  |
| --- | --- |
| Score table | |
| Score  Calculate  Max  Id(time-stamp) | Player |

|  |  |
| --- | --- |
| Level | |
| Load from file | Dungeon  Match  Coins |

|  |  |
| --- | --- |
| Dungeon | |
| Size  Name  Number of coins | Character  File  Level |

|  |  |
| --- | --- |
| Character | |
| Position  Gold coins collected  Movement  Track collected coins | Player  Score |

|  |  |
| --- | --- |
| Tile | |
| Type  Visibility | Dungeon |

|  |  |
| --- | --- |
| Match | |
| Check  Victory  Condition | Score |

1. 2. Process of Creating CRC Cards

CRC cards useful in OO paradigm. Cards represent class of objects, its behavior and its relationships. There is no universal syntax for CRC cards. [4] According Alshehri and Benedicenti from University of Regina creating CRC cards are the way of measuring the quality of the your product and ensuring about design simplicity.

After development of the use cases, after sprint 1 in the team meeting, investigation of CRC cards is done by team. Participants have read and familiarized themselves the documentations. For the next meeting, each team member came up with a CRC card case based on their use case. Each CRC card is converted from the use cases. Domain experts are leading to team to make correct assumptions.

In the brain storming session team decided, each player should have four different responsibilities: Username, password, level and player type (single or multiplayer). Each player should have a relationship with character and the score. After score relationship is added for player, CRC card of score table is created. It has an id and for each player a max score should be calculated. Responsibilities are decided based on which class will affect the system more?

Level CRC card should have collaboration with match, dungeon and coins. Classes are created with the idea of "what happens when...” like “What is going to be happen when all the coins are collected?” Dungeon CRC card should have size, name and number of coins. For the character card “How position will change based on movement?” Tile type is defined like wall, passage and extra items. Tile should have a relationship and collaboration with the dungeon. For the last CRC card new ideas (“What is going to happen to a match after win condition” and “How lose conditions are specified?”) are thought. Every match should have a relationship with score.

At the end we had a live prototyping, identified the holes in the requirements and now we can continue with the other uses cases by using these CRC cards.

1. Uses Cases v2
   1. Use Case 1 – Login with System and Client Relationship

UC1-1: Use Case: Log In

UC1-2: Author: QZ, XF

UC1-3: 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.

|  |  |  |
| --- | --- | --- |
| **Actor Actions** | **Client System Actions** | **Server System Actions** |
| 1. **Open the game page.** |  |  |
| 1. **Input the username and password.** |  |  |
| 1. **Press “OK” button.** |  |  |
|  | 1. Sends the username and password to server. |  |
|  |  | 1. Check if the username already exists in the membership database and validates the password. |
|  |  | 1. Sends response. |
|  | 1. Receives and parses the response. |  |
|  | 1. 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.

* 1. Use Case 2 – Registration with Server and Client Relationship

UC2-1: Registration

UC2-2: Author: AG

UC2-3: Sprint3

UC2-4: Purpose: Player registers to the game

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

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

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

UC2-6: Cross Reference: [R1.1](#R1_1), [R1](#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.** |  |  |
| 1. **Input the username, the email and the password.** |  |  |
| 1. **Press “OK” button.** |  |  |
|  | 1. Sends the username, the email and password to server. |  |
|  |  | 1. Stores the username, the email and the password to system’s database. |
|  |  | 1. Sends response. |
|  | 1. Receives and parses the response. |  |
|  | 1. Jump to the log in page. |  |

UC2-9: Alternative flow of events:

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

UC2-9: Exceptional flow of events:

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

* 1. Use Case 3 – Main Menu with Server and Client Relationship

UC3-1 Use Case: Player Choose Menu

UC3-2 Author: XF, QZ

UC3-3: Sprint3

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), [R2.1](#R2_1), [R2.2](#R2_2), [R2.3](#R2_3), [R2.4](#R2_4), [R2.5](#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.

Single/Multiplayer

|  |  |  |
| --- | --- | --- |
| Actor Actions | Client System Actions | Server System Actions |
| 1. Begins when player click single player of multiplayer button |  |  |
|  | 1. Sends button request to the server |  |
|  |  | 1. The server initiates the player’s session. |
|  | 1. Jumps to level selection web page. |  |
| 1. The user selects a match to play in. |  |  |
|  | 1. The client sends a request to the server |  |
|  |  | 1. The server responds with dungeon data. |
|  | 1. Client receives the response and creates a graphical representation of the data. |  |

Score

|  |  |  |
| --- | --- | --- |
| Actor Actions | Client System Actions | Server System Actions |
| 1. Begins when player click score button |  |  |
|  | 1. Sends button request to server |  |
|  |  | 1. Retrieve the scoreboard from database. |
|  | 1. The user is prompted to score table web page and shows the data. | 1. Send response |

Tutorial

|  |  |  |
| --- | --- | --- |
| Actor Actions | Client System Actions | Server System Actions |
| 1. Begins when player click Tutorial button |  |  |
|  | 1. The user is prompted to tutorial page. |  |

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.

* 1. Use Case 4 – World Creation with Server and Client Relationship

UC4-1 Use Case: World Creation

UC4-2 Author: SK

UC4-3: Sprint3

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](#R3), [R4](#R4), [R4.1](#R4_1), [R4.2](#R4_2), [R4.3](#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 | Client System Actions | Server System Actions |
| 1. Begins when player selects a match. |  |  |
|  | 1. Sends button request to server |  |
|  |  | 1. Dungeon is created with rooms, passages and coins. Server saves this information on its memory. |
|  |  | 1. Sends response with visible area by the player |
|  | 1. Receives and parses response |  |
|  | 1. The client draws the graphical representation of the dungeon. |  |

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. 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), [R5.1](#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 |  |  |
|  | 1. Sends movement request to server |  |
|  |  | 1. Checks whether or not the player can move in that direction. |
|  |  | 1. Sends response |
|  | 1. Receives and parses response |  |
|  | 1. 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.

* 1. Use Case 6 – Coins with System and Client Relationship

UC6-1: Use Case: Picking up Gold coins

UC6-2: Authors: AG, MJ

UC6-3: Sprint3

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 in 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), [R5.2](#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 | Client System Actions | Server System Actions |
| 1. Player begins to move to a gold coin. |  |  |
|  | 1. The client sends a request to the server. |  |
|  |  | 1. The server receives the request. |
|  |  | 1. The server validates the action. |
|  |  | 1. The server increments player’s gold coin collection. 2. The server updates gold coin location. 3. Sends response |
|  | 1. Receives and parses the response 2. Updates the graphical representation. |  |
|  |  |  |

* 1. 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: Sprint3

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), [R5.3](#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 |
|  |  | 1. Decides which tiles are visible to the character |
|  |  | 1. Responds to move request, including the current state of tiles now visible to the player character |
|  | 1. Receives and parses the response |  |
|  | 1. Adds the updated and/or new tiles to its memory |  |
|  | 1. Updates the graphical representation |  |

* 1. Use Case 8 – Exit with System and Client Relationship

UC8-1: Use Case: Leaving the Game

UC8-2: Authors: AG, MJ

UC8-3: Sprint3

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), [R5.4](#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 exit 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 exit button. |  |  |
|  | 1. Sends request to the server. |  |
|  |  | 1. Receives the request. |
|  |  | 1. Removes the player’s character. |
|  | 1. 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.

* 1. Use Case 9 – Winning Condition with System and Client Relationship

UC9-1: Use Case: Winning condition

UC9-2: Authors: AN

UC9-3: Sprint3

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

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, unlocks the new level 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 unlocked level with an option to return to main menu.

UC9-6: Cross References: [R6](#R6), [R6.1](#R6_1), [R6.2](#R6_2), [R7](#R7)

UC9-7: Actors: Player

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Actor Actions | | Client System Actions | Server System Actions | |
| 1. The player won the game. | |  |  | |
|  | 1. Sends the updated result to the server. |  | |
|  |  | 1. Receives the request. | |
|  |  | 1. Update the scoreboard. 2. Unlock a new level 3. Sends the result screen. | |
|  | 1. Shows the result screen 2. Shows the main map with unlocked level 3. Returns to main menu. |  | |

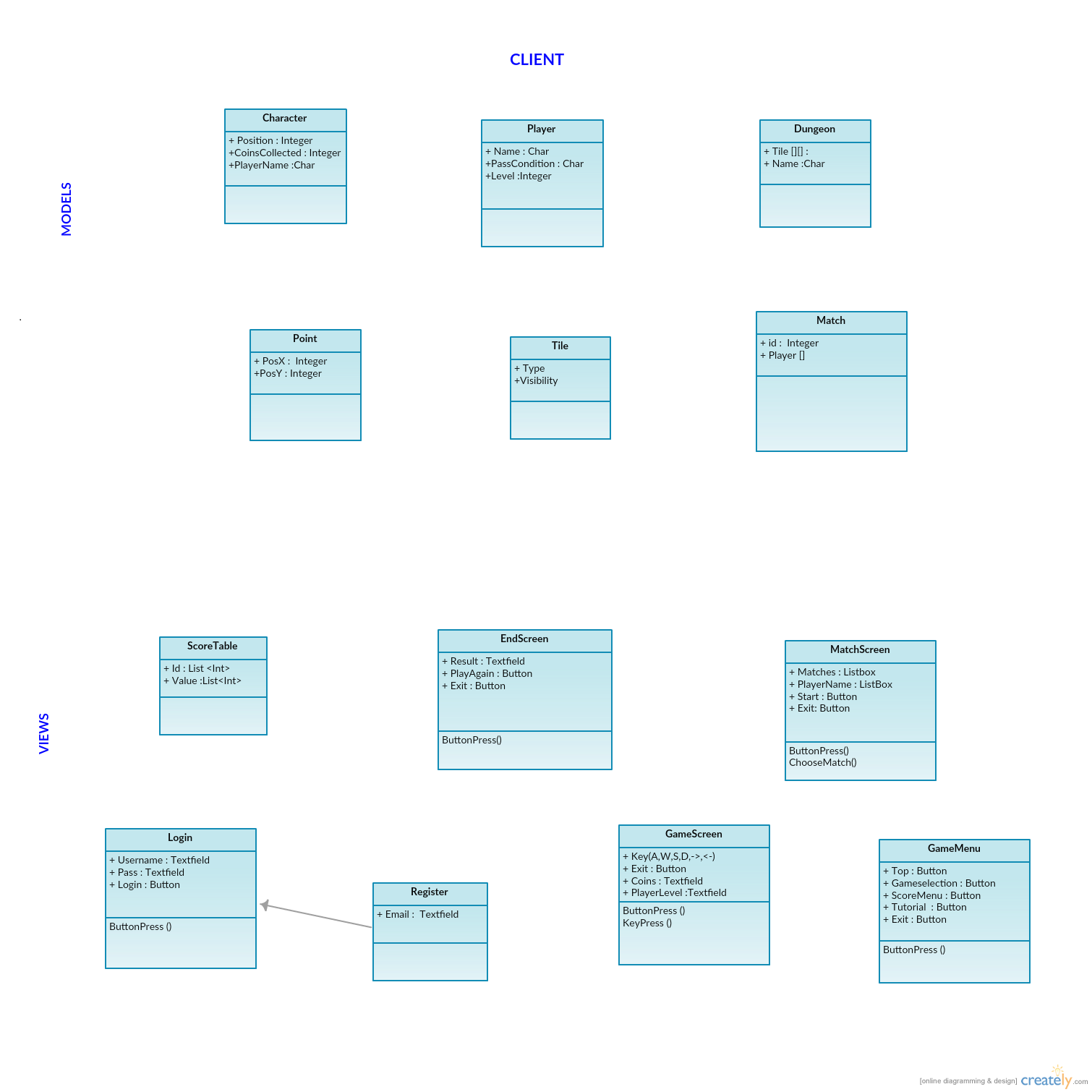
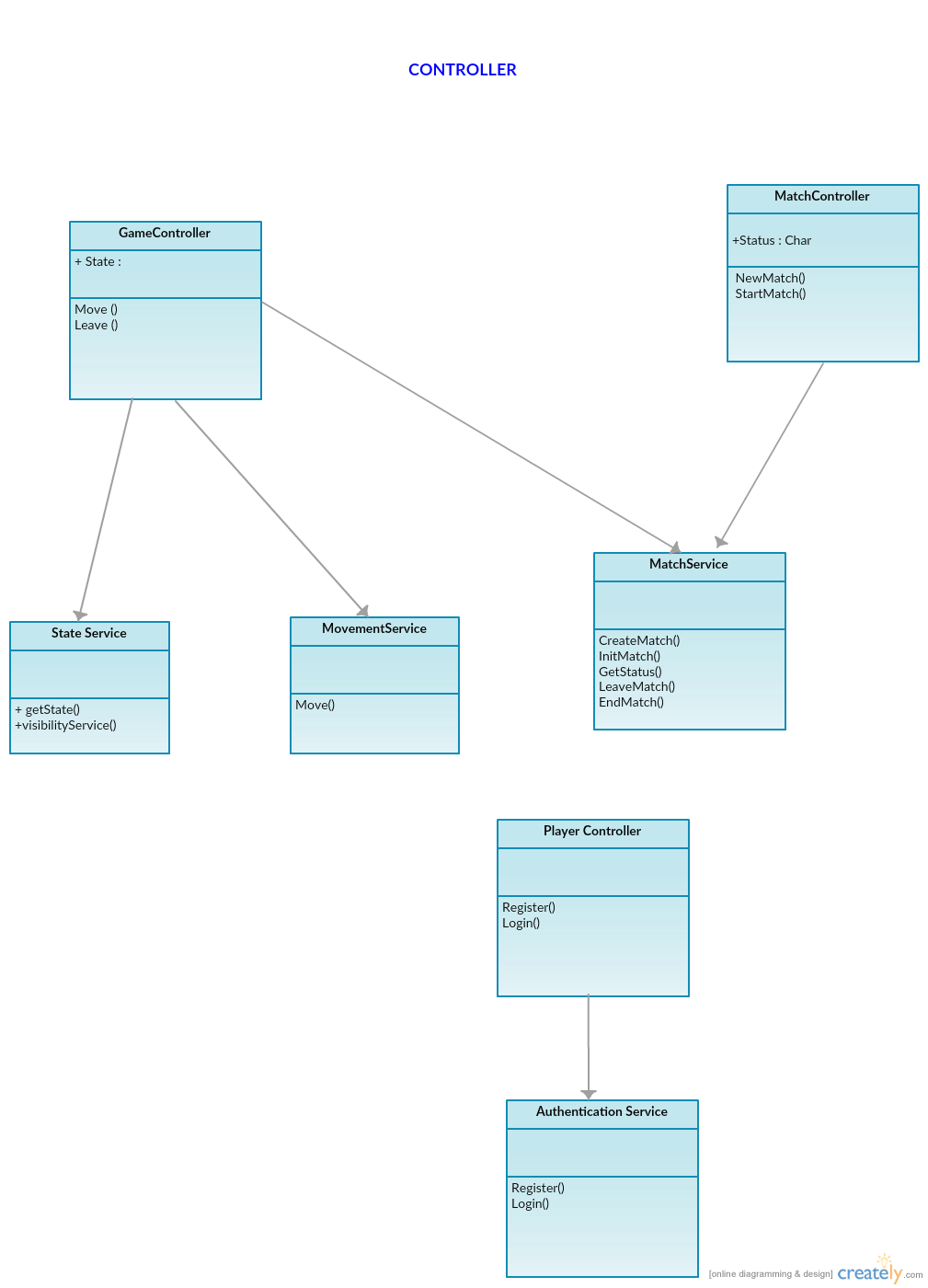
UC9-11: Exception flow of events:

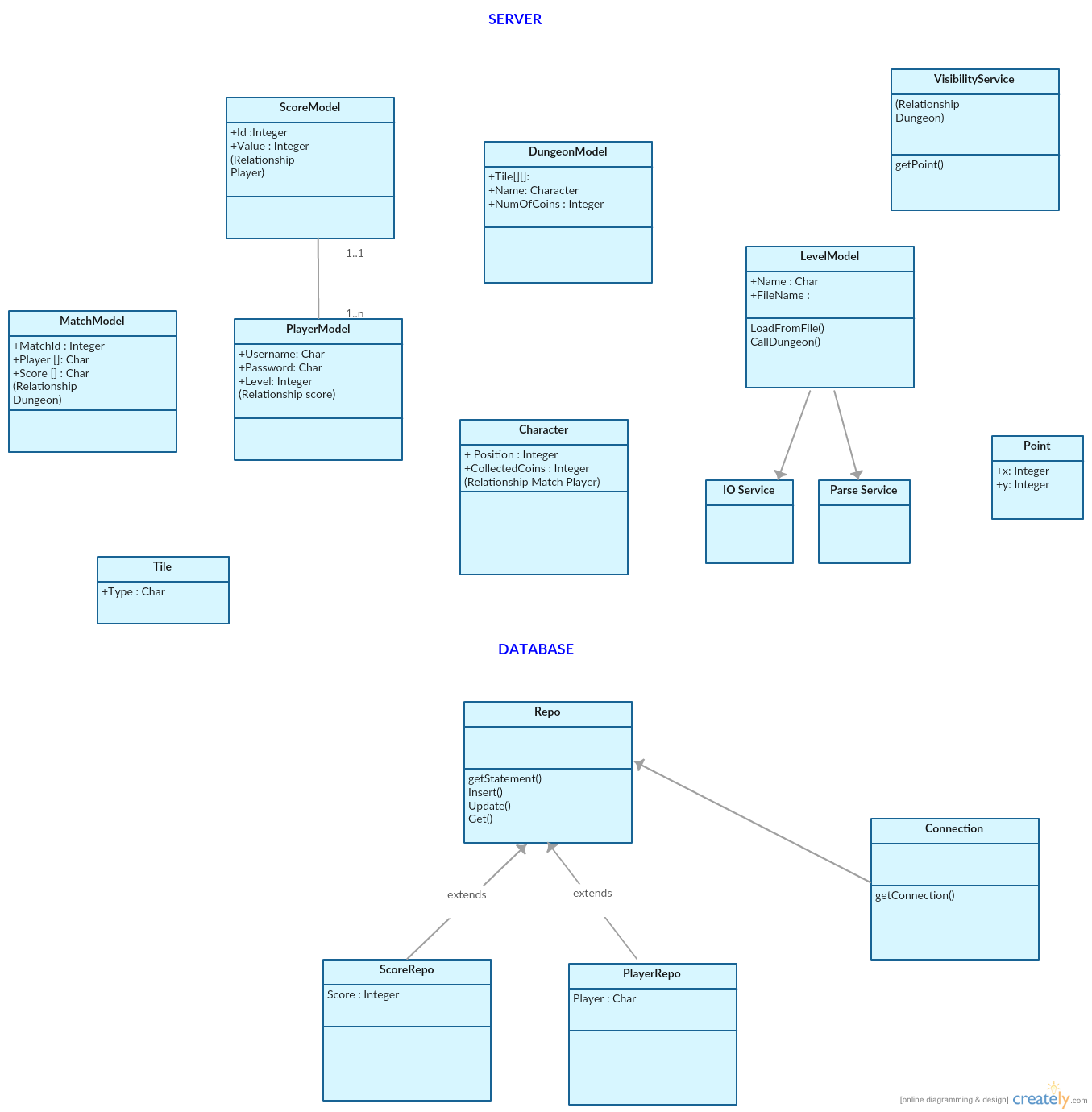
Step1: If player loses the connection to the network after winning the game, client wait for a specified amount of time. If player comes back, client sends request to server and steps 2-8 will carry on. If the server hasn’t got any request within the time limit, then server will remove the player’s character.

1. UML Diagrams v1

There are two sides in the system: server and client side. The server uses MVC, where the JSON results are the view. Models on the client have attributes from the uses cases. A database keeps track of data that needs to persist between session and you can see how we envisioned using the Repository pattern (“Repo”) to access the database.

You can also see our plan to write our game logic in *services*, such that they are reusable and separated from the controllers. This way the controllers can be very simple.

* 1. 4.1 Client Side
  2. 4.2 Server-side Controllers
  3. 4.3 Server-Side Models & Database



After the development of UML diagrams, system is ready for development. Now proper system components should be selected based on nonfunctional requirements. In third sprint,

Team discussed about what languages, components should be used for both front-end and back-end. How should MVC be implemented? Which database is more suitable for our requirements? How should this design be implemented and with what technologies?

1. Defining Non-Functional Requirements

System consists of 3 components: Presentation layer, Business Logic Layer and Database Layer

4.1 Database Layer has a MySQL database that contains information about players and score table (server side).

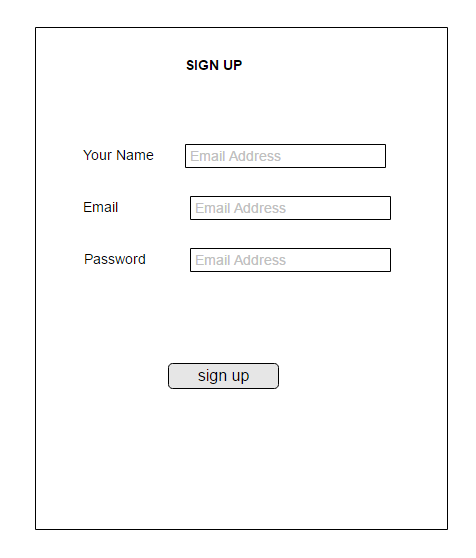
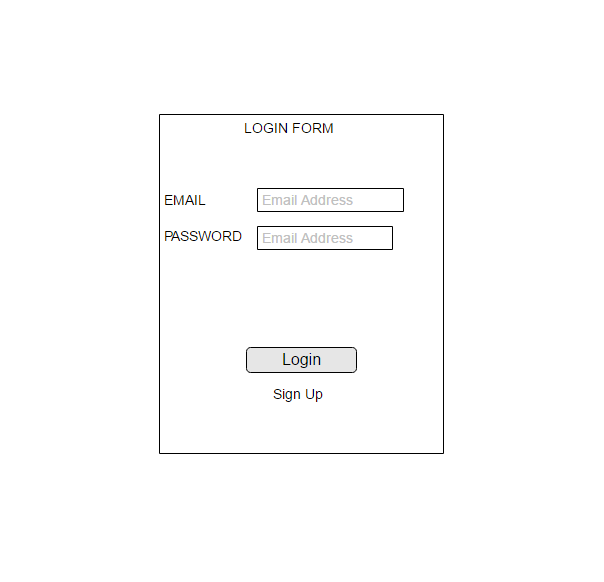
4.2 Business Logic Layer contains Web services and Models of game’s components. These components will be written in Java (server side) using the Jersey API framework.

4.3 Presentation Layer contains GUI (view) for the user and controllers that interact with the server via Web services. These components will be written in HTML5 & CSS3 and JavaScript (client side).

1. Layouts of Design
   1. 1. Mock up Design

In the 3.sprint meeting, design alternatives are discussed. Pages should be user friendly. User should understand how to handle game easily. And for team it should be easy to code. Firstly simple drafts of design are drawn to the paper. Then simple Mock Up tool is searched and these designs are conveyed into online mock up tool to show client. <https://moqups.com/>

Login Form Registration Form

Figure 1.Login Layout Figure 2.Registration Layout

Score Table

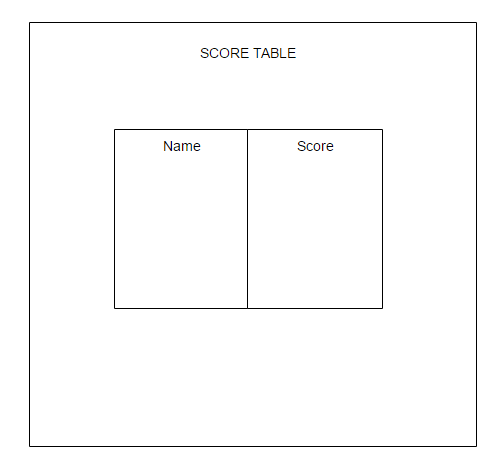
**

Figure 3. Score Table Layout

* 1. 2. Implementation of Design and Interface

// wil be Implemented with (json and template)

1. System Architecture
   1. 1. Three-Tier Architecture

The online multiplayer will be implementing by three-tier client–server architecture model. Three-tier architecture allows any one of the three tiers to be upgraded or replaced independently. The user interface is implemented on a desktop PC, which is client side of multiplayer game and uses a standard graphical user interface with different modules running on the application server. The relational database management system on the database server contains the computer data storage logic, which is implemented using json and mysql.

The three tiers in three-tier architecture are:

**Presentation Tier**: It includes the top level and presents the information related to services available on client side. This tier sends data to other tiers to communicate. For ex: sending result of game to database. All communication with the Presentation layer is done through Web Services.

**Application Tier**: Also called the logic tier or business logic, this tier is responsible for controlling application functionality by performing detailed processing.

**Data Tier**: Houses database servers where information is stored and retrieved. The Data Layer provides persistence for the system and all communication is done through SQL queries and views. Data in this tier is kept independent of application servers or business logic. For ex: storing score, storing records of players

* 1. 2. Design Pattern

The design pattern associate with our multiplayer game is MVC (Model View Controller) design pattern. The MVC design pattern clearly separated the web application’s behavior, presentation and control. The modularity of this design pattern allows for easier code reuse, more centralized control, bugs easier to track down and code easier to modify. The client will be running on presentation layer. The presentation, or view, of the multiplayer game will be implemented in HTML, CSS and Javascript. The database layer includes json files and mysql.

The business tier includes the model classes of multiplayer game.

Apart from that, the system needs to satisfy any functional, non-functional or aesthetic needs in a software system. The Dungeon of Doom follows the three-tier architectural style, which includes presentation tier, business tier, and data tier. The following is a simple description of what will be included in each of the tiers:

Presentation Tier

Business Tier

Data Tier

**Figure 4. Three Tier Architecture**

**Presentation Tier:** used to present the information to the player.

**Business Tier:** used to implement the logic of the system

**Data Tier:** To storing the data and other external services that the system may use.

1. 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.

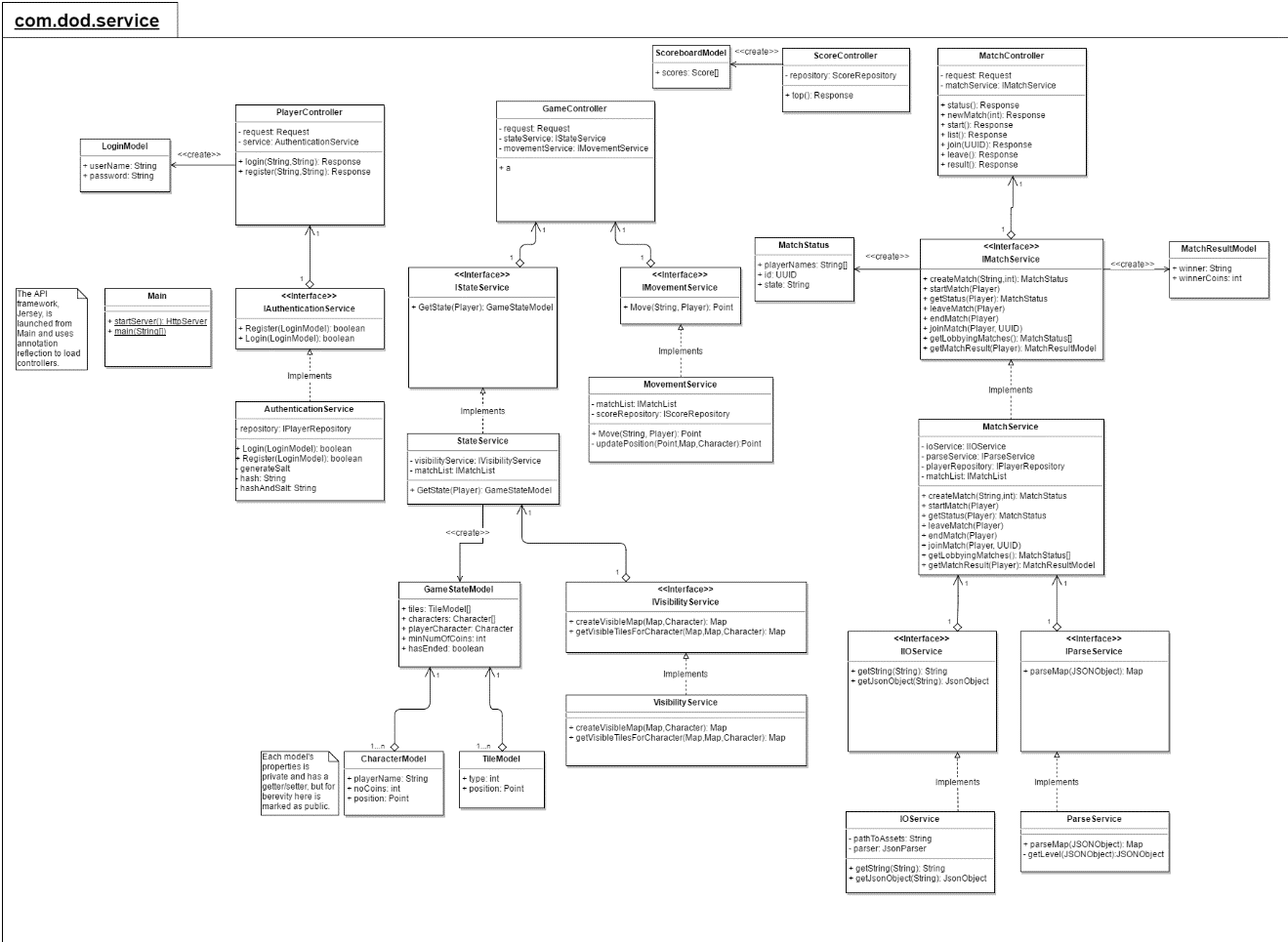
* 1. Domain package

This package contains our domain models and database layer. You can see how we used interfaces to abstract the repositories for testing and also 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.



* 1. 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.



* 1. 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.



1. User Guide

//will be written

References

1. ISTQB Exam Certification. (n.d.). What is Component testing? Retrieved 11 26, 2016, from ISTQB Exam Certification: http://istqbexamcertification.com/what-is-component-testing/
2. Kaner, C. (2006, Novemer 17). Exploratory Testing from Florida Institute of Technology, Quality Assurance Institute Worldwide Annual Software Testing Conference, Orlando, FL. Retrieved from Kaner.com: http://www.kaner.com/pdfs/ETatQAI.pdf
3. Writing effective use case (2016) Available at: https://www.visual-paradigm.com/tutorials/writingeffectiveusecase.jsp (Accessed: 10 December 2016).
4. Introduction to CRC cards (no date) Available at: http://www.openloop.com/education/classes/umn\_98/umn\_cpp/crc.htm (Accessed: 10 December 2016).
5. Cockburn, A. (no date) *Humans and Technology*. Available at: https://www.infor.uva.es/~mlaguna/is1/materiales/BookDraft1.pdf (Accessed: 11 December 2016).
6. Prioritizing CRC cards as a simple design tool in extreme programming - IEEE Xplore document(2016) Available at: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6567820 (Accessed: 11 December 2016).