Team 16 - Magic Realm

By Michael Hum, Tony Cheng, Anvar Gazizov

1 - Introduction

Magic Realm is a fantasy-adventure board game first published by Avalon Hill in 1979. It is more complex than many other board games, is somewhat similar to a role-playing game, can be played either solo or with a group of up to 16 players at a time, and can last for several hours.

1.1 - Motivation

This document includes a brief introduction to the project, functional and nonfunctional requirements of the software as outlined by the correction grid, as well as use cases and assumptions drawn up for the software.

The goal of this project was to convert the board game to a functional, networked piece of software while trying to respect the rules of the game as closely as possible.

1.2 - Terminology

Client, Player - A user of the software.
Chit - A token in the game that represents and contains information for some object in the game, such as a weapon, an action a player can use, etc.

2 - Game Rules and Client Requests

This section contains the rules and the game mechanics that the functional requirements are based on. It also contains the client requests which were adressed by the functional requirements. The rules which are not related to functional requirements are omitted for the purpose of being concise. For an in-depth version of the rules please refer to:
http://people.scs.carleton.ca/~jeanpier//304W15/magic%20realm/magicRealm2ed.pdf

ID	Rule
R-1	Certain characters posses special abilities.
R-2	Rules pertaining to combat.
R-3	The player can select from multiple characters.
R-4	The board consists of hexagon tiles which are made up of several clearings. These clearings contain items, such as treasures and armour.
R-5	The game contains monsters, which are considered enemy entities.
R-6	Characters are only allowed to move from one clearing to another if there is a path between the two clearings.
R-7	A player can hide his location from others.
R-8	Natives are entities which can be cooperative, or combative.
R-9	Characters can block other characters from passing the clearing which they are in.
R-10	The turn order as well as all other undetermined mechanics in the game are determined by rolling dice or completely at random.
R-11	A player acquires fatigue counters during certain activities. These can be counteracted with rest chits.

R-12	Tiles enchanted with magic change color.
R-13	Some treasures affect the way the game is played

ID	**Client Request**
CR-1	The game must be played over a network.
CR-2	The player must be able to pick his/her characters starting position.
CR-3	The player is able to manually manipulate some of the aspects of the game in cheat mode.

3.1 - Requirements

This section contains what features must be implemented. Each requirement contains a unique identifier, description and the traceability of that requirement. The source of traceability may either be from the official game rules(2nd edition),client requests or assumptions.

Description	**ID**	**Traceability**	**Test**
Display correct tiles of the board.	FR-1	R-4	Launch the game and validate placements
The game must be played over a network.	FR-2	CR-1	Launch a server, and connect to it with a client

All of the players must have a refreshed version of the game model based on the most recent move made.	FR-3	CR-1	When the players all end their turn, look for changes in the board (especially the referesh)
Initial character selection offers two or more distinct characters.	FR-4	R-3	Launch the game, verify at character select
Initial manual set-up of dwellings and ghosts.	FR-5	CR-3	Launch the game, verify dwellings and ghosts are placed at the correct chits.
Initial manual set-up of sound and warning chits.	FR-6	CR-3	Launch the game, verify chits are placed at the right tiles
Displaying sound and warning chits.	FR-7	R-5	End a turn in a tile with a sound or warning chit, look for the display of the chit at sunset
Player is able to select starting location.	FR-8	CR-2	Verify through character selection, and check if location is the desired one.
Character legal moves within the same tile.	FR-9	R-6	Do actions for two characters on the same tile, verify the actions
Character legal moves across tiles.	FR-10	R-6	Attempt to make legal moves and verify that they are done. Attempt to make illegal moves and verify that they are not done at end of daylight.
Captain's special abilities.	FR-11	R-1	Select captain and verify.
Amazon extra move.	FR-12	R-1	Select amazon and verify

Hiding and unhiding and rehiding.	FR-13	R-7	Covered by unit test: Check if after a hide phase is recorded, player is hidden if 6 is not rolled, and not hidden if 6 is rolled
Manual dice set up for hiding.	FR-14	CR-3	Start game in cheat mode, verify manually rolling is available
1 round of combat between two characters: 1 death.	FR-15	R-2	Covered by unit test Creates two characters and simulates one round of combat between them where a kill occurs. Checks if the killed player is respawned and loses gold.
Selection of fight and move counters for combat.	FR-16	R-2	Verify functionality when starting a combat in the game
Selection of armor for combat.	FR-17	R-2	There's a unit test for damaging armor. To test selecting armor, start a combat with someone in the game and verify armor is able to be selected.
1 round of combat between two characters: no death.	FR-19	R-2	Covered (partially) by a unit test Test checks if two misses are resolved correctly, but a part of it is if no character dies.

Random order of player turns.	FR-20	R-10	Finish a turn for all players, verify phases are done in random
Separate display for multiple 'things' in one clearing.	FR-21	R-4	Click on a tile, verify contents of tile are in the display
Alerting weapons.	FR-22	R-2	Covered by unit test: Test checks if after an alert phase, a character's weapon is alerted
Alerted and non-alerted weapons.	FR-23	R-2	Covered by unit test: Checks if an alerted weapon can be used, and that the sharpness is greater than an unalerted weapon
Fatigued counters.	FR-24	R-11	Finish the combat phase after actual combat is done. Verify characters have fatigue counters after
Wounded counters.	FR-25	R-2	Covered by unit test: In same test as damaging armor. Checks if after damage is done to character, character is wounded.
Resting fatigued counters.	FR-26	R-11	Covered by unit test: create a character with fatigue counters. Do a rest phase and check if

			character is not fatigued and wounded.
Manual monster roll (for treasure appearance).	FR-27	CR-3	Check if after game is launched, user is prompted to enter a manual roll (this is for which monster appears at which site)
Manual specification of treasure's notoriety and fame.	FR-28	CR-3	After treasure is discovered, verify a prompt for manual roll appears
Treasure appearance.	FR-29	R-4	Launch the game, verify gold chits are present in certain tiles
Treasure searching.	FR-30	R-4	Covered by unit test: Creates a treasure at the player's clearing, does a search phase for the player. Verifies player receives the treasure if the correct roll is made.
Treasure looting (with manual specification of roll).	FR-31	CR-3	Discover a treasure in the game, and verify treasure can be manually rolled for
Multi-round combat.	FR-34	R-2	Verify in the game when two characters do combat with each other. Each character is able to attack and resolve their attack.
Random placement of all chits.	FR-36	R-10	Launch game, verify chits are placed at different tiles everytime.

Random rolls.	FR-37	R-10	Check if game has different values for monsters appearing, moving, and other roll related moves.
Character blocking.	FR-38	R-9	Covered by unit test
Selling to natives.	FR-39	R-8	Move to a leader's tile, start a trade phase and check results
Buying from natives.	FR-40	R-8	Move to a leader's tile, start a trade phase and check results
Natives blocking.	FR-41	R-8	Move to a native's tile, check if the native blocks that turn
Monsters appearing according to monster roll.	FR-42	R-5	Check if a new monster spawns at daylight
Monsters roaming from clearing to clearing.	FR-43	R-5	Play the game, monsters have a chance to move from their starting clearing once spawned.
6+ distinct monsters activated by 6 different monster rolls.	FR-44	R-5	Pass by several days in the game, test if there are at least 6 monsters present on the board
Missile attacks.	FR-45	R-2	During combat, pick a weapon to use. Verify missile hit does damage.
Clearing coordinates are relative to tiles.	FR-46	R-4	Verify chits placed on clearings are also rotated when tiles are rotated, and locations correspond with tile

Several combats within the same day.	FR-47	R-2	Have multiple clients do combat with each other, and verify all combat results are correct.
Tiles change color when enchanted.	FR-48	R-12	Covered by unit test: Phase created to enchant a tile, checks if tile is enchanted. To check for actual color change, enchant a tile in the game and verify
Some treasures have a special effect on the game.	FR-49	R-13	Covered by unit test: Discover a phase treasure, check if player receives an extra phase

3.2- Assumptions

Throughout the process of developing the game we took the liberty of making some assumptions about the game and the way it was to be played. The assumptions are listed below with and id number used for further reference and a justification.

ID	Assumption	Justification

A-1	It is assumed that the user is going to follow through with the action the dialog requires from the user.This means that the user should complete all of the actions before closing the dialog.	The handling of this event was secondary to developing a playable game which resembles the rules provided.
A-2	Great treasures are assumed to be treasures which have notoriety of greater than 15 or fame greater than 15 or if the gold value is greater than 30.	The rules were unclear in describing great treasures, so a decision was made to approach great treasures in this way.
A-3	Blocking is automatic for any entity.	This was a secondary feature and was simplified in order to complete the core features of the game.
A-4	With respect to trading, it is assumed that the players will respect the rules of the game.	Restricting the players was omitted in order to focus on game functionality.
A-5	The combat encounter step is simplified. Players may choose to activate or run away if they have the action chits, but this happens before monster luring and doesn't penalize the user in any way.	This way of making the game involved the user in more decisions with respect to combat, we believe that this makes for a more interesting game.
A-6	Natives and Denizens get attack and maneuver values assigned at random.	The addition of these attack values was a secondary feature and was simplified in order to complete other features.
A-7	Secret passages and hidden paths are the same thing.	We felt that the difference between the two in the rules was trivial and chose to focus on more essential features.

A-8	Trading between characters is omitted.	Restricting the players was omitted in order to focus on game functionality.
A-9	All items are activated at all times.	The functionality of this feature was not a priority.
A-10	Combat ends in either one or no deaths.	Our understanding of the game rules prevented two deaths from happening; combat according to the rules isn't executed at the same time; rather one person attacks first and then defends.
A-11	Combat always uses the time of the chits.	Adding in weapon times was a secondary objective, and action chits already had the times added.
A-12	Rest activity is simplified, if user selects a wounded chit the user becomes fatigued and if it selects a fatigued chit it goes back to normal.	This rule is easier to follow and doesn't deviate too much from the actual rules, which would require more time to implement when there were higher priority tasks to be completed.
A-13	Berserk chit effect stays active once it is used once.	This rule is easier to follow and provides a basic implementation when there were other tasks to work on.
A-14	During sunset if the monster is prowling they have a chance to move to the adjacent clearing in their tile. If a player is in the tile, however, the monster will move to that location in the tile.	This was the interpretation of the official rules on monster combat.

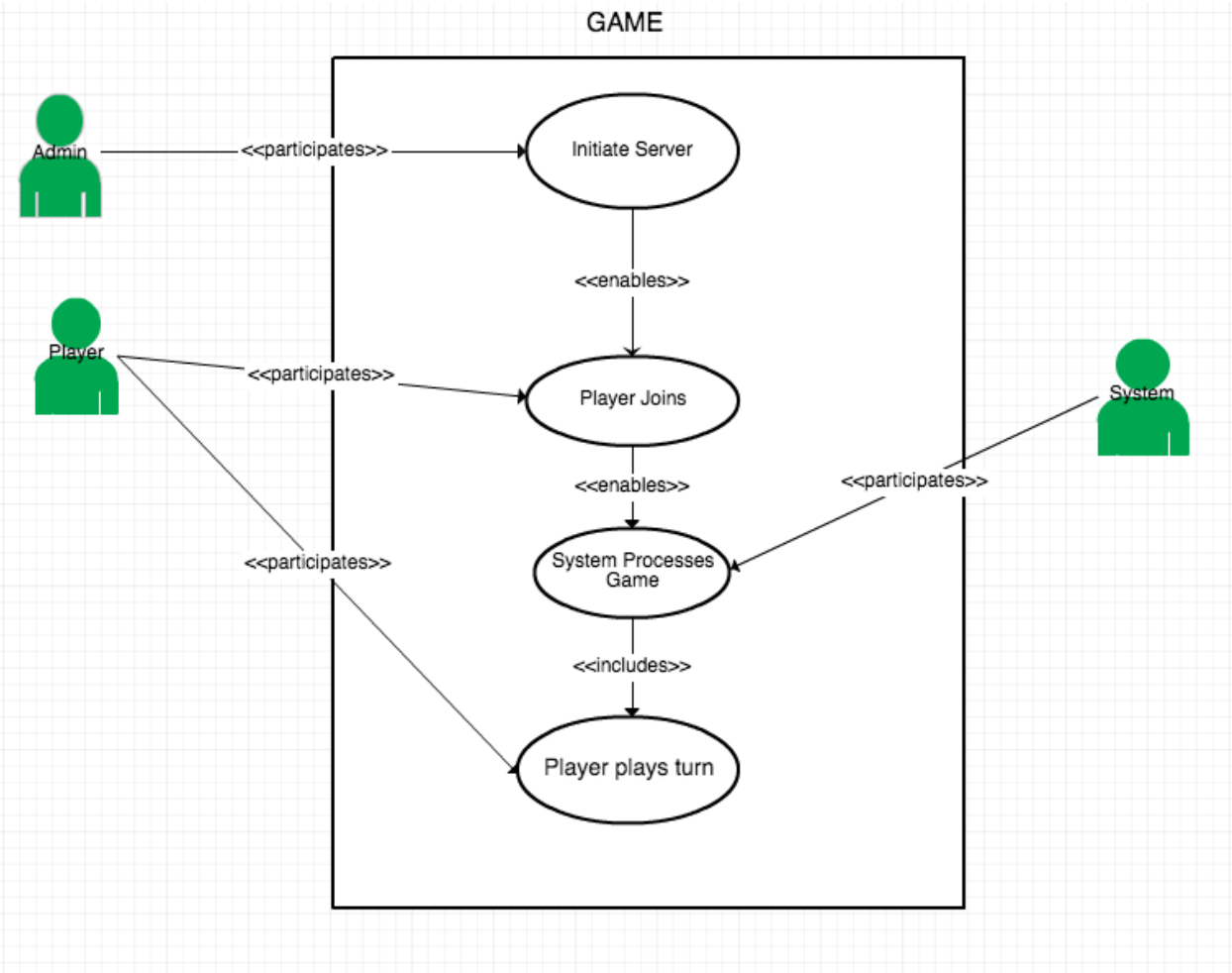
A-15	Fatigue step is calculated at the end of the combat, instead of after every round.	Combat was initially designed as one combat round only. When this evolved, re-factoring the fatigue step was considered a secondary priority.

4 - Use Cases

A use case is a list of steps that define the interactions between an “actor” and a system in order to achieve a goal. In the case of this piece of software, actors are the users of the software and the system is the game.

4.1 - Use Case Diagram

The following diagram corresponds to the use cases in section 4.2 Actors are the people.



4.2 - Use Case Description

UC-01	Admin Initializes Server
Description	Admin initializes the server with a specified number of players, preparing it to host a game.
Actors	Admin
Triggering Event	Admin chooses to run initialize the server.

Pre-Condition	N/A
Main Sequence	1.Admin runs server. 2.Admin specifies the number of players for the game. 3.Server waits for players to join.
Post-Condition	Server is running.
Resulting Event	Server now waits for players to join.
Alternative Scenarios	N/A
Traceability	FR-3,FR-3

UC-02	**Player Joins Game**
Description	Player connects to the server to join a game.
Actors	Player
Triggering Event	Player starts the application
Pre-Condition	Server is running.
Main Sequence	1.Player opens the application. 2.Player selects character and victory points. 3.Player waits for the appropriate amount of other players to join the game.
Post-Condition	Player waits for other players to join the game. or If Player is the final player needed for the game the game has begun.
Resulting Event	Player is registered for the game. of If player is final player to join the game , the game has begun.
Alternative Scenarios	Player is the final player needed to begin the game: 1.Player opens the application. 2.Player selects character and victory points. 3.Board is set up and the game begins.

Traceability	FR-1,FR-4,FR-29,FR-36

UC-03	**System Processes Game**
Description	The process of the player playing magic realm.
Actors	System
Triggering Event	All of the players needed for the game join the game.
Pre-Condition	Game has begun.
Main Sequence	1.System determines order in which players will take turns for the day. 2. System determines time of day. 3. System sends turn message to players. 4. All players execute turn. 5. System increments the day counter.
Post-Condition	Day counter is incremented one day. or If day counter is on the last day of the game the game is finished.
Resulting Event	Day counter is incremented. or Game finished.
Alternative Scenarios	N/A
Traceability	R-10,CR-3

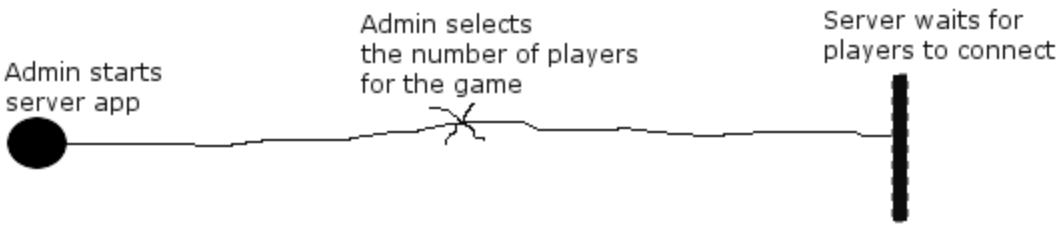
UC-04	**Player Plays Turn**
Description	A single turn for a player.
Actors	Player

Triggering Event	Player receives a turn message.
Pre-Condition	Player is ready for next turn
Main Sequence	1.Player receives turn message. 2.Player executes appropriate turn, according to the time of day in the turn message. 3.Player sends turn finished message.
Post-Condition	Players turn is finished.
Resulting Event	Player sends finish message.
Alternative Scenarios	N/A
Traceability	FR-9,FR-10,FR-15,FR-16,FR-17,FR-18,FR-19,FR-23,FR-30,FR-31,FR-34,FR-38,FR-39,FR-40,FR-41,FR-42,FR-43,FR-47

4.3 - Use Case Maps

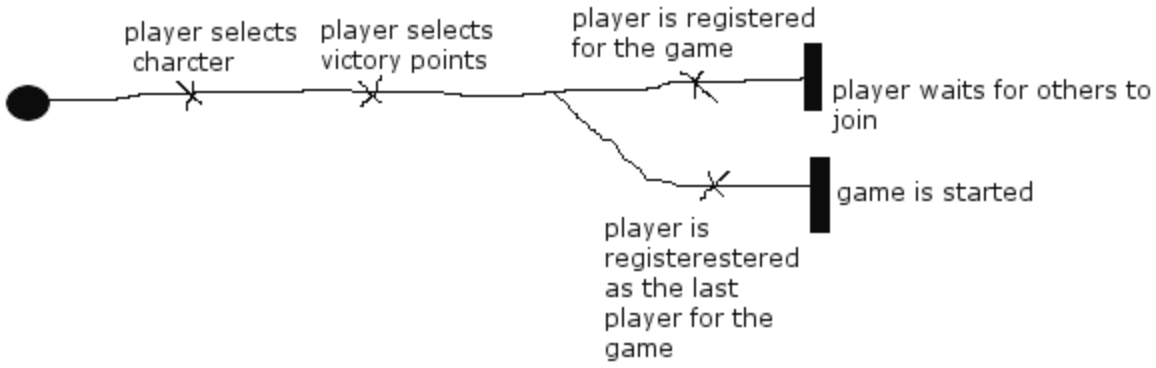
Use case maps correspond to their counterparts in section 4.2.

UC-1



ID	Responsibility
R-1	Support network based play.
R-2	Update the player gui each time the game is effected by a player.

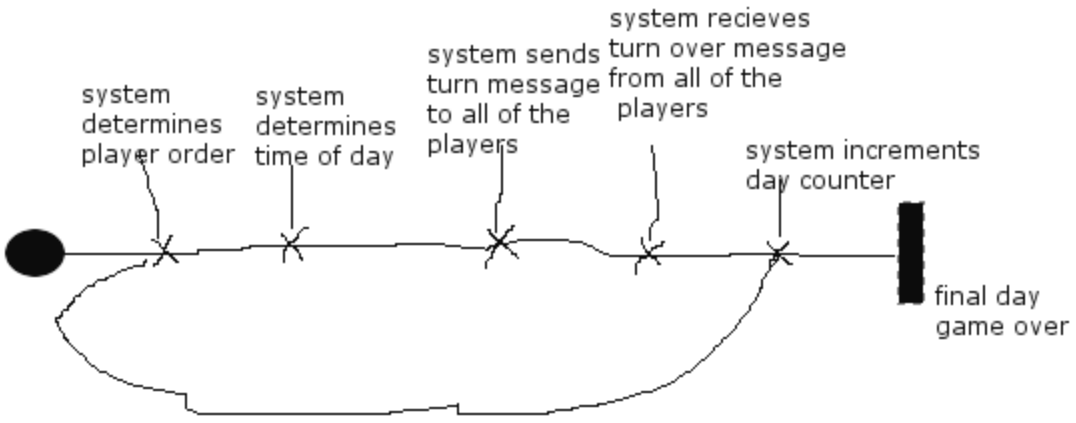
UC-2



ID	Responsibility

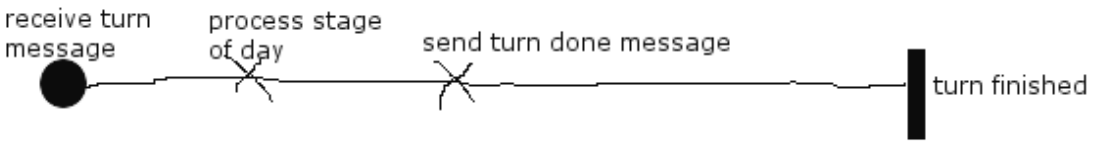
R-1	Provide a selection of multiple characters.
R-2	Provide a selection of victory points.
R-3	Set-up the proper board configuration along with other features of the game.

UC-3



ID	Responsibility
R-1	Random turn order for play at the beginning of each day.
R-2	Turn based play.
R-3	Game expires within a certain amount of days.
R-4	Game consists of different “times of day”.

UC-4



ID	Responsibility
R-1	Combat stage,

R-2	Daylight stage.
R-3	Birdsong stage.
R-4	Game mechanics.

5 - Design Decisions

This portion of the document outlines design decisions that were made with respect to the writing of the software itself (the implementation). Included in section 5.2 are class diagrams.

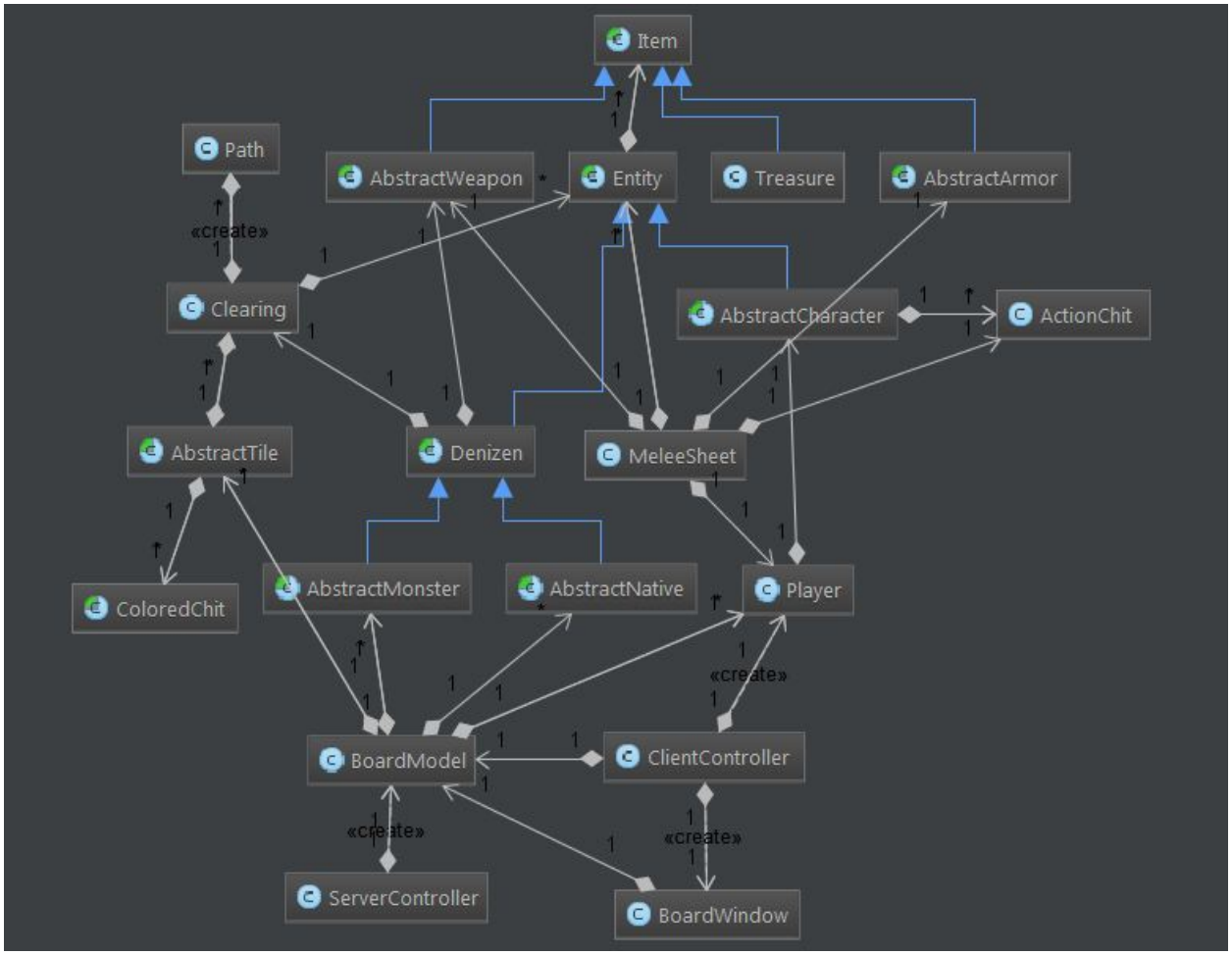
5.1 - Decisions

ID	Design Decision	Traceability
DD-01	**Write the software in Java.** We decided to use the Java programming language as it a popular language that was used in previous courses by all members of the group. In addition, some members also had more advanced knowledge of the language's ecosystem, including the Maven build tool as well as various third party libraries that would help with the development of the software (Junit, etc.).	Group decision.

DD-02	**Attempt to follow the idiom of "test-driven development" as much as possible.** Rather than develop blindly, we wrote write test cases for almost all parts of the game and ensured that they passed, giving us a safety-net for regressions and confirming that our code runs as expected.	Group decision.
DD-03	**Use MVC (model-view-controller) as the general design for the software.** MVC is a well-known pattern, taught to all students in previous years. It provides a clear description of the duties required by classes and provides much-needed separation of the different elements of the code, allowing easier readability, testing and implementation.	Group decision.
DD-04	**Use the marker interface pattern for character relationships.** Rather than having to store the relationship between a player and game denizen (friendly, etc.) for each and every relationship, instead we decided to use a marker interface for native factions and (if it were to be implemented) visitors that allowed us minimize the code written and simplify the logic for dealing with character relationships (trading, combat, etc.).	Group decision.
DD-05	**Use the factory pattern for the creation of entities.** The use of the factory pattern allowed us to create concrete implementations of the entities within the game without having to manually specify the exact class required. Since most of the game doesn't concern the concrete class, this simplified logic used throughout the game and in the character design process.	Group decision.
DD-06	**Use the state pattern for alert values.** The state pattern modifies an object's behaviour according to its state. Weapons can be in the alert state, in which they provide boosted combat effectiveness, or regular ('unalert'), which provides no boost. This was used while writing the combat code in order to provide seamless integration when checking weapon values (none of the actual checking in the combat logic itself).	Group decision.
DD-07	**Use the flyweight pattern for combat melee sheets.** Each entity in the game that can do combat needs a melee sheet that contains information for their combats with other entities. Rather than destroying and having to re-create these (which for denizens especially is pointless, since they are assigned randomly), the MeleeSheets class acts a flyweight that stores these melee sheets for entities that is then used by the application for all combats.	Group decision.

DD-08	**Use the strategy pattern for daylight actions.** The strategy pattern allowed for the quick, efficient implementation of the various actions players can make during daylight. Rather than having around each different action, each was split into a concrete strategy that only applies to the given context (i.e what the player chose). This decision improved the time to design these actions by a large margin; adding new actions was quick and painless.	Group decision
DD-09	**Use the builder pattern for action chits.** Action chits were created with very specific values in mind (according to the rules). They are immutable (those values don't change) which made the builder pattern a good option to make writing in these chits easier and clearer to understand.	Group decision.
DD-10	**Use Junit and Hamcrest for testing.** Testing the functionality of the code is largely done in part by the unit tests written (as stated above in DD-02). As stated, these unit tests (run with every build of the project) gave us a safety net that ensured what we wrote was correct, and any changes to the code would need to be addressed in the tests, which allowed us to spot bugs as they were created and quickly fix them.	Group decision.
DD-11	**Provide a separate dialog for logging output and player statistics.** We decided to provide a simple dialog that is simply a list of the character's current state (inventory, stats, etc.) to allow for easier reading. The game also provides a window that uses a simple, custom log appender to write the detailed logging to the user, so they may have an easier time understanding what is going on.	Group decision.
DD-12	**Use git and GitHub for our VCS.** Git is now the largest VCS tool, and github provides free hosting services for centralized repositories that allowed us to work together quickly and easily. Github repository link: https://github.com/Sacredify/3004-Magic-Realm-16	Group decision.

5.2 - Structural Model



A copy of the model and a more detailed version has been provided with this document.

5.3 - Client/Server Functionality

This portion of the document will outline which functionality resides on the server and which is handled by the client. The table reads logically with the sequence of events in the game, from top to bottom.

Client	Server
Connect to the game.	Start the game.
	Board generation + chit placement.

Enter actions for daylight (birdsong).	
Execute daylight for their own player.	
	Execute sunset phase.
Enter combat data (encounter/melee step)	
	Execute combat for all entities.
	Execute game over (if applicable).
	Execute sunrise and repeat.

6 - Object Specifications

Details for the objects can be found by looking at the source code. There is absolutely zero reason to include in a doc file details about methods, when one can look at the code and the javadoc written for that very purpose.