Magic Realm

Team 6

Couturier, Stefan

Lam, Ryan

Sayyeau, Christopher

COMP 3004

Date Due: April 8th, 2015

Table of Contents

1. Introduction p.3
    1.1 Motivation p.3
    1.2 Terminology p.3
2 Game Rules p.4
3 Requirements p.19
    3.1 Functional Requirements p.19
    3.2 Assumptions p.21
4 Use Cases p.22
    4.1 Use Case Diagram p.22
    4.2 Use Cases p.22
    4.3 Responsibilities p.25
    4.4 Unbounded Use Case Maps p.26
5 Design Decisions p.31
    5.1 Decisions p.31
    5.2 Structural Model p.32
    5.3 Client/Server Functionality p.33
6 Object Specifications p.33
7 Interaction Diagrams p.63

Note: Some diagrams may no longer be completely accurate due to minor code alterations.

1 - Introduction

Magic Realm was first released in 1979 by Avalon Hill. It was designed as a fantasy adventure board game by Richard Hamblen. It is a very complex role-playing war game that can be played from 1-16 players over the course of several hours. The rules were rereleased in a second edition in 1986 and a third edition was released by fans after the company went out of business in 1998.

1.1 - Motivation

This report will briefly cover our interpretation and implementation of this board game. It will outline the functional and non-functional requirements of the software, as well as its use cases and the assumptions made throughout its construction.

The goal of the project was to turn Magic Realm into a networked computer game. The official 3rd edition rules were followed as closely as possible to create the software.

1.2 - Terminology

A number of key terms are used throughout this report, which will be listed alphabetically in this section. Definitions are included, however if the term is covered in the Game Rules it will be covered in its own section (2).

Term	Definition
Attention Chit – The combat chit which has the character symbol on one side and is blank on the other.	
Cave Clearing – A clearing that is in a cave.	
Caves Tile – Any tile with at least one cave on it, in which treasure sites can appear.	
Character – One of the 16 adventurer types that can be played by a player. Described on the character cards.	
Chit – The smallest square counters, including combat chits, sound chits, warning chits, site chits, Lost City and Lost Castle chits, Monster Roll, Day (Turn), weather chits, visitor chits, and number chits.	
Counter – Any game piece other than the hex tiles, character cards, and chits.	
Denizen – A monster or native of the Magic Realm.	
Garrison Natives – Native groups that start out on the map at a dwelling, including the Order, Guard, Soldiers, and Rogues.	
Mountain Clearing – A clearing that has ridges drawn around it on a mountain tile; not every clearing on a mountain tile is a mountain clearing.	
Mountain Tile – A tile with at least one mountain clearing in which treasure sites can appear. Also includes the Deep Woods tile (even though there are no mountains in it) to round out the 5 Mountain tiles, along with the 5 Cave tiles, in which treasure sites can appear.	

Native – A member of a native group which can be hired, traded with, or battled.

Player – A person who plays one of the characters. A distinction between the character and the player is made in some rules.

Prowling – The denizens that are prowling each day are those in the row corresponding to the day's Monster Roll. Prowling denizens will move from the Appearance Chart to the playing board if summoned by game chits, and prowling monsters will move within their hex tile when characters and hired leaders finish their turn.

2 - Game Rules

The following extracts were taken from the official third edition game rules for Magic Realm. The individual parts have been labeled for traceability.

1.4.1. Each player plays the part of one character in the game. He controls that character's pieces and uses that character's counter to represent him on the map. The characters compete in accumulating "Great Treasures", usable Spells, Fame points, Notoriety points, and Gold. Before the start of play, each character records the number of points he needs in each category to win the game. He gains these points by owning or selling weapons, armor, horses, and Treasure cards, by reading runes at treasure sites or on magical artifacts and spell books to learn new spells, or by fighting and killing monsters, natives, or other characters.

1.4.2. The game is played in the clearings on the map. The characters start the game in the same clearings with the Dwellings and move from clearing to clearing by following the roadways. Each character shows where he is currently located by putting his character counter in that clearing.

1.4.3. Also on the board, but turned face down at the beginning of the game, are chits representing treasure sites and sounds and warnings of monsters that may arrive on the map. When characters end a turn in the hex, these chits are revealed. As characters move around the board, more and more of these chits will be revealed, letting the player know where monsters and treasures are to be found.

a. In some clearings on the board the characters can find treasure sites, represented by the gold site chits, which they can loot to take treasures (represented by treasure cards) or, at some sites, read runes to learn the spells on the Spell Cards there.

b. Hex tiles also have monster warnings (represented by the yellow Bones, Ruins, Smoke, Stink, and Dank chits) and sounds (represented by the red Howl, Flutter, Roar, Patter, and Slither chits). When characters enter a hex tile and reveal these chits, prowling monsters (as determined by the Monster Roll each day) may arrive on the board at the end of their turn and, in the evening, battle with the characters. Prowling monsters already on the board will move to a character's clearing and block unhidden characters from continuing their turn. Characters may be able to run away from or avoid the monsters, or they may be able to kill them and earn Fame and Notoriety points toward their Victory Requirements. On the other hand, the monsters may kill the character, forcing the player to start over at the Inn as the same or different character, forfeiting all his possessions, Fame, Notoriety, Gold, and discoveries.

c. At the Dwellings in the Valley or Woods tiles the characters will find groups of natives that they can trade with to sell treasures for gold or to buy the treasures and items kept by the group's leader. They may also attempt to hire the natives to help them search for treasure or battle monsters, characters, or other natives. The

characters can battle with the natives, hoping to kill them and earn Notoriety points, Gold bounty, and loot their abandoned possessions. On the other hand, characters may be killed by the battling natives, losing everything and having to start over. Unfriendly or enemy natives may block and battle a character unexpectedly.

d. Accompanying the native groups or at certain treasure sites, the characters may find chits representing Visitors, individuals who have treasure or spells for sale and will buy items from characters for Gold, Missions that the characters can carry out for gold, and Campaigns that the characters can undertake with various native groups as their allies.

1.4.4. Characters may cooperate with other characters to increase their chances of finding useful treasures and gaining Fame and Notoriety by battling monsters and natives. But beware! Characters may also attack and kill other characters to gain Notoriety and steal their gold and possessions.

1.4.5. At the end of the game, Victory Points are calculated by comparing the characters' Fame, Notoriety, Gold, Great Treasures, and Spells with the Victory Requirements recorded at the beginning of the game. Characters who have achieved their requirements or earned a positive score have won the game. The character who has achieved the most Points above his Requirements is the victor.

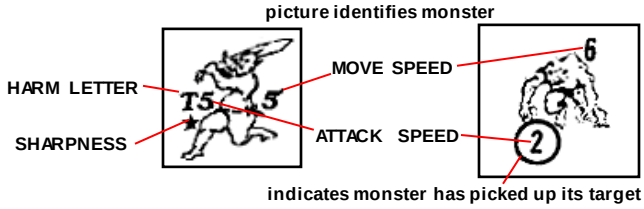
2.2 DENIZENS

2.2.1. The monster, native and visitor pieces represent the "denizens" who live in the Magic Realm.

2.2.2. The monster and native counters represent the monsters and humans pictured on them. The numbers, letters, and stars on these counters are combat values.

a. The letter on each counter defines the strength and harm inflicted by its attack, and any stars are "sharpness stars" that add to the damage it inflicts. The number with this letter defines his attack speed, and the other number on the counter defines his maneuver or move speed. These numbers show how much time it takes the denizen to complete an attack or a move. Lower numbers mean faster actions.

b. Each side of the counter shows different combat values (generally, the darker side is turned up when the monster or native is being more aggressive). Each monster or native always uses the values that are face up at the moment.



2.2.3. Monsters: Each monster counter represents the monster pictured on the counter. The size of each monster counter signifies the size and "vulnerability" of the monster - the largest counters are the largest, and toughest, monsters. The monster's "vulnerability" is the damage that must be inflicted on it to kill it, and is the same as its size and its move or flying strength. The largest counters are Tremendous monsters, the middle-sized counters are Heavy monsters and the smallest counters are Medium monsters.

a. The List of Monsters identifies each monster and lists its size, weapon and weapon length, method of attack, Fame bounty, Notoriety bounty, and whether it is armored, and shows the combat values on both sides of each monster. The Fame bounty and Notoriety bounty are used to determine the points a character gets for killing the monster.

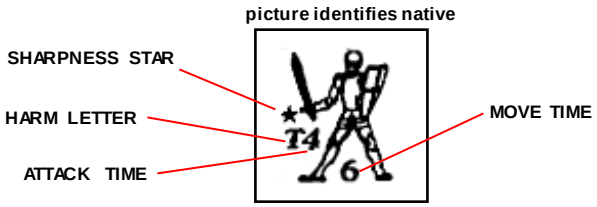
b. Most monsters have a weapon length of zero ("tooth/claw"), and attack by striking. If the monster has a weapon, head or club, its weapon and weapon length are listed. Demons and Imps attack using spells on their light side, but use tooth/claw on their red side.

c. The red side of the twelve Tremendous monster counters has a special meaning: it shows the monster's combat values when it picks up an enemy to rip him apart! Tremendous monster counters are normally turned red side down, and other monster counters (including club and head counters) are normally turned lighter side up. These counters turn over only during combat, and when combat ends they turn normal side up again.

d. The Dragons, Trolls and Serpents (including Vipers) are the only armored monsters. These monsters are protected by armor in all directions; every attack that hits one of them hits armor and loses one sharpness star. A monster's armor cannot be damaged or destroyed while the monster lives.

g. Five of the monsters have two counters: each Giant has an extra counter that pictures a club, and each Tremendous Dragon has an extra counter that pictures its head. Each extra counter moves with its monster, but it attacks separately, with its own combat values. The extra counters cannot be attacked; the only way to kill one of them is by killing the monster it belongs to.

h. Flying Values: Some monsters have "flying values" instead of move values (see the List of Monsters), and other denizens and characters can acquire flying values by magic. Flying values, like move values, consist of a strength letter and a move time. When a character or denizen has a flying strength that will carry him, he can use it to fly. Flying monsters always fly. They fly when they charge in combat, and when hired or controlled they fly when they move.



2.2.4. Natives: Each native counter represents the native pictured on the counter. The List of Natives identifies each type of native and indicates his Gold bounty, Notoriety bounty, weapon length, method of attack, weight, move strength, vulnerability, whether he is armored, and the basic "wage" that determines how much recorded Gold it costs to hire him. Each native has the weapon length, method of attack and armor implied by the picture on his counter. Except for the Knights, each native's move strength and vulnerability equal his weight. The Knights have Heavy weight but Tremendous move strength and vulnerability.

a. The natives are divided into nine groups. Each native counter has an identity code (or "ID code") that identifies him and the group he belongs to. The *Company* ("C" ID code) are the blue counters, the *Bashkars* ("B") and *Rogues* ("R") are red, the *Guard* ("G") and *Order* ("O") are gold, the *Lancers* ("L") and *Woodfolk* ("W") are green, and the *Patrol* ("P") and *Soldiers* ("S") are brown.

b. The ID code on each native counter identifies the native and his group. The first letter identifies his group, and the rest of the code identifies him within the group: the leader of the group has the "HQ" code, and his underlings are identified by numbers.

c. The leader of a native group is special – he is the only one who trades, and he is the only one who summons visitor/mission chits, even when the group is scattered. When he is killed (or hired), no one in that native group can trade or summon visitor/mission chits. The leader also has greater capabilities than the other members of his native group when hired.

d. The other numbers, letters and symbols on each native are his combat values. The harm letter and sharpness stars define the harm he inflicts when he hits. The number with the harm letter defines his attack time and the other number is his move time. The Table of Natives lists all the natives in each native group and their combat values, including those of their horses.

2.3.4. Weapons: Each weapon counter is identified by the silhouette on the counter. The List of Weapons identifies each weapon and defines its weapon length and its method of attack. Certain treasure cards also function as weapons, and their attributes are listed on the card or in the List of Treasures.

a. Each weapon has a length value from 0 to 18 (these numbers are not indicated on the counters; they are listed in the List of Weapons). Larger numbers indicate longer weapons.

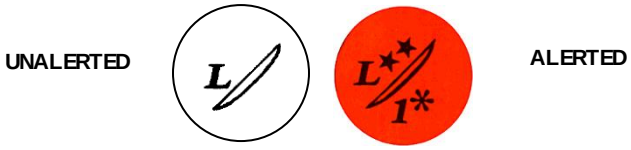
b. The letters, numbers, and symbols printed on the weapons define how they function in combat.

b.1) The number on a weapon is its attack time, which defines how much time it takes the weapon to complete an attack (lower numbers mean faster attacks).

b.2) The letter on the weapon defines the weight of the weapon and also defines the harm it inflicts when it hits, measured in the same levels as weight ("L" for Light, and so on). *Special:* When unalerted side up (with no letter), a Spear inflicts Negligible harm. It still has Medium weight.

b.3) Each sharpness star on the weapon adds one level of harm, but when the weapon hits armor one of the sharpness stars does not count (additional stars increase the damage normally). *Example:* A Light weapon with two sharpness stars inflicts Heavy damage normally, Medium damage against armor.

c. There are two methods of attacking: by *striking* ("striking weapons") and by *missile* ("missile weapons"). The weapon's method of attack modifies the harm it inflicts: missile weapons must roll on the Missile Table to adjust the harm level, and striking weapons gain one harm level when they are played with excessive Fight strength. The method of attack is listed on the List of Weapons.

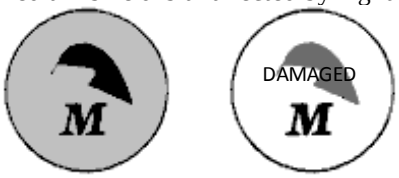


d. A weapon counter is "alerted" when the red side of the counter that shows an asterisk is face up. When the other side is face up, it is "unalerted". Each counter uses the values that are face up at the moment. Only active weapons can be alerted; when inactive or unowned, a weapon must have its unalerted side face up. Weapons can attack with either side up, but most weapons are more effective when the alerted side is face up. *Note:* Weapon cards cannot be alerted. Weapon cards stay face up whenever they are active and always have the values shown on the card.

e. Weapon counters are placed unalerted side up just before the start of each day. They remain unalerted side up unless they are alerted during the day using an Alert phase or at certain times during combat. Whichever side is face up, it must keep that side face up until the counter is turned over again in an Alert phase or by hitting or missing in combat.

2.3.5. Armor: Each armor counter represents the armor pictured on it. There are four kinds of armor: helmets, breastplates, shields, and suits of armor.

a. Each armor card and counter displays a letter that defines the armor's weight and toughness. This toughness defines the amount of harm needed to affect the armor - the armor ignores attacks that inflict less harm. *Example:* A Medium shield is unaffected by Light harm.



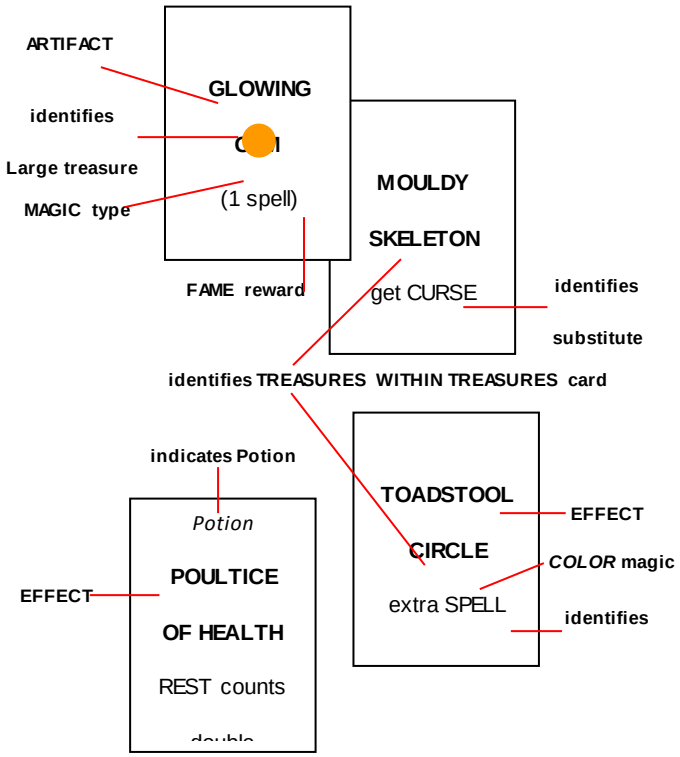
b. Each armor counter is turned "intact" side up at the start of the game. When damaged it is turned "Damaged" side up, and it remains damaged side up until it is repaired or destroyed. While damaged, it protects its owner normally.

2.3.6. Treasure Counters: The four gold weapon counters (the Bane, Truesteel, Living and Devil Swords) and the four gold armor counters labeled Gold Helmet, Silver Breastplate, Jade Shield and "T" Armor) are valuable "treasure counters"; the other weapons and armor counters are ordinary items. The pony labeled "L2" on one side

and "L4" on the other, and the war horse labeled "T3" on one side and "T5" on the other are also treated as treasure counters; the other horses are ordinary.

2.3.7. Treasure Cards: Treasure cards represent a variety of items including weapons, armor, boots, gloves, other clothing, jewelry, potions and more. Each Treasure card represents the item named on the card. Cards with a gold dot are "large Treasures" with obvious value; cards with no gold dot are "small Treasures" that are less valuable (or less obvious about their value). Cards with a red dot are "Great Treasures".

a. Each Treasure card has a letter that defines its weight and a number that defines its Gold price. The phrase in the center of the card indicates what it can be used for in the game. The List of Treasures explains how each Treasure card is used, and the Table of Treasures provides statistics for each treasure card, such as fame, notoriety, gold, weight, spell type, magic type, large and great treasures, and other information found on the treasure card.



a.1) Some Treasure cards also have a Notoriety value and a Fame value or Fame reward. A Fame *value* is identified by the word "Fame:" followed by a number. A Fame *reward* is identified by the name of a native group, a number and "F", all within parentheses.

3.3 SELECTING CHARACTERS. Each player plays the part of one character in the game. He controls that character's pieces and uses that character's character counter to represent him on the map. The List of Characters explains each character, and his Character card summarizes his qualities. This information is common knowledge - anyone can examine the Character cards and list at any time.

3.3.1. If a player has no hex tiles left when it is his turn to play, then instead of playing a tile he chooses the character he will play in the game. He can choose any character who has not yet been taken by another player. He takes his character's Character card, his character counter, his Attention chit and all twelve of his combat chits.
3.3.5. The player secretly chooses the Dwelling where he will start the game. He can start the game at any of the Dwellings listed for him on the List of Characters (none of the characters can start at the Large or Small campfires). He secretly records the Dwelling on his sheet, and when the Dwelling is put on the map his character counter is put with it.

6.2 BIRDSONG
6.2.1. Each character gets one turn per day in which he can do activities. During *Birdsong* he records his activities, and when he takes his turn during *Daylight* he does the activities he recorded. He uses the **Move** activity to move around the map, the **Search** activity to search, the **Trade** activity to trade with natives, the Hide activity to hide, the **Rest** activity to recover from fatigue and wounds, the **Alert** activity to alert weapons or magic, the **Hire** activity to hire denizens, the **Enchant** activity to enchant tiles or magic chits, and the **Follow** activity to follow other

characters or hired leaders. All of the activities are described in detail in Section 7.

6.2.2. Each character's turn is divided into "phases", and the number of phases varies from day to day. He can do one activity per phase. Before recording his turn, he determines how many phases he is entitled to.

a. He always gets at least **two** phases - these are termed his Basic phases.

b. If he spends the whole day outside of the caves, he gets **two** additional Sunlight phases. If he starts the day in a cave clearing or records a move into a cave clearing, he does not get these extra Sunlight phases. Thus, characters can normally do **two** activities per day in caves, **four** activities per day outside of the caves. *Exception:* The Dwarf cannot use Sunlight phases due to his Short Legs. He gets only the two basic phases each day. *Note:* If playing with Seasons/Weather, the numbers and types of phases each day vary by season and weather.

c. He also gets any "extra" phases caused by his Special Advantages, Treasure cards, Spells, or horses. *Example:* The Amazon's extra Move phase means she can do four activities plus a Move (or two activities plus a Move in the caves).

c.1) He can record an extra phase caused by a Treasure card or horse only if he has that Treasure card or horse active during *Birdsong.* He must have already activated it the previous day to record an extra phase due to the item. *Clarification:* A character cannot record any extra phases for a horse while he is in a cave during sunrise, because a horse cannot be activated in a cave.

c.2) He can record an enhanced activity that is caused by a Spell only if that Spell is active during *Birdsong. Special:* He can record an activity caused by a *Permanent* spell even if the spell is inert during *Birdsong.* Once he records the activity he is committed to energizing the spell if he can.

c.3) Extra phases are cumulative. *Example:* When riding a workhorse, the Amazon gets two basic phases plus two sunlight phases plus her own extra Move plus an extra Move for the horse.

6.2.3. During *Birdsong* each character secretly chooses and records the activity he will do in each phase. He can record only one activity per phase, but he can record any activity in any phase, repeating or switching activities as he wishes. When asked, he must reveal how many phases he is entitled to, but he must keep each activity secret until he does it during his turn. All of the characters record their activities secretly and simultaneously.

6.2.4. The character records his activities on his Turn Record, in the line for the current game day. He records the activities from left to right in the order he plans to do them. He can leave phases blank and not use them. When he does his turn, he must do the activities in the order he records them.

a. He can record any activity for each basic or sunlight phase. Each extra phase that specifies an activity must be used for that activity - it cannot be used for other activities. *Example:* The Magician's extra Alert phase can be used only to do the Alert activity.

b. He can record an extra phase that is restricted to specific clearings even if he is not in a clearing where he can use it. He must be in the proper clearing when he starts the activity, not when he records it. *Example:* A character can record an extra phase for the Shielded Lantern even if he is not in a cave clearing. He must be in a cave when he starts the extra phase, however.

c. Each extra Move phase caused by a pony must be preceded by a Move phase that was not caused by the pony. Move phases caused by the pony must alternate with other Move phases, and the pony cannot cause the first Move in the turn. The Move phases need not be consecutive - other activities can intervene. *Example:* The Amazon could record her own extra Move phase, then some other activity, then a Move phase caused by a pony. She could not record another Move for the pony until she recorded some other Move phase.

d. He must note each extra phase and enhanced activity he records, and the Special Advantage, Treasure card, Spell, or horse that caused it. *Important:* If a character uses a Treasure card or horse to record an extra activity or an enhanced activity, he is committed to keeping that belonging and having it active when he does that activity. The item must be active when the phase is executed. He cannot voluntarily abandon or transfer the belonging until he has finished the phase it caused. He can inactivate the belonging normally, but he must activate it when he starts to do the extra phase. *Example:* If he records a move into and out of a cave followed by an extra move caused by a workhorse, he must inactivate the horse when he enters the cave and activate it again when he starts the extra phase.

e. A character can record an impossible activity in hopes that it will be possible when the time comes to do it. *Example:* He can record a Move through an undiscovered hidden path, in hopes that he will discover the path before he has to move.

f. If he is unable to do an activity, it is cancelled and the phase is treated as a blank phase. The rest of his turn is not affected. An activity is cancelled if it violates the rules governing that activity.

g. When he does a blank phase, he does no activity. The blank phase can still be used for trading, rearranging items, and blocking.

7.1 DETERMINING TURN ORDER

7.1.1. During *Daylight* the attention chits that have been mixed together are picked one at a time, randomly, and when a character, hired leader, or controlled monster's chit is picked he takes his turn. He does his phases in the order he recorded them, and when he does each phase he does the activity he recorded for that phase.

7.3 THE MOVE ACTIVITY

The game is played in the clearings on the map. The characters move from clearing to clearing by following the roadways, either walking or riding horses. Characters can also use magic to fly or walk the woods to travel between clearings.

7.3.1. Normal Movement: Each character, hired leader, or controlled monster uses the Move activity to move from clearing to clearing. He can record several Move phases to move several clearings in the same turn.

a. He must follow the roadways when he moves, and he must end every move in a clearing. He cannot stop on the road between clearings, and he cannot move off the roads into the forest. Each time he moves he goes onto a roadway out of his clearing and moves along this roadway until he enters another clearing, which completes his move. The roadway can be an open road, tunnel, bridge, secret passage, or hidden path, but it must run from the first clearing to the second without passing through any other clearings.

a.1) He can move along open roads, bridges and tunnels freely.

a.2) He cannot move along a hidden path or secret passage until he has discovered it using the Search activity or learned its location from other players. Once he has discovered a hidden path or secret passage, he can move along it as if it were an open road for the rest of the game.

a.3) If he comes to a place where his roadway runs over or under another roadway, he must stay on the roadway he is using. He cannot change roadways where a tunnel runs under an open road, and he cannot change roadways where an open road runs under a bridge.

a.4) Leaving the Map: A character can move along a roadway to a map edge in the same way he moves to a clearing (he records "Edge" as the clearing he is moving to). When he does the activity he leaves the game (see Rule 9.2.2b).

b. When an individual does the Move activity he must move to the clearing he recorded, if it is connected to his clearing by a roadway he can use. If he cannot move to the clearing he recorded, he forfeits this phase.

c. Normally, it requires one Move phase to move one clearing, with the exceptions listed below.

c.1) Climbing Mountains: To enter a mountain clearing, he must do two Move activities on consecutive phases of the same day. He does not move at all until he does the second Move, whereupon he moves all the way to the mountain clearing. He must do two Move activities each time he enters a mountain clearing, even when he is moving from another mountain clearing. He needs only one Move phase to move from a mountain clearing to a woods or cave clearing. *Note:* If playing with the Seasons/Weather optional rule, the number of Move phases required to move into a mountain clearing may vary by season.

c.2) Entering Caves: He cannot enter (or be in) a cave clearing on the same day he uses a "sunlight" phase. *Example:* The Black Knight has already used two basic phases to move to two woods clearings. He cannot move into a cave on his third phase because the third phase would be a sunlight phase.

d. A character can do an extra Move phase that is limited to certain clearings only if he ***starts*** the move in one of those clearings. *Example:* The Captain can use the extra phase he gets at Dwellings to leave a Dwelling, but he cannot use it to move to a Dwelling.

7.4 THE HIDE ACTIVITY

7.4.1. A character, hired leader, or controlled monster uses the Hide activity to try to hide. To record a Hide activity, he records "H". When he does the Hide activity he rolls the dice and consults the Hide table to find his result. *Important:* Once he hides successfully he remains hidden. He does not have to roll for any additional Hide phases he has recorded - he automatically remains hidden.

7.4.2. A character's counter is placed tan side up when unhidden and green side up when hidden. A hired or controlled denizen's counter is placed light side up when unhidden and dark side up when hidden.

7.4.3. A character, hired leader, or controlled monster who hides successfully remains hidden until he starts his next turn, unless he is revealed by being blocked during Daylight, being targeted in Combat, or voluntarily becoming unhidden. He can choose to stop hiding at any time.

7.4.4. Characters, hired leaders, and controlled monsters can continue to do activities while they are hidden. They remain hidden as they move, search, trade, hire, etc.

7.4.5. Underlings become unhidden at the beginning of the day and can hide only by following a guide who hides successfully.

7.5 THE SEARCH ACTIVITY

7.5.1. A character, hired leader, or controlled monster uses the Search activity to search. To record a Search activity, he records "S". When he does the activity he states where he is searching and specifies the Search table he will use. Then he uses the table he specified to find his result.

7.5.2. The places where the individual can search depend on the clearing that he is in, and what is in that clearing. The table that he can choose depends on where he is searching and what he is searching for. There are eight Search tables: Peer, Locate, Loot, Reading Runes, Magic Sight, and three Treasure within Treasure Site card tables. *Note:* Hired leaders and controlled monsters cannot use the Reading Runes and Magic Sight tables.

7.5.3. The individual does not have to specify where he is searching and the table he will use until he does the activity, just before he rolls the dice. He can roll the dice only once, on one Search table, per Search activity.

7.5.4. Peer and Locate: The hidden paths, secret passages and treasure sites are considered to be concealed in the clearings where they are located. Individuals must "discover" these roadways and sites before they can use them.

a. He can discover a hidden path, secret passage or treasure site by searching in the clearing where it is located, using the Peer and Locate search tables. Each treasure site can be discovered in the clearing indicated by its Site chit. Each hidden path and secret passage can be discovered in either clearing it runs into; when an individual discovers one end of the roadway he discovers the whole roadway. He can discover only those roadways that are on the side of the tile that is face up.

b. The only clearing he can search is the clearing he is in. *Important Exception:* If he is in a mountain clearing, he can use the Peer table to search any woods or mountain clearing in his tile or any adjacent tile (he cannot search from a mountain into a cave). He must use the Peer table, he can search only one clearing and he must specify the clearing before rolling the dice.

c. A character can record and do the Peer activity only when he is enabled to by a Special Advantage, Spell, or Treasure card. The Peer activity allows characters to Peer into clearings in other areas of the board. To record a Peer activity, he records "P" and the clearing he is searching. He identifies the clearing by its number and the tile it is on. When he does the activity he rolls the dice and consults the Peer table to find his result.

d. If an individual gets a secret look at map chits through a Clues result, no one else sees them. Finding a Site chit in this way does not "discover" the site; the number on the chit just identifies the clearing that contains the site. If he finds a substitute chit, he can either exchange it or put it back without revealing it; if he exchanges the Lost City or Lost Castle for its five chits, he gets a secret look at the five chits.

e. When he discovers a roadway or treasure site, he is the only one who discovers it; it remains concealed from others, who must discover it on their own if they wish to use it. He does not have to admit whether he actually discovers a treasure site. He must reveal what he rolled, but he does not have to reveal whether there is a treasure site chit in his clearing.

f. Once an individual discovers a hidden path, secret passage or treasure site, he never has to discover it again. He keeps a record of each discovery by crossing it off the Discoveries list on his Personal History sheet.

f.1) Once he has discovered a hidden path or secret passage, he can use it for movement for the rest of the game.

f.2) Once he has discovered a treasure site, he can search it for treasure whenever he is in its clearing.

g. The Peer or Locate tables cannot be used to discover Treasure within Treasure Site cards (the only way to discover a Treasure within Treasure Site card is by drawing the card while looting).

h. Finding hidden enemies allows the individual to spy on hidden individuals when they are in the same clearing with him, and to block them. This ability remains in effect for the rest of the day, in each clearing he moves to. When an individual finds hidden enemies, he records this result in the Find Enemies column of his Turn Record.

h.1) Finding hidden enemies is not retroactive. He starts spying on hidden individuals at the moment he rolls the result.

h.2) The "Hidden enemies" result expires at *Midnight* of each game day. At the start of each day, no one can spy on hidden individuals.

i. Hired leaders and controlled monsters have their own Personal History sheet, and record their discoveries just like characters, by crossing them off the Discoveries section of their sheets.

i.1) The hiring character looks at any map chits the hired leader or controlled monster finds, but any other discoveries the hired leader or controlled monster makes are crossed off his own list, not the character's. The character cannot use these discoveries until they have been transferred to him, as described in 7.7.2a1.

i.2) When a hired leader or controlled monster finds "Hidden enemies", check off the "Find Enemies" column on the his own sheet, not the character's. This result cannot be transferred.

7.5.5. Looting Treasures: The treasures are hidden within the Treasure Site chits. Each Site chit has a box in the Treasure Location section of the Setup Card that contains the Treasure cards that are hidden at that site. To obtain these treasures, a character, hired leader, or controlled monster must search the site. Each time he searches successfully, he takes one treasure from the site's box. Searching for treasures is termed "looting". Individuals can also loot abandoned piles of belongings in a clearing.

a. To obtain treasures, an individual must: 1) identify a Site chit so he knows where to look for a site; 2) discover the site using the Locate Table; and 3) loot the site to take treasures from its box, using the Loot Table.

b. If he is in the same clearing with a Treasure Site he has discovered or a pile of abandoned belongings, he can loot it (search it for treasures). He must declare exactly which site or pile he is looting before he rolls the dice.

b.1) He can loot a pile of abandoned belongings any time he is in its clearing; he can loot a treasure site only if he has crossed it off his Discoveries list.

b.2) When he loots a face-down Site chit, or a Treasure within Treasure Site card that is in a box that belongs to a face-down Site chit, he must turn the Site chit face up and show it to all the other players (if the chit has not yet been put in its clearing he can look through the chits in the tile to find it). Then he turns the Site chit face down and puts it in the clearing where the site is located.

c. When an individual loots a Site chit or a pile of belongings, he uses the Loot table. His result indicates which treasure he gets, counting from the top of the pile. If his result is larger than the number of items in the pile, he gets nothing. *Example:* If he rolls a "1" and a "3", he gets the third item from the top. If there are only two items in the pile, he gets nothing.

d. Certain conditions must be met to loot certain sites. If an individual cannot meet the conditions, he cannot loot the site.

d.1) A character must fatigue chits to loot the Cairns and Pool. This reflects the effort needed to search these sites. He can use any chits showing an asterisk, including Magic chits. He cannot use any horses or items. Hired leaders, controlled monsters, and characters transmorphized into creatures or monsters can loot these sites without paying a penalty. Looting a Treasure Within Treasure site at the Pool or Cairns does not require fatiguing asterisks; instead, the rules and special search tables pertaining to those sites are used.

*Tearing apart the **Cairns**:* He must fatigue one asterisk each time he rolls to loot the Cairns box, whether he gets an item or not.

*Wading and diving in the **Pool**:* He must fatigue one asterisk each time he takes an item from the Pool box, whether he keeps it or not. He does not have to pay if he failed to get an item. Finding a Treasure Within Treasure site at the Pool does not count as getting an item.

d.2) To loot the Vault or Crypt of the Knight, a character must either have the Lost Keys card active (to open doors) or he must play a piece with Tremendous strength (to break down doors). He can use a Fight chit, a

Move chit, a Duck chit, a Berserk chit, a Horse, a Gloves card, or a Boots card, as long as it has Tremendous strength. He can also use the T strength of any of his followers. If he plays an action chit, it is fatigued. He can use a T strength attack created by the Wish for Strength; he cannot use any other spell with T strength to open these sites. A hired native, controlled monster, or transmorphized character can loot these sites if they have T move or fight strength.

*Prying into the **Crypt**:* He must use the Lost Keys or Tremendous strength each time he rolls to loot the Crypt of the Knight, whether he gets an item or not.

*Opening the **Vault**:* The Vault cannot be looted until someone uses the Lost Keys or Tremendous strength to open it. This penalty is paid only once per game - once the Vault is open anyone can loot it. It cannot be closed again.

e. When an individual draws an Enchanted card, a Site card, the Mouldy Skeleton or the Remains of Thief, he must reveal it instantly (see the List of Treasures). When he draws any other card he keeps it secret until he activates it.

f. As soon as a character loots an item, he can decide to activate it, keep it inactive, or abandon it in the clearing. Native leaders and controlled monsters can choose whether to keep or abandon an item, but cannot activate or inactivate items. If the item is an Enchanted Card, it activates automatically, and it energizes permanent spells after he decides whether to keep it or abandon it.

g. Three of the cards hidden at the treasure sites are Treasure within Treasure Site cards that identify minor treasure sites (see the List of Treasures). Each minor site contains the treasures that are in its box in the Treasures Within Treasures section of the Setup Card.

g.1) When an individual draws a Treasure Within Treasure Site card he crosses it off his Discoveries list, and thereafter he can loot it like any other treasure site. The only way to discover one of these sites is by drawing its Site card, or by learning its location from another individual.

g.2) After being drawn, the Site card is turned face up and is put back in the box it came from, at the bottom of the other treasures in that box. This gives others a chance to draw the card and discover the site.

g.3) When an individual loots a Treasure Within Treasure Site card, he must use its special table instead of the Loot table (see the List of Treasures).

7.8 THE REST ACTIVITY

7.8.1. Each character can use the Rest activity to return his inactive action chits to play. A guide can record this activity for the benefit of his followers. He records "R" to record each Rest activity.

7.8.2. When he does the activity, he can either activate one fatigued asterisk back into play, convert one wounded asterisk into a fatigued asterisk, or convert one wounded chit with no asterisks to an in-play chit.

a. He can rest using any *one* of the following options: he can place a fatigued one-asterisk chit or a wounded chit that has no asterisks back into play; he can put into play a fatigued two-asterisk chit and "make change" by fatiguing an active one-asterisk chit; he can convert a wounded one-asterisk chit into a fatigued chit; or he can convert a wounded two-asterisk chit into a fatigued chit and "make change" by fatiguing an active one-asterisk chit.

b. When a character "makes change", he must fatigue an active chit; he *cannot* make change by converting a fatigued or active chit into a wound. The chit can be of any type as long as it shows the right number of asterisks. *Example:* He could activate or convert a Move chit and use a Magic chit to make change.

7.8.3. When a character converts a wounded chit into a fatigued chit, he turns it face up but leaves it out of play.

7.8.4. When all of a character's chits are fatigued, wounded, and/or committed to spells, he can do only the Rest activity. Any other recorded activities are cancelled and he must rest at the next opportunity. If he cannot do the Rest activity (due to the Ill Health Curse or because all his chits are still committed to spells), he is killed.

7.9 THE ALERT ACTIVITY

A character can use the Alert activity to alert (or unalert) his weapon during his turn, or to prepare magic spells for use during combat. A hired leader can record this activity for the benefit of his followers. To record the Alert activity, he records "A".

7.9.1. When a character uses the Alert activity with a weapon, he can turn his active weapon either side up, as he

wishes. All alerted weapons turn unalerted (light) side up at midnight.

7.9.2. When a character does the Alert activity, he can use it to alert a Magic chit instead of alerting a weapon. When a Magic chit is alerted, it has an effective time number of zero instead of the time number printed on the chit. All alerted Magic chits fatigue at midnight, if not used during combat. *Note:* Characters cannot start the game with magic chits alerted.

7.13 END OF TURN – PROWLING DENIZENS

7.13.1. Prowling: On each game day, some denizens are "prowling" and the rest are "dormant". Prowling denizens can move from the Appearance Chart to the map, and prowling monsters that are already on the map can move from clearing to clearing. Denizens cannot appear or move when they are dormant.

a. The Ghosts are always prowling and can move every day. Garrison natives are always dormant and never move away from their dwelling unless they are hired.

b. On each game day, one row of the Appearance Chart will be prowling. This chart is divided into six rows, numbered 1 to 6; the small "Die Roll" boxes along the left edge of the chart define the rows (the arrow in each box points to the row, and the number identifies it). At *Sunrise* of each game day one die is rolled and the Monster Roll chit is put in the box that matches the number rolled. All of the denizens pictured or listed in this row are prowling that day, whether they are on the map or are still on the Appearance Chart (the visitor/mission chits are prowling when the Monster Roll is "6"). The denizens in the other five rows are dormant. *Example:* When the Monster Roll is "4", only the Giants, Trolls, Lancers, and Ghosts (who prowl every day) can appear on the board or move.

8.4.5 Playing Attacks and Maneuvers: After all of the spells have started working or have been cancelled, the characters secretly play their attacks and maneuvers to resolve weapon attacks and attack spells by comparing attack direction and speed with maneuver direction and speed.

a. A character cannot attack with a weapon or Fight chit in the same round that he successfully casts a spell. He can play a maneuver normally. If a character's spell was cancelled before it started working, he can play a normal attack. The only target he can attack is the target where his Attention chit is located. If this target is not a character or denizen, he cannot attack.

b. An attack represents a single blow that is coming at the target from a specific direction. Each attack has a direction and a time number (or attack time). There are three attack directions: Thrust (straight ahead), Swing (from side to side) and Smash (downwards). The attack time defines the amount of time before the blow hits; lower numbers represent faster attacks.

c. A maneuver represents a move in one direction to evade attacks that are coming from the other directions. Each maneuver has a direction and a maneuver time. The three maneuver directions are Charge (straight ahead), Dodge (sideways) and Duck (downwards). The maneuver time defines the amount of time before the target completes his maneuver; lower numbers mean faster maneuvers.

d. During the Melee Step each character can make one attack against one other character or denizen, and each denizen makes one attack against one character or another denizen. *Exception:* Attack Spells can attack multiple characters or denizens. Each character and denizen can also do a defensive maneuver to avoid enemy attacks.

e. Character Attacks and Maneuvers: Each character can play an attack, a maneuver and his armor, if he has armor active. He can choose to not play an attack or maneuver if he wishes, but if he has a shield or other armor active he must play it. All of the characters make their plays secretly, concealing their Melee Sections from each other. When they have finished they reveal their plays simultaneously. The plays cannot be changed once they are revealed.

e.1) To attack with a weapon, a character puts his active weapon and a Fight chit in one of the Attack circles on his melee sheet. His attack is always directed against the target he placed his attention chit on at the beginning of the Melee Step. The circle he chooses defines his attack direction: each circle names the direction it represents and the maneuver it intercepts. His attack time equals the time number printed on his weapon; if his weapon has no time number, his attack time equals the time number on the Fight chit.

e.2) To play a maneuver, a character puts a Move chit in one of his Maneuver squares. The square he chooses defines his maneuver direction: each square names the direction it represents. His maneuver time equals the time number on the Move chit.

f. Playing Restrictions: There are limits on the action chits that a character can play during combat. However,

the monsters or natives on his sheet do not restrict what chits he can play during the melee step.

f.1) One Use: Each action chit can be played only once per round. At the end of each Encounter Step and Melee Step, each character puts the action chits he played in the Used This Round box on his Melee Section. These chits stay in the Used This Round box until the end of the round, when he gets them back to use in the next round of combat.

f.2) Effort limit: Each character can play no more than a total of two effort asterisks per round of combat. If he plays a chit that causes his asterisks for the round to total more than two, then the play is cancelled. *Example:* if the White Knight plays his Move H4** chit to maneuver, he cannot play his Fight T5* chit to attack. Magic chits are also subject to the Effort limit: if the Druid plays his Magic I3* chit to cast a spell, he cannot play a Move L2** chit to maneuver.

f.3) Strength restrictions: The character's items limit the action chits he can play. He can play a Fight chit only if its strength equals or exceeds the weight of his active weapon. He can play a Move chit only if its strength equals or exceeds the weight of every item he is carrying (this does not include inactive items being carried by a Pack Horse). If he plays a chit that has insufficient strength, he must cancel the play - during the Melee Step he does not have the option of abandoning the over-weight item(s). *Example*: The Black Knight cannot use his Move M4 chit to maneuver as long as he has his Heavy armor (even if inactive). To use his Medium Move chits to maneuver in the Melee Step, he would have to abandon his armor and any other Heavy items he was carrying during the Encounter Step.

g. Illegal plays are cancelled and have no effect, and the chits that were played are put in their owner's Used This Round box. They cannot be played again that round and their effort asterisks do count towards their owner's effort limit and fatigue.

h. Weapons: Weapons can be used only if they are active. *Important:* Each character is limited to one active weapon (counter or card). He can own any number of weapons, but only one at a time can be active. Monsters and other denizens cannot use the weapon counters and cards. He can use his active weapon whether it is alerted or unalerted (alerted weapons just have better combat values). When a character has a weapon active when he plays a Fight chit to attack, he must play the weapon in the attack. Combat characteristics of alerted and unalerted weapons are shown in the List of Weapons.

h.1) If his weapon has a time number, this number must define his attack time - he does not have the option to use the chit's time number instead. If his weapon has no time number, then the time number on his Fight chit defines his attack time.

h.2) Each time a weapon hits in combat, it must be turned unalerted side up. Each time it attacks and misses, it must be turned alerted side up. In place of an attack, a character can play a Fight chit during the Melee Step without specifying a target; if he does so, he can turn his weapon either side up at the end of the round. This is normally done to alert a weapon for a later round of combat.

h.3) If he does not have an active weapon, he can attack with a Fight chit alone. *Explanation:* He is assumed to own a dagger that he uses when no better weapon is available. The dagger does Negligible harm with one sharpness star and no time number. He cannot use this dagger when he has a weapon active - he must play the weapon.

i. Armor: During combat, each character must put all of his active armor pieces in the ovals on his Melee Section. He cannot put inactive armor on his sheet. Each active armor piece protects him from the attack directions named in the oval where it is put.

i.1) He must put his helmet, breastplate and suit of armor counters in the ovals where they are pictured.

i.2) If he has an active shield, he must put it in one of the Shield ovals. It protects only the direction named in the oval. He secretly puts the shield in the oval during the melee step, at the same time that he plays his attack and maneuver.

i.3) The Ointment of Steel armor card protects all three directions, like a suit of armor. It is put in the Suit of Armor oval. If a Suit of Armor is also present in that oval, any attack which intercepts the oval hits the Ointment of Steel before it hits the Armor. The Suit of Armor cannot be damaged or destroyed until the Ointment of Steel is destroyed.

i.4) The other three armor cards are put in the Shield ovals: the Bejewelled Dwarf Vest must be put in the oval labeled "Thrust," the Golden Arm Band must be put in the oval labeled "Swing," and the Golden Crown must be put in the oval labeled "Smash". If a Shield is also present in an oval with an Armor card, any attack which intercepts the oval hits the Shield before it hits the Armor card. The Armor card cannot be destroyed until the

Shield is destroyed.

8.4.8. Resolving Attacks: After the denizens are randomized and change tactics, compare each attack with its target's maneuver to see if the attack misses. Attacks that hit do not go into effect yet - they can still be stopped by killing the attacker before his attack hits. Each attack automatically hits unless its target maneuvers to avoid it or the attacker is killed before the attack hits.

a. Compare the attack time of each attack to its target's maneuver time. If the attack time is lower, the attack will hit the target by "undercutting" the target's maneuver time.

b. If the attack did not undercut, compare the direction of the attack to the direction of its target's maneuver. If the attack intercepts the maneuver, the attack will hit. The arrows on the Melee Section illustrate which attack directions intercept which maneuver directions.

b.1) The grey arrows show which red box is hit by each Attack circle. Follow the grey arrow from the circle containing his attack until it comes to a red box. If the target is in this box or in a matching box on another sheet, the attack intercepts and hits. *Note:* The attack cannot hit other denizens in the box. The only denizen that can be hit is the designated target. *Example:* If a character has played his Fight chit in the Smash attack circle, he intercepts his target if the denizen with his Attention chit on it is in any "Duck and Smash" box on any sheet.

b.2) The red arrows show which Maneuver square is hit by each red box. If a red box contains denizens, follow the red arrow from that box until it comes to a Maneuver square (after passing through the armor ovals). If this square contains the target's maneuver, all of the denizens in that red box intercept the maneuver and hit. *Special:* If a character does not play a maneuver, all of the denizen counters on his sheet intercept and hit him.

c. If an attack neither undercuts nor intercepts its target, it misses. *Explanation:* The target completes his maneuver and gets out of the way of the attack. When a character misses, he removes his attack from its circle and turns his active weapon alerted side up. When a denizen misses, it is put at the top of the target's Melee Section to indicate that it is still engaged with its target even though it has missed this round.

d. When one character attacks another, his Attention chit attacks with the combat values defined by the weapon and Fight chit he played. His chit attacks from the red box on the target's sheet that matches the direction of the attack. *Example:* If he plays a Swing attack, he attacks from the "Dodge and Swing" box on his target's sheet. If the targeted character has played his Move counter, Boots card, or horse in the "Dodge" maneuver box, he is intercepted.

e. If a character attacks a target on a denizen's Melee Sheet, his attack circle intercepts the matching circle on that Melee Sheet, the red box that this circle points to, and the Maneuver square that this red box points to. *Example:* If he plays a Smash attack, he intercepts his target if it is in the "Smash" circle, the "Duck and Smash" box or the "Duck" square.

f. On a denizen's sheet, each attacker in an Attack circle intercepts the red box his circle points to, the owner intercepts the Maneuver square that his red box points to, and the attacker in the Maneuver square intercepts the red box that points to his square.

g. Whether or not an Attack spell hits is determined in exactly the same manner as a weapon. It must either undercut (based on the spell's completion time) or intercept the target(s) to hit.

8.4.9 Inflicting Harm: Attacks that hit go into effect one at a time, in the order defined by their weapon length and attack times.

a. On the first round of combat in each clearing each day, attacks hit in the order defined by their weapon length, longer weapons before shorter. If attacks have the same weapon length, the attack with the faster attack time hits first.

b. In all subsequent rounds of combat in that clearing that day, attacks hit in the order defined by their attack times, faster attack times before slower. If attacks have the same attack time, the attack with the greater weapon length hits first. *Note:* If two or more attacks have both the same time and the same weapon length, they are simultaneous and inflict damage at the same time (see Rule 8.4.9m).

c. Hits inflict harm one by one, in the order defined by their length and attack time. When a hit inflicts harm the results go into effect instantly, before the next hit inflicts harm: if the hit changes the situation then the next hit inflicts harm in accordance with the new situation. *Exception:* Wounded action chits are not removed from play until the fatigue step.

c.1) When a denizen or character is killed, all attacks that are waiting to hit him are cancelled. Any weapons making the cancelled attacks turn alerted side up.

c.2) If a character or denizen is killed before his attack inflicts harm, his attack is cancelled and has no effect. *Example:* If a character and monster play killing attacks against each other, the one who hits first survives.

d. When a character's attack hits (if it is not cancelled before it can take effect), his weapon becomes unalerted.

e. When an attack hits, it inflicts "harm" on its target. Harm is measured in the same levels as weight: "T" for Tremendous, "H" for Heavy, "M" for Medium and "L" for Light. The harm that a weapon or monster counter inflicts is equal to the harm letter on the attacking piece, plus levels for the sharpness stars on the attacking piece.

e.1) If the attack does not hit armor, each sharpness star increases the harm one level.

e.2) If the attack does hit armor, one of the sharpness stars does not add to the harm (the star is not lost permanently, it just does not count in the current attack), but each additional star increases the harm one level. *Example:* A counter with an "M" (Medium) harm letter and two sharpness stars inflicts Tremendous harm normally, but only Heavy harm when it hits armor.

e.3) If the attacking piece has no sharpness stars, the harm it inflicts is not affected by armor. A Heavy counter with no sharpness stars inflicts Heavy harm whether it hits armor or not.

e.4) When an attack hits a monster, the attack hits armor only if the monster is a Dragon, Troll or Serpent (including Vipers).

e.5) Armored natives: Each Knight, Great Swordsman, Pikeman, Short Swordsman and Crossbowman is protected by armor in all directions, as are all Warhorses (no other natives have armor). Every attack that hits one of these natives hits armor and is reduced by one sharpness star.

f. The method of attack of the character's weapon can modify the harm he inflicts.

f.1) If he hits with a missile weapon, he must roll on the Missile Table and adjust the harm level as indicated by his result.

f.2) If he plays a striking weapon with a Fight chit whose strength exceeds the weight of the weapon, the harm increases one level. The harm increases only one level no matter how much extra strength the Fight chit has. *Example:* If he plays a Medium, Heavy or Tremendous Fight chit with a Light striking weapon, the harm increases one level (to Medium, plus any sharpness).

f.3) *Special:* A "dagger" (a Fight chit played alone) is a "Negligible" striking weapon with one sharpness star. It always gains a level for excess Fight strength, so it inflicts Medium harm when it misses armor, Light harm when it hits armor.

g. Bowmen: The Archers and Crossbowmen fire missile weapons, so the harm they inflict is modified by the Missile Table. When an Archer or Crossbowman hits, one of the players rolls for him on the Missile Table and the result modifies the harm he inflicts.

g.1) The player's die roll modifiers do not affect this roll - he is rolling for the native, not for himself. Any player can roll.

g.2) The Archers have their own die roll modifier: when a player rolls on the Missile Table for an Archer's attack, he rolls only one die. This does not apply to the crossbowmen, who roll two dice. *Note:* When light side up (the side with no attack values), an Archer counter cannot attack.

h. When the target is a denizen, compare the final harm inflicted by the attack to the denizen's vulnerability. Monsters' vulnerabilities are defined by counter size. Natives' vulnerabilities are defined in the List of Natives, and horses' vulnerabilities (belonging to both characters and natives) are shown in the List of Horses. If the harm equals or exceeds its vulnerability, the denizen is killed. If the harm is less than the vulnerability, the harm has no effect. After the hit takes effect, the attacking weapon counter is turned un-alerted side up.

8.5 FATIGUE AND WOUNDS

At the end of each round of combat, during the Fatigue Step, each character must inactivate action chits to pay for the fatigue and wounds he incurred during the round.

8.5.1. Fatigue: Each character can normally play Fight and Move chits totaling up to two asterisks per round (some Treasures allow more asterisks to be played). If he plays chits with no asterisks or a total of one asterisk, no fatiguing is required. If he played chits with a total of two or more asterisks, he must fatigue by moving chits with asterisks from active status to inactive (fatigued) status. The number of asterisks he must fatigue is equal to the number of asterisks he played, minus one.

a. He can fatigue any chits with asterisks he has in play, including chits he played this round, as long as they

are of the same type as the extra asterisks he played. If he played only Move chits, he must fatigue a Move chit, and if he played only Fight chits, he must fatigue a Fight chit. If he played both Move and Fight chits, he must specify which asterisk does not fatigue, and fatigue asterisks of the same type(s) as the remaining asterisks played. If he fatigues a double-asterisk chit he can bring back a single-asterisk chit of the same type to "make change" (see Rule 4.2.5).

b. When a character plays no asterisks he does not gain asterisks - he just does not have to fatigue any.

c. Normally a character will play no more than two asterisks, but if he uses the Potion of Energy or Girtle of Energy Treasure cards to play more than two asterisks, then the extra asterisks also fatigue. The number of asterisks he fatigues is always one less than the number he played.

8.5.3. Wounds: Each character must wound one active chit for each wound that was inflicted on him during this round. He gets one wound for each attack that inflicts Light or heavier harm on him, or Medium or heavier harm on his armor. He is not wounded by attacks that hit his horse. He can wound any action chits he has in play, including chits he played this round (see Rule 4.2.6). A transmorphized character cannot be wounded.

8.7 SPOILS OF COMBAT

8.7.1. When a character kills a monster, he immediately gains recorded Fame and Notoriety points. He gets the bounty points for the monster, multiplied by the number of monsters he has killed that day (including the monster he just killed). *Example:* He collects the basic bounty points for the first monster he kills each day, double points for the second monster he kills, triple points for the third monster, and so on. *Note:* Head and club counters do not count as separate kills.

a. As he kills monsters he lines them up to keep track of how many he has killed that day. He starts a new line each day.

b. If two or more characters kill the same monster simultaneously, they divide the bounty points equally, retaining fractions, and each character counts it as a kill: he gains his share of the bounty points multiplied by his number of kills (including the shared monster). *Example:* The Elf and Druid simultaneously kill a Wolf, worth 1 Notoriety point. Each character gets half a Notoriety point, multiplied by his number of kills. It is the Elf's fourth kill of the day, so he gains 2 Notoriety and his next kill will count as his fifth. It is the Druid's first kill of the day, so he gets only half a point of Notoriety, and his next kill will be his second.

c. Characters share a kill only if their attacks are perfectly simultaneous, equal in attack time and length. If one character hits first, he gets all the points and the characters who hit later get nothing.

8.7.2. Each character or native that a character kills counts as a kill just like a monster. He adds it to his line of kills for the day, and it multiplies the Fame and Notoriety he gets when he kills a denizen.

a. When a character kills a native, he gains its Notoriety bounty, multiplied by the number of kills he made that day (including the native he just killed), and he gains the native's Gold bounty, without multiplication. Natives do not have a Fame bounty.

b. When he kills another character, he takes the victim's recorded Notoriety, without multiplication. (If his victim has negative Notoriety, he loses Notoriety.) He does not get his victim's Fame points - when a character is killed, his Fame just vanishes. The victim does count as a kill.

c. When a spell kills a character or denizen, the spell caster counts it as his kill and gets the points for it. If he kills several individuals simultaneously, he calculates their points as if he had killed them one at a time, starting with the individual worth the most points (without multiplication) and ending with the one worth the least.

d. When a character kills a character or hired leader, he also takes any belongings and recorded Gold the victim has with him. When a character is killed, his recorded spells vanish. The killer does not get them.

8.7.5. Killing Denizens: When a monster or native is killed, it is removed from play until it regenerates and returns to play at the end of a week. Visitors can never be killed.

8.8 ENDING COMBAT IN A CLEARING

8.8.1 End of the Round: After the fatigue step the characters determine whether there will be another round of combat. If combat continues, the characters and denizens start the next round of combat.

9.0 ENDING THE GAME

9.1 CHARACTER DEATHS AND RESTARTS

9.1.1. Character Deaths: Characters may be killed by monsters, natives, other characters, or Power of the Pit spell results. Characters may also be killed by wounds or fatigue, or by suiciding.

a. Characters can be killed outright by taking a direct blow in combat that has an attack strength equal to or greater than their vulnerability. *Note:* Under the Serious Wounds optional rule, blows equal to their vulnerability may result in multiple wounds rather than death.

b. Characters can also be wounded to death in combat by multiple blows. As soon as all of a character's chits become wounded or committed to spells, he is killed. When determining which chits to wound, players must first wound all of their un-enchanted in-play chits, then their enchanted in-play chits, then any fatigued chits. This is the only time that fatigued chits can be wounded. Chits dedicated to spells cannot be wounded and do not count when determining whether all chits have been wounded (see Rule 4.2.6).

c. When all of a character's chits are fatigued, wounded, or dedicated to spells, the only activity he can record and carry out is a Rest phase. If the character cannot do a rest phase on the next Daylight phase (for example, due to the Ill Health curse), he is killed. *Note:* Certain types of weather in the optional Seasons/Weather rules require involuntary fatiguing of chits, and can eventually result in death. See the Seasons/Weather rules for details.

d. Characters can be killed by Power of the Pit spell results, whether cast by characters or Demons, or by looting the Toadstool Circle. If the character is killed by a Power of the Pit attack from the Toadstool Circle, fatigue, or suicide, his items are abandoned in the clearing, and his recorded fame, notoriety, gold, spells, and discoveries are lost.

3 – Requirements

This Section contains what features must be implemented. Each requirement has its own identifier and a short description. The source of traceability may either be from the official game rules, assumptions, other requirements, or team decisions.

3.1 – Functional Requirements

Functional requirements define what behavior and functionality the software must have. They have been categorized below, based on different game states or areas of functionality. All of the functional requirements specified in the CorrectionGrid file provided are represented below.

ID	Functional Requirements	Traceability
FR-1	Display correct titles of the board.	A-1
FR-2	Board is scrollable for each client.	A-1
FR-3	Networking capabilities. Capable of having multiple players. The actions of one player are represented in each window of his opponents.	A-3
FR-4	Initial character selection offers 2 or more distinct characters.	GR-1.4.1. GR-3.3.
FR-5	Cheat Mode allows user to assign 4 types of Map Chits (V, W, M, C) to specific tiles.	GR-1.4.3.
FR-6	Can give the 5 Map Chits to Lost Castle.	GR-1.4.3.
FR-7	Can give the 5 Map Chits to Lost City.	GR-1.4.3.
FR-8	Can manually place Lost Castle.	GR-1.4.3.
FR-9	Can manually place Lost City.	GR-1.4.3.

FR-10	Sound and Warning chits are correctly displayed to clients.	GR-1.4.3.
FR-11	Support choosing starting location for characters.	GR-1.4.2. GR-3.3.5.
FR-12	Support of moving within the tile.	GR-1.4.2. GR-7.3.1.
FR-13	Support of moving across tiles.	GR-1.4.2. GR-7.3.1.
FR-14	Support Captain's extra phase when he is in a dwelling.	GR-3.3. GR-6.2.
FR-15	Support Amazon's extra Move Phase.	GR-3.3. GR-6.2.
FR-16	Support the choice of dice roll results in cheat mode.	A-4
FR-17	Support hiding successfully (1-5).	GR-7.4.
FR-18	Support hiding unsuccessfully (6).	GR-7.4.
FR-19	Selection of fight and move counters for combat. Then resolve combat.	GR-8.4.5.
FR-20	Can choose shield direction.	GR-8.4.5.
FR-21	Support Multiple Rounds of combat between 2 characters. Resulting in a death or no death.	GR-1.4.4. GR-7.9. GR-8.4.5. GR-8.4.9. GR-8.8. GR-9.1.
FR-22	Support a 2 effort limit per round of combat.	A-5 GR-1.4.4.
FR-23	Support random order of player turns.	GR-7.1.
FR-24	Support Dwarf only having 2 phases a day.	GR-3.3.
FR-25	Support Black Knight	GR-3.3.
FR-26	Support the recording of player's phases and then implementing them after all players have finished recording.	GR-6.2.
FR-27	Support the viewing of objects in clearing.	A-2 GR-1.4.3.
FR-28	Support the alerting of weapons.	GR-2.3.4. GR-7.9.
FR-29	Support using different stats of an alerted weapon.	GR-8.4.9.
FR-30	Support combat with un-alerted weapons.	GR-8.4.9.
FR-31	Support fatiguing counters.	GR-8.5.1.
FR-32	Support wounding counters.	GR-8.5.3.
FR-33	Support resting fatigued counters.	GR-7.8.
FR-34	Support resting wounded counters.	GR-7.8.
FR-35	Support locating of treasure.	GR-7.5.4.
FR-36	Support looting of treasure.	GR-7.5.5.
FR-37	Support Great treasures.	GR-2.3.6. GR-2.3.7.
FR-38	Support specification of treasure's notoriety and fame.	GR-2.3.6.

		GR-2.3.7. 8.7.
FR-39	Player's current gold, fame, notoriety updated regularly.	GR-1.4.4. GR-7.5.5. 8.7.
FR-40	Support cheat mode for searching tables.	A-4
FR-41	Support locating of secret paths and passages.	GR-7.5.4.
FR-42	Support of Cloak of Mist.	GR-3.3. GR-7.5.5.
FR-43	Support of Magic Spectacles.	GR-3.3. GR-7.5.5.
FR-44	Support Cave phase restriction.	GR-1.4.2. GR-6.2. GR-7.3.1.
FR-45	Support of random dice rolls in normal mode.	A-4
FR-46	Support monsters prowling to other clearings.	GR-7.13.1
FR-47	Support missile attacks.	GR-8.4.9.
FR-48	Support viewing of clearing values.	A-1 GR-1.4.3.
FR-49	Support monsters.	GR-2.2.
FR-50	Support character death. Creating pile of his belongings.	GR-8.7. GR-9.1.
FR-51	Support Looting of his belongings	GR-7.5.5.
FR-52	Support victory point selection.	GR-1.4.1. GR-1.4.5.
FR-53	Support victory point calculation.	GR-1.4.5.

3.2 – Assumptions

While building the software many assumptions were made about the rules so that the rules could be captured the software requirements. Many of the requirements above will trace to these items. The following table will identify them along with their justification.

ID	Assumption	Justification
A-1	Each player has a labelled scrollable map.	Players will want to be able to remember quickly where everything is located.
A-2	Players want to view the contents of a clearing.	Players will want to know what treasure, monster or other character is in the clearing so they can make informed choices.
A-3	Game must have minimum of 1 player and a maximum of 6. All should be able of viewing the others actions.	Game becomes longer the more players are added.
A-4	Player wants to be able to choose Dice	By fixing the result, it guarantees

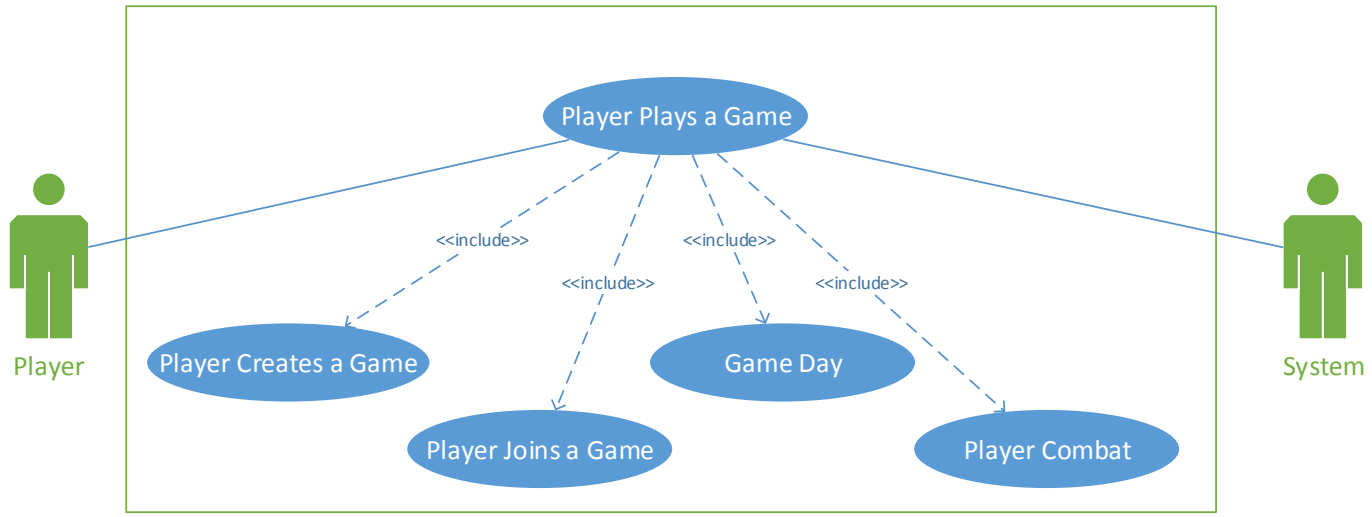
	result.	access to all portions of the game.
A-5	Put a 2* limit on combat rounds.	This keeps the game moving quickly. Also ensures new players don't accidentally kill their characters in round 1.

4 – Use Cases

A use case covers a scenario. It details the path of events needed for this scenario.

4.1 – Use Case Diagram

The following diagram corresponds to the use cases in 4.2. Actors are depicted as persons. Ellipses represent use cases, dashed arrows with an <<include>> stereotype represent an "includes" relationship, and regular lines represent association.



4.2 – Use Cases

Each use case is detailed in its own table row, which describes its sequence of events.

ID	Name	Description
UC-1	Player Plays a Game	A player starts and plays a game of Magic Realm. **Actors:** Player, System **Triggering Event:** A new game of Magic Realm is started. **Pre-Conditions(s):** The player is not in the game. **Main Sequence:** 1. Player arranges a game with 1-6 players (execute UC-3 if

		hosting or UC-4 if player is a client). 2. Until everyone has played 28 Game Days 2.1. System displays the current game state. 2.2. A day will pass (Execute UC-2). 3. Calculate victory points based on players choices when creating the game. 4. Display the total score. 5. Player then leaves the game. **Post-Condition(s):** Game has ended. Player is not in a game. **Resulting Event:** The Player leaves the game. **Alternative Scenario:** Player leaves game Early – can happen at any time by closing application or quitting. Player is not replaced by another and is not removed from game. **Traceability:** FR-53
UC-2	Game Day	During a game of Magic Realm all players progress through the day concurrently. **Actors:** Players, System. **Triggering Event:** The system enters a new day round. **Pre-Condition(s):** There exists a player. The game has not ended yet. **Main Sequence:** 1. Player records what he intends to do on his turn. 2. Monsters roam a little in their tiles. 3. Players are chosen to do their recorded turns in a random order. 4. After all players have done their turn. If there are possible player vs player combats execute UC-5. 5. If possible the player may trade. 6. All weapons become un-alerted. **Post-Condition(s):** It is no longer the current day. **Resulting Event:** The game state has been advanced. **Traceability:** FR-12 to FR-18, FR-23, FR-24, FR-26 to FR-29, FR-31 to FR-46, FR-48 to FR-51
UC-3	Player Creates a Game	A player sets up a game of Magic Realm. **Actors:** Player, System. **Triggering Event:** The player chooses to host a new game of Magic Realm. **Pre-Condition(s):** The player is not in a game. **Main Sequence:** 1. The host player starts the server, specifies the number of players and then waits for them to connect to his game. 2. He will then specify whether the game is in cheat mode or not.

		3. If in cheat mode he will then place all map chits. 4. All players will then choose their character, their starting location and their victory points. 5. Once all players have chosen the game will commence. **Post-Condition(s):** The player is in a session. **Resulting Event:** The players begin a session. **Traceability:** FR-1 to FR-11, FR-25, FR-37, FR-38, FR-42, FR-43, FR-52
UC-4	Player Joins a Game	A player joins a game of Magic Realm as a client. **Actor:** Player, System **Triggering Event:** The player chooses to join a game of Magic Realm as a client. **Pre-Condition(s):** The player is not in a game. **Main Sequence:** 1. The player finds and specifies a game to join that is available. 2. Will then do steps 3-5 of UC-3. **Post-Condition(s):** The player is in a session. **Resulting Event:** The players begin the session. **Traceability:** FR-1 to FR-11, FR-25, FR-37, FR-38, FR-42, FR-43, FR-52
UC-5	Player vs Player Combat	A player fights against another player. **Actors:** Player, Player, System **Triggering Event:** Two players end the day in the same clearing. **Pre-Condition(s):** Both players are alive and in the same clearing. Neither player has already fought today. **Main Sequence:** 1. Both players choose how they will attack their opponent, their shield defense and their defensive maneuver. 2. Player with the best attack attribute will attack first. 2.1.Determine if attack hit. 2.2.Determine attack damage. 2.3.Resolve damage. 3. If second player is still alive he will attack. Repeat step 2 using second player. 4. If both players are still alive execute UC-5. **Alternative Scenario:** Both players choose to do nothing- combat ends. **Post-Condition(s):** One player is dead or both players are alive. **Resulting Event:** Combat is finished, the day may resume. **Traceability:** FR-19 to FR-23, FR-26, FR-28 to FR-34, FR-47, FR-50

4.3 – Responsibilities

These are created from the use cases above; they will be used in the use case maps in section 4.4. The reference for the respective use case is on the right.

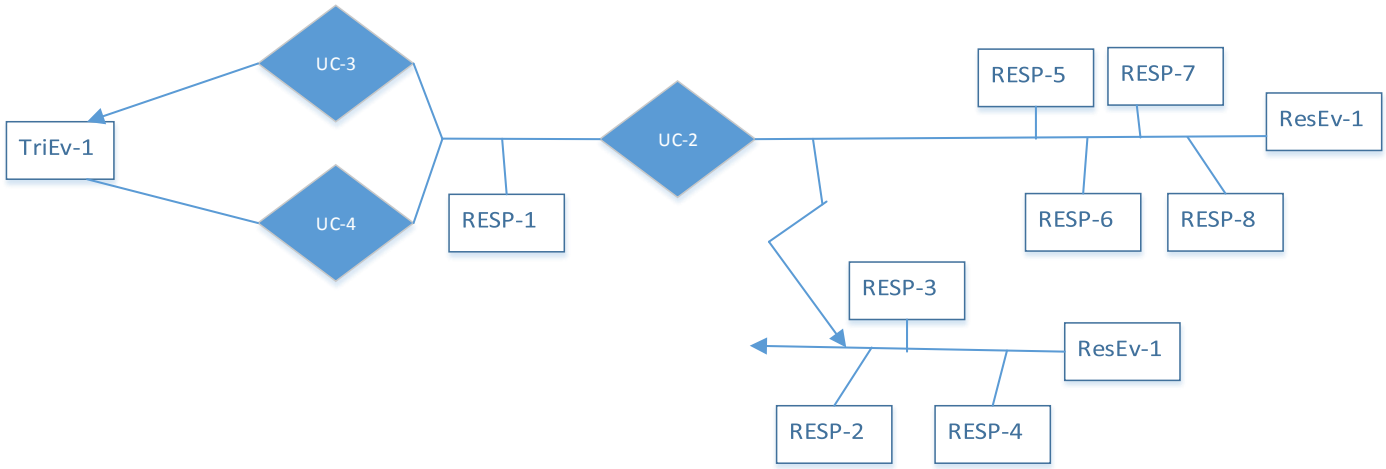
ID	Responsibility	Use Case(s)
RESP-1	The system displays the current game state.	UC-1
RESP-2	The system calculates victory points for the players.	UC-1
RESP-3	The system checks for any human players that have exited.	UC-1
RESP-4	The system removes any exiting human players.	UC-1
RESP-5	The system determines whether 28 game days have been played.	UC-1
RESP-6	The system notifies the player that the game has terminated unexpectedly.	UC-1
RESP-7	The system displays the scoreboard to each player.	UC-1
RESP-8	The player exits the game.	UC-1
RESP-9	The player records his actions for this day.	UC-2
RESP-10	The system moves the prowling monsters around their tiles.	UC-2
RESP-11	The system randomly selects a player to do their turn.	UC-2
RESP-12	The system executes the player's turn.	UC-2
RESP-13	The system checks if there is possible combat then calls UC-5.	UC-2
RESP-14	The system checks if there is possible trading.	UC-2
RESP-15	The system un-alerts all weapons.	UC-2
RESP-16	The system displays the partially updated game state.	UC-2 UC-5
RESP-17	The system determines whether every player has completed their turn during this day.	UC-2 UC-5
RESP-18	The system advances the game state.	UC-2 UC-5
RESP-19	The player will choose where treasures, sounds, ghosts, dwellings, lost castle and the lost city are placed if this is cheat mode.	UC-3 UC-4
RESP-20	The player will choose their character, his starting location and the victory points needed.	UC-3 UC-4
RESP-21	The host player waits for a guest player to join.	UC-3
RESP-22	The host checks that there are enough players.	UC-3
RESP-23	The host begins the session.	UC-3
RESP-24	The host will choose if the game is in cheat mode or not.	UC-3
RESP-25	The system verifies whether or not the game to be joined is accessible.	UC-4
RESP-26	The system connects the player to the game as a client.	UC-4
RESP-27	The guest player waits for the host player to begin the session.	UC-4
RESP-28	The system initializes the session on behalf of the host player.	UC-4
RESP-29	The players selects attack direction and strength, shield direction, evasion direction and strength.	UC-5
RESP-30	The system verifies if the attack hits the other player.	UC-5
RESP-31	The system verifies if any damage is done to the other player.	UC-5

RESP-32	The system applies damage to the other player.	UC-5
RESP-33	The system will verify that the player is still considered alive.	UC-5
RESP-34	The players decide to leave the combat.	UC-5
RESP-35	The player is removed from play.	UC-5

4.4 –Unbounded Use Case Maps

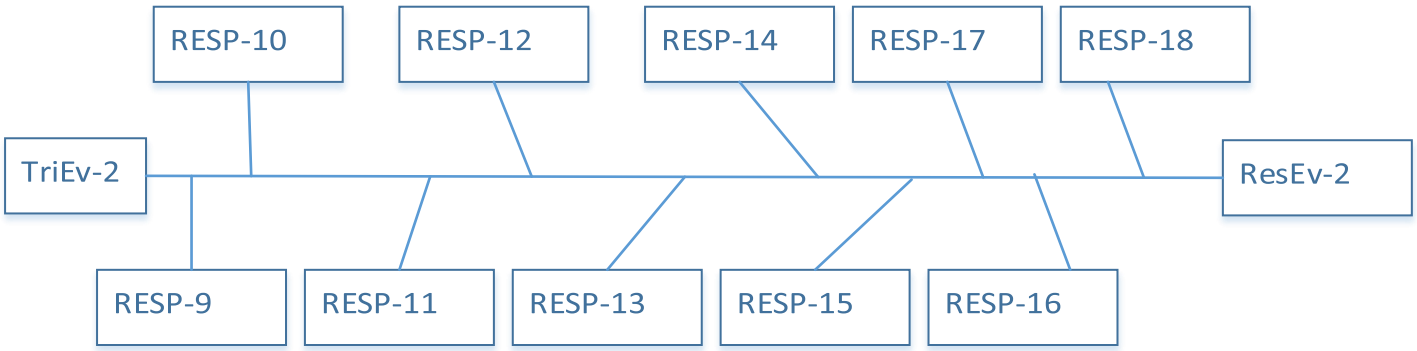
The unbounded maps correspond to section 4.2 use cases. The Triggering and resulting events are labeled in the tables following them.

UUCM-01



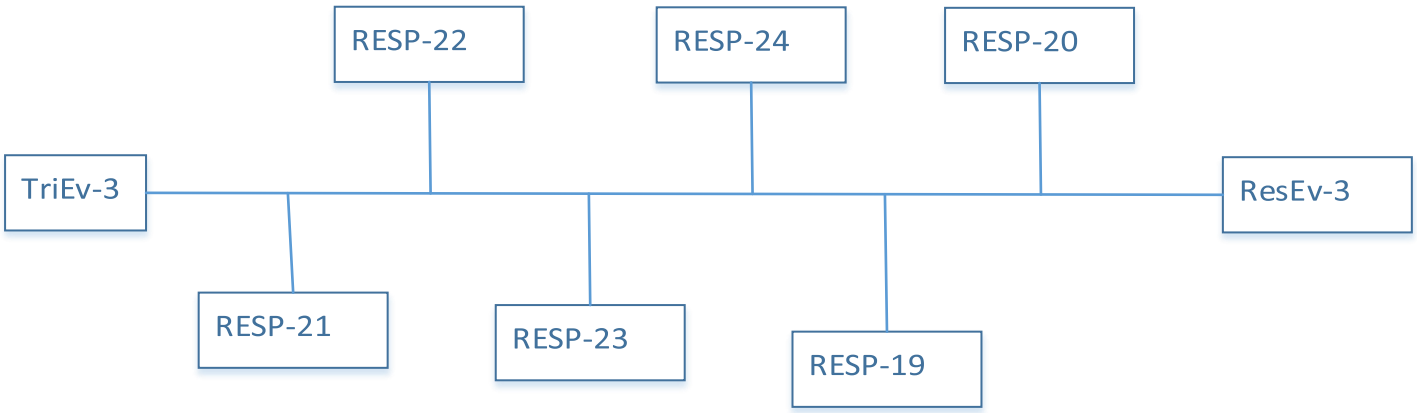
TriEv - 1	A new game of Magic Realm is started.	
ResEv - 1	The Player leaves the game.	
ID	**Responsibility**	**Use Case(s)**
RESP-1	The system displays the current game state.	UC-1
RESP-2	The system calculates victory points for the players.	UC-1
RESP-3	The system checks for any human players that have exited.	UC-1
RESP-4	The system removes any exiting human players.	UC-1
RESP-5	The system determines whether 28 game days have been played.	UC-1
RESP-6	The system notifies the player that the game has terminated unexpectedly.	UC-1
RESP-7	The system displays the scoreboard to each player.	UC-1
RESP-8	The player exits the game.	UC-1

UUCM-02



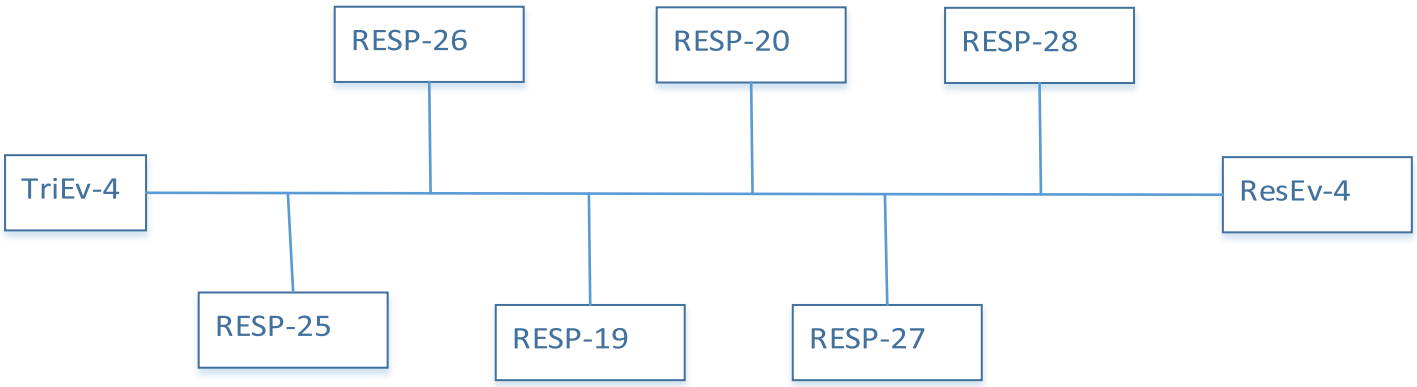
TriEv - 2	The system enters a new day round.	
ResEv – 2	The game state has been advanced.	
ID	**Responsibility**	**Use Case(s)**
RESP-9	The player records his actions for this day.	UC-2
RESP-10	The system moves the prowling monsters around their tiles.	UC-2
RESP-11	The system randomly selects a player to do their turn.	UC-2
RESP-12	The system executes the player's turn.	UC-2
RESP-13	The system checks if there is possible combat then calls UC-5.	UC-2
RESP-14	The system checks if there is possible trading.	UC-2
RESP-15	The system un-alerts all weapons.	UC-2
RESP-16	The system displays the partially updated game state.	UC-2 UC-5
RESP-17	The system determines whether every player has completed their turn during this day.	UC-2 UC-5
RESP-18	The system advances the game state.	UC-2 UC-5

UUCM-03



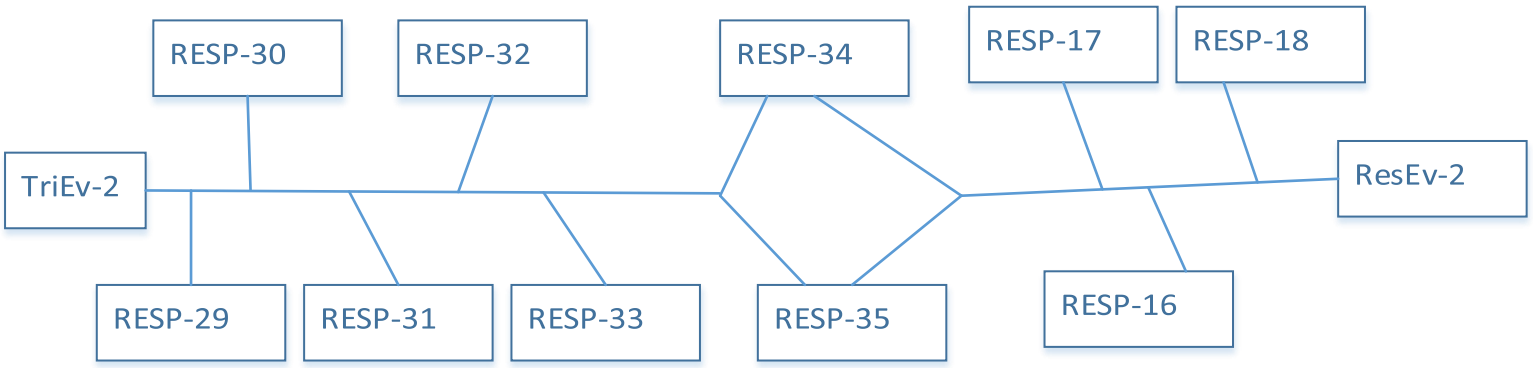
TriEv - 3	The player chooses to host a new game of Magic Realm.	
ResEv – 3	The players begin a session.	
ID	**Responsibility**	**Use Case(s)**
RESP-19	The player will choose where treasures, sounds, ghosts, dwellings, lost castle and the lost city are placed if this is cheat mode.	UC-3 UC-4
RESP-20	The player will choose their character, his starting location and the victory points needed.	UC-3 UC-4
RESP-21	The host player waits for a guest player to join.	UC-3
RESP-22	The host checks that there are enough players.	UC-3
RESP-23	The host begins the session.	UC-3
RESP-24	The host will choose if the game is in cheat mode or not.	UC-3

UUCM-04



TriEv - 4	The player chooses to join a game of Magic Realm as a client.	
ResEv – 4	The players begin the session.	
ID	**Responsibility**	**Use Case(s)**
RESP-19	The player will choose where treasures, sounds, ghosts, dwellings, lost castle and the lost city are placed if this is cheat mode.	UC-3 UC-4
RESP-20	The player will choose their character, his starting location and the victory points needed.	UC-3 UC-4
RESP-25	The system verifies whether or not the game to be joined is accessible.	UC-4
RESP-26	The system connects the player to the game as a client.	UC-4
RESP-27	The guest player waits for the host player to begin the session.	UC-4
RESP-28	The system initializes the session on behalf of the host player.	UC-4

UUCM-05



TriEv - 5	Two players end the day in the same clearing.	
ResEv – 5	Combat is finished, the day may resume.	
ID	**Responsibility**	**Use Case(s)**
RESP-16	The system displays the partially updated game state.	UC-2 UC-5
RESP-17	The system determines whether every player has completed their turn during this day.	UC-2 UC-5
RESP-18	The system advances the game state.	UC-2 UC-5
RESP-29	The players selects attack direction and strength, shield direction, evasion direction and strength.	UC-5
RESP-30	The system verifies if the attack hits the other player.	UC-5
RESP-31	The system verifies if any damage is done to the other player.	UC-5
RESP-32	The system applies damage to the other player.	UC-5
RESP-33	The system will verify that the player is still considered alive.	UC-5
RESP-34	The players decide to leave the combat.	UC-5
RESP-35	The player is removed from play.	UC-5

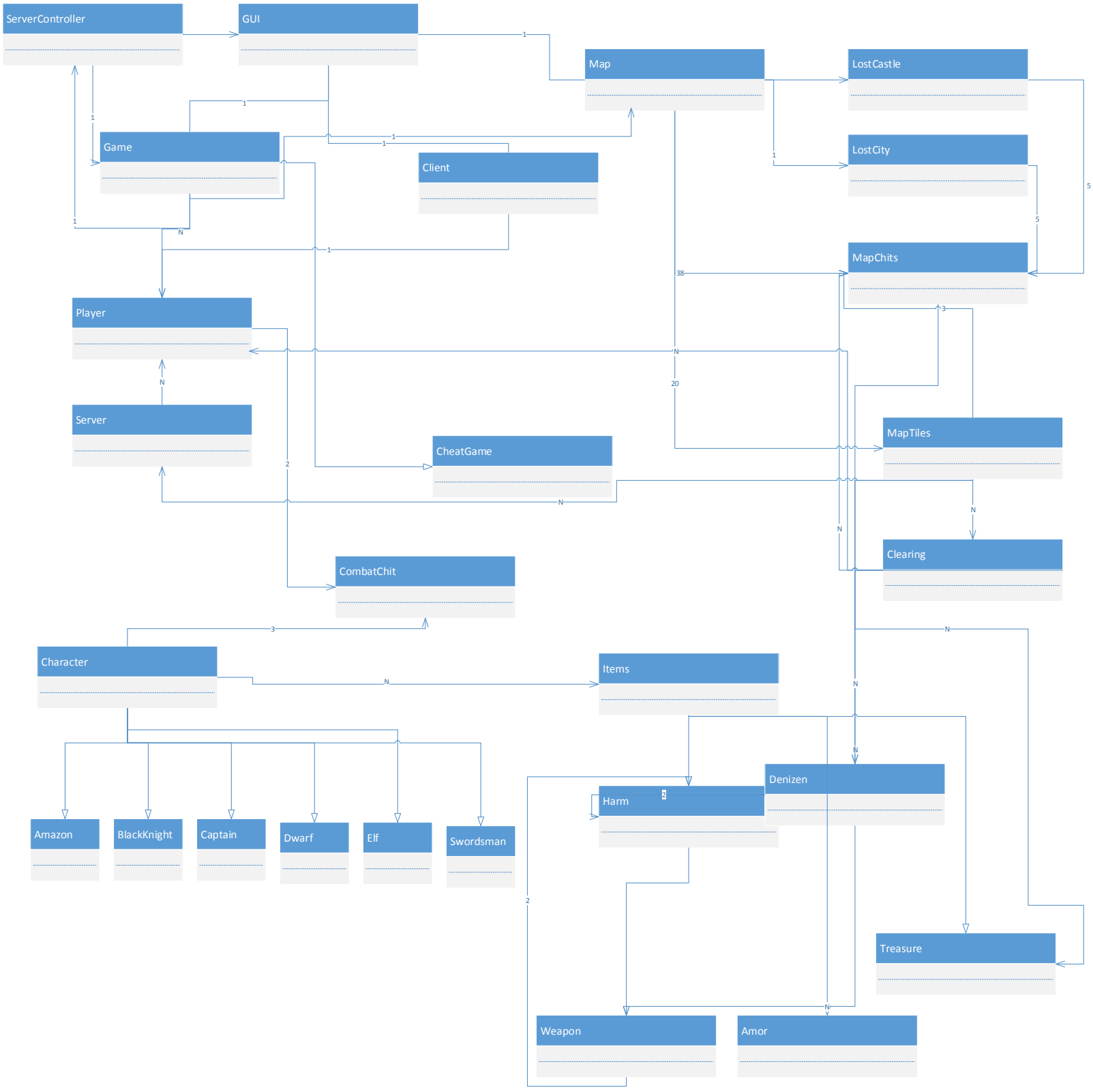
5 – Design Decisions

This section will cover the design decisions that were taken with respect to classes and objects chosen for the system. The UML diagram will be in section 5.2.

5.1 – Decisions

ID	Design Decision	Traceability
DD-1	**Strategy Pattern** The Strategy Pattern is extremely useful for implementing the game chits of all kinds and their Behaviours. We wanted to be able to reuse generic classes, rather than implement the various classes separately. Using the Strategy pattern, we could do this and keep the code very clean and simple, organizing each individual classes actions into its own inherited class. This was also used for MapTile types, charater profile types and the different kinds of items a person could carry.	Group Decision
DD-2	**Relevant statistics for each player** The game records all the resources, victory points, and other statistics with tallies of how much the player has accumulated for each. This decision reduces interface clutter, and lets the player or system instantly know how many of a type of resource they have instead of trying to count all the symbols they have.	Group Decision
DD-3	**Testing of behaviors and strategies** We were able to test how our functions were behaving and what strategies they were using. To do this, we added instances of them into a running game and looked at what they were outputting. As they were active debugging information was printed through their consoles, aiding us greatly in fixing their implementations.	Group Decision
DD-4	**Layer Pattern** Several classes utilize this pattern. They keep all of the functions related to their entities in separate files. This allows for easier code reuse.	Group Decision
DD-5	**Anti-Pattern** Several of our classes used copy/paste to be created. As such they resemble they have a blobbing problems. Where several classes contain most of the code.	Group Decision
DD-6	**Observer Pattern** We had originally attempted to use this pattern to handle the moving, adding, removing of players. To ease their implementation. However as the program got more complex the observer pattern was abandoned.	Group Decision

5.2 – Structural Model



5.3 - Client/Server Functionality

This section will detail which functionality resides on the server side of the code and what code resides on the client side.

Client	Server
Handles the user input and sends messages to the Server.	Server starts and runs the game. The majority of the functionality remains here.

6 – Object Specifications

Amazon

Name	ID	Description	Inheritance
Amazon	OS-1	Amazon character	Character

Data Member Dictionary

Variable Name	Type	Status	Procedures
-	-	-	-

Procedure Dictionary

Procedure Name	Return Type	Parameter Name	Parameter Type	Responsibilities
-	-	-	-	-

BlackKnight

Name	ID	Description	Inheritance
BlackKnight	OS-2	Black Knight character	Character

Data Member Dictionary

Variable Name	Type	Status	Procedures
-	-	-	-

Procedure Dictionary

Procedure Name	Return Type	Parameter Name	Parameter Type	Responsibilities
-	-	-	-	-

Captain

Name	ID	Description	Inheritance
Captain	OS-3	Captain character	Character

Data Member Dictionary

Variable Name	Type	Status	Procedures
-	-	-	-

Procedure Dictionary

Procedure Name	Return Type	Parameter Name	Parameter Type	Responsibilities
-	-	-	-	-

Dwarf

Name	ID	Description	Inheritance
Dwarf	OS-4	Dwarf character	Character

Data Member Dictionary

Variable Name	Type	Status	Procedures
-	-	-	-

Procedure Dictionary

Procedure Name	Return Type	Parameter Name	Parameter Type	Responsibilities
-	-	-	-	-

Elf

Name	ID	Description	Inheritance
Elf	OS-5	Elf character	Character

Data Member Dictionary

Variable Name	Type	Status	Procedures
-	-	-	-

Procedure Dictionary

Procedure Name	Return Type	Parameter Name	Parameter Type	Responsibilities
-	-	-	-	-

Swordsman

Name	ID	Description	Inheritance
Swordsman	OS-6	Swordsman character	Character

Data Member Dictionary

Variable Name	Type	Status	Procedures
-	-	-	-

Procedure Dictionary

Procedure Name	Return Type	Parameter Name	Parameter Type	Responsibilities
-	-	-	-	-

Character

Name	ID	Description	Inheritance
Character	OS-7	Each player has this as a profile	None

Data Member Dictionary

Variable Name	Type	Status	Procedures
Special	Enum	Read	-
type	String	Read/Write	getType() setType()
startSpots	String[]	Read/Write	getStartLocations()
currentLocation	Int	Read/Write	getCurrentLocation() setCurrentLocation()
Weight	Int	Read/Write	getVulnerability()
Fame	Int	Read/Write	getFame() setFame()
Notoriety	Int	Read/Write	getNotoriety() setNotoriety()
Gold	Int	Read/Write	getGold() setGold()
startGold	Int	Read/Write	getStartGold() setStartGold()
Belongings	Items[]	Read/Write	setWeapon() setDefense() getGreatTreasure() haveCloak() haveGlasses()
allyTrading	String[]	Read/Write	-
friendlyTrading	String[]	Read/Write	-

unfriendlyTrading	String[]	Read/Write	-
enemyTrading	String[]	Read/Write	-
Weapon	Weapon	Read/Write	getWeapon() setWeapon()
Defense	Armor[]	Read/Write	-
shieldActive	Boolean	Read/Write	-
Specials	Special[]	Read/Write	-
Action1	CombatChit	Read/Write	-
Action1Num	Int	Read/Write	-
Action2	CombatChit	Read/Write	-
Action2Num	Int	Read/Write	-
Action3	CombatChit	Read/Write	-
Action3Num	Int	Read/Write	-
foughtToday	Boolean	Read/Write	getFoughtToday() setFoughtToday() resetFight()
amazonUsed	Boolean	Read/Write	resetFight()
elfUsed	Boolean	Read/Write	resetFight()
cloakUsed	Boolean	Read/Write	resetFight()
glassesUsed	Boolean	Read/Write	resetFight()

Procedure Dictionary

Procedure Name	Return Type	Parameter Name	Parameter Type	Responsibilities
getVulnerability	int	-	-	Return value
getStartGold	int	-	-	Return value
setStartGold	-	g	int	Set to new value
getFoughtToday	Boolean	-	-	Return value
setFoughtToday	-	s	boolean	Set to new value
getType	String	-	-	Return value
setType	void	type	String	Set to new value
getWeapon	Weapon	-	-	Return value
setWeapon	Void	Weapon	Weapon	Set to new value
getCurrentLocation	Int	-	-	Return value
getDefense	Armor[]	-	-	Return value
getDefense	Armor	I	Int	Return value
setDefense	Void	Defense	Armor[]	Set to new value
setCurrentLocation	Void	i	Int	Set to new value
getStartLocations	String[]	-	-	Return value
getGreatTreasure	Int	-	-	Return value
getFame	Int	-	-	Return value
setFame	Void	newFame	Int	Set to new value
getNotoriety	Int	-	-	Return value
setNotoriety	Void	newNotoriety	Int	Set to new value
getGold	Int	-	-	Return value

setGold	Void	gold	Int	Set to new value
resetFight	Void	-	-	Reset values
haveCloak	Boolean	-	-	Check if it is in the belongings
haveGlasses	Boolean	-	-	Check if it is in the belongings

Armor

Name	ID	Description	Inheritance
Armor	OS-8	The armor used by characters and natives	Items

Data Member Dictionary

Variable Name	Type	Status	Procedures
Protect	Enum	Read	-
Destroyed	Boolean	Read/Write	isDestroyed() setDestroyed()
Damaged	Boolean	Read/Write	isDamaged() setDamaged()
Toughness	Int	Read/Write	getToughness() setToughness()
Intact_price	Int	Read/Write	-
Damaged_price	Int	Read/Write	-
Destroyed_price	Int	Read/Write	-
Protect	Protect	Read/Write	getDirection()

Procedure Dictionary

Procedure Name	Return Type	Parameter Name	Parameter Type	Responsibilities
isDestroyed	Boolean	-	-	Is the armor destroyed
setDestroyed	Void	destroyed	Boolean	Make armor destroyed
isDamaged	Boolean	-	-	Is the armor damaged
setDamaged	Void	damaged	Boolean	Make armor damaged
getToughness	Int	-	-	Return toughness
setToughness	Void	tough	Int	Set the toughness to the new value
getDirection	String	-	-	Return direction

ArrayUtils

Name	ID	Description	Inheritance
ArrayUtils	OS-9	Handles the functions for an Array.	None

Data Member Dictionary

Variable Name	Type	Status	Procedures
-	-	-	-

Procedure Dictionary

Procedure Name	Return Type	Parameter Name	Parameter Type	Responsibilities
Remove	Object	Array, Index	Object, Int	Remove at position index of array
getLength	Int	Array	Object	Get length of the array
Add	T[]	Array, Element	T[], T	Add element to array
copyArrayGrow1	Object	Array, newArrayComponentType	Object, Class	Create a new array in a bigger size.
indexOf	Int	Needle, haystack	String, String[]	Get the location of needle from the haystack
indexOf	Int	Needle, haystack	Player, Player[]	Get the location of needle from the haystack

Clearing

Name	ID	Description	Inheritance
Clearing	OS-10	Multiple clearings in a tile	None

Data Member Dictionary

Variable Name	Type	Status	Procedures
playersInClearing	Player[]	Read/Write	putPlayer() removePlayer()
numPlayersInClearing	Int	Read/Write	putPlayer() removePlayer()
monstersInClearing	Denizen[]	Read/Write	putDenizen() removeDenizen()

numMonstersInClearing	Int	Read/Write	putDenizen() removeDenizen()
playerDrops	PlayerDrop[]	Read/Write	putPlayerDrop() getPlayerDrop()
isDrop	Boolean	Read/Write	putPlayerDrop()
value	Int	Read/Write	getValue() setValue()
connectedTo	Int[]	Read/Write	getConnectedTo()
guardHouse	Boolean	Read/Write	-
chapel	Boolean	Read/Write	-
house	Boolean	Read/Write	-
inn	Boolean	Read/Write	-

Procedure Dictionary

Procedure Name	Return Type	Parameter Name	Parameter Type	Responsibilities
putPlayerDrop	Void	drop	PlayerDrop	Put playerDrop in the clearing
getPlayerDrop	PlayerDrop	-	-	Get playerDrop from clearing
getValue	Int	-	-	Get value
setValue	Void	Value	Int	Set value
getConnectedTo	Int[]	-	-	Which clearing this one is connected to
putPlayer	Void	Player1	Player	Put player in clearing
removePlayer	Void	Player1	Player	Remove player from clearing
putDenizen	Void	Monster	Denizen	Put denizen in clearing
removeDenizen	Void	Monster	Denizen	Remove denizen from clearing

CombatChit

Name	ID	Description	Inheritance
CombatChit	OS-11	The stats for the players actions	None

Data Member Dictionary

Variable Name	Type	Status	Procedures
Type	String	Read/Write	setType() getType()

			toString()
Time	Int	Read/Write	setTime() getTime() toString()
Effort	Int	Read/Write	setEffort() getEffort() toString() getActiveEffortChits()
Strength	Int	Read/Write	setStrength() getStrength() toString()
Fatigued	Int	Read/Write	getFatigueWoundChits()
Wounded	Int	Read/Write	getFatigueWoundChits()

Procedure Dictionary

Procedure Name	Return Type	Parameter Name	Parameter Type	Responsibilities
getType	String	-	-	Get value
setType	Void	String	String	Set value
getTime	Int	-	-	Get value
setTime	Void	I	Int	Set value
getEffort	Int	-	-	Get value
setEffort	Void	I	Int	Set value
getStrength	Int	-	-	Get value
setStrength	Void	String	String	Set value
toString	String	-	-	Return the values as a string
getActiveChits	String[]	Player, options	Player, String[]	Get all active fight chits
getActiveEffortChits	String[]	Player, options	Player, String[]	Get all active chits that cost effort
getActionChits	String[]	Player, options	Player, String[]	Get all active chits
getFatigueWoundChits	String[]	Person	Character	Get all tired and injured chits

CombatFunctions

Name	ID	Description	Inheritance
CombatFunctions	OS-12	Functions needed to	None

		carry out combat between characters	

Data Member Dictionary

Variable Name	Type	Status	Procedures
-	-	-	-

Procedure Dictionary

Procedure Name	Return Type	Parameter Name	Parameter Type	Responsibilities
resolveCombat	Void	View, player, opponent, cheating	GUI, Player, Player, Boolean	Prepare players for combat and determines who hits first
endOfRound	Void	Player, opponent	Player, Player	User picks which chits to fatigue or wound. General resets to round values.
finishCombat	Void	Attacker, defender, cheating, view	Player, Player, Boolean, GUI	Determine if attack hits defender and then remove the effort needed
attackHits	Void	Attacker, defender, cheating, view	Player, Player, Boolean, GUI	Determine harm to defender and damage to his armor
killPlayer	Void	Defender, view	Player, GUI	Remove defender from play and create a playerDrop from his belongings

Denizen

Name	ID	Description	Inheritance
Denizen	OS-13	Monsters and Natives found in the Magic Realm universe	None

Data Member Dictionary

Variable Name	Type	Status	Procedures
Name	String	Read/Write	getName()
Size	Int	Read/Write	-

Armored	Boolean	Read/Write	-
tradeType	tradeType	Read/Write	-
Weapon	Weapon	Read/Write	setWeapon()
fameBounty	Int	Read/Write	-
notorietyBounty	Int	Read/Write	-
goldBounty	Int	Read/Write	-
regCombat	Harm	Read/Write	-
aggresiveCombat	Harm	Read/Write	-
currentClearing	Int	Read/Write	getCurrentLocation() setCurrentClearing()
startClearing	Int	Read/Write	getStartLocation() setStartClearing()
regMove	Int	Read/Write	-
aggressiveMove	Int	Read/Write	-
prowling	Boolean	Read/Write	-

Procedure Dictionary

Procedure Name	Return Type	Parameter Name	Parameter Type	Responsibilities
setWeapon	Void	Weapon	Weapon	Set the value of denizen weapon
getCurrentLocation	Int	-	-	Return the value of the current location
setCurrentClearing	void	newLocation	Int	Set the current clearing value of the character
getStartLocation	Int	-	-	Get the location where this denizen started the game
setStartClearing	Void	newLocation	Int	Set the value of this denizen's start location
getName	String	-	-	Get the denizen's type

Die

Name	ID	Description	Inheritance
Die	OS-14	Functions for dice behavior	None

Data Member Dictionary

Variable Name	Type	Status	Procedures
-	-	-	-

Procedure Dictionary

Procedure Name	Return Type	Parameter Name	Parameter Type	Responsibilities
dieRoll	Int	-	-	Roll a die.
dieRollCheat	Int	-	-	Ask user to pick a die value

Harm

Name	ID	Description	Inheritance
Harm	OS-15	The stats for weapons and denizen	None

Data Member Dictionary

Variable Name	Type	Status	Procedures
Weight	Int	Read/Write	Harm()
Sharpness	Int	Read/Write	Harm()
Time	Int	Read/Write	Harm()

Procedure Dictionary

Procedure Name	Return Type	Parameter Name	Parameter Type	Responsibilities
-	-	-	-	-

Items

Name	ID	Description	Inheritance
Items	OS-16	Main class for several object types	None

Data Member Dictionary

Variable Name	Type	Status	Procedures
Notoriety_value	Int	Read/Write	-
Gold_price	Int	Read/Write	-
Fame_value	Int	Read/Write	-
name	String	Read/Write	-

Procedure Dictionary

Procedure Name	Return Type	Parameter Name	Parameter Type	Responsibilities
-	-	-	-	-

LostCastle

Name	ID	Description	Inheritance
LostCastle	OS-17	Contains 5 map chits.	None

Data Member Dictionary

Variable Name	Type	Status	Procedures
sound	Private RedChit[]	Read/Write	setSound()
treasure	Private GoldChit[]	Read/Write	setTreasure()
found	Public boolean	Read/Write	

Procedure Dictionary

Procedure Name	Return Type	Parameter Name	Parameter Type	Responsibilities
setSound	Void	S	RedChit	Set the value of sound
setTreasure	Void	S	GoldChit	Set value of treasure

LostCity

Name	ID	Description	Inheritance
LostCity	OS-18	Contains 5 map chits.	None

Data Member Dictionary

Variable Name	Type	Status	Procedures
sound	Private RedChit[]	Read/Write	setSound()
treasure	Private GoldChit[]	Read/Write	setTreasure()
found	Public boolean	Read/Write	

Procedure Dictionary

Procedure Name	Return Type	Parameter Name	Parameter Type	Responsibilities
setSound	Void	S	RedChit	Set the value of sound
setTreasure	Void	S	GoldChit	Set value of treasure

Map

Name	ID	Description	Inheritance
Map	OS-19	The gameboard	None

Data Member Dictionary

Variable Name	Type	Status	Procedures
View	GUI	Read/Write	Build() canHeMove() buildCheat() buildMapChits()
Startplacement	Boolean	Read/Write	-
mapTiles	MapTiles[]	Read/Write	getMapTiles() getMapTile() setMapTiles() resetMonsters() denizensProwling() denizenzProwlingStop() getClearing() denizensProwlingCheat()
warningsV	YellowChit[]	Read/Write	Build() buildWarningChits()
warningsW	YellowChit[]	Read/Write	Build() buildWarningChits()
warningsC	YellowChit[]	Read/Write	Build() buildWarningChits()
warningsM	YellowChit[]	Read/Write	Build() buildWarningChits()
Sites	GoldChit[]	Read/Write	getTreasure() build() buildTreasureChits()
Sounds	RedChit[]	Read/Write	getSound() build() buildSoundChits()
lostCastle	LostCastle	Read/Write	getLostCastle() build()
LostCity	LostCity	Read/Write	getLostCity() build()

Procedure Dictionary

Procedure Name	Return Type	Parameter Name	Parameter Type	Responsibilities
getMapTiles	MapTiles[]	-	-	Get all MapTiles
getMapTile	MapTiles	a	Int	Get a single tile
setMapTiles	Void	mapTiles	MapTiles[]	Set all map tiles
getSound	RedChit	a	Int	Get sound chit

getTreasure	GoldChit	a	Int	Get treasure chit
getLostCastle	LostCastle	-	-	Get value
getLostCity	LostCity	-	-	Get value
build	Void	-	-	Build map
buildSoundChits	Void	-	-	Build sound chits
buildTreasureChits	Void	-	-	Build treasure chits
buildWarningChits	Void	-	-	Build warning chits
moveCharacters	Void	Player1, newLocation	Player, int	Move Character
moveDenizen	Void	Monster, newClearing, tile	Denizen, int, int	Move denizen
returnDenizensToStart	Void	-	-	Return denizen to start positions
resetMonsters	Void	-	-	Reset monsters to start positions
denizensProwling	Void	-	-	Move prowling monsters
denizensProwlingStop	Void	-	-	Stop the monsters from prowling
giveWholeTreasure	Void	Player, shinyStuff	Player, Items[]	Give whole treasure to player
giveOneTreasure	Void	Player, shinies	Player, Items[]	Give a single treasure to player
canHeMove	Boolean	oldLocation, newLocation, player	Int, Int, Player	Can player move to the clearing
getClearing	Clearing	Tile, clearing	Int, Int	Return clearing
buildCheat	Void	-	-	Build chits
buildMapChits	Void	-	-	Build map
denizensProwlingCheat	Void	-	-	Move prowling monsters
resetGhostsCheat	Void	-	-	Move ghosts to the start positions
randomSmallTreasure	Treasure	-	-	Return treasure
randomLargeTreasure	Treasure	-	-	Return treasure

MapChits

Name	ID	Description	Inheritance
MapChits	OS-20	Different tokens which may be found in clearings and tiles	None

Data Member Dictionary

Variable Name	Type	Status	Procedures
Clearing	Int	Read/Write	-
Type	String	Read/Write	-
Found	Boolean	Read/Write	-

Procedure Dictionary

Procedure Name	Return Type	Parameter Name	Parameter Type	Responsibilities
-	-	-	-	-

MapTiles

Name	ID	Description	Inheritance
MapTiles	OS-21	Map has 20 of these	None

Data Member Dictionary

Variable Name	Type	Status	Procedures
Clearing	Clearing[]	Read/Write	setGuardHouse() putNativeGuards() setHouse() putNativeSoldiers() setInn() putNativeRogues() setChapel() putNativeOrder() putGhosts() secretRoads()
playersInTile	Player[]	Read/Write	putPlayer() removePlayer()
monstersInTile	Denizen[]	Read/Write	putDenizen() removeDenizen()
numMonstersInTile	Int	Read/Write	putDenizen() removeDenizen()
Warning	YellowChit	Read/Write	getWarning() setWarning()
Sound	RedChit	Read/Write	setSound()

Treasure	GoldChit	Read/Write	getTreasure() setTreasure()
lostCastle	Boolean	Read/Write	setLostCastle()
lostCity	Boolean	Read/Write	setLostCity()
topLeft	Int	Read/Write	-
topRight	Int	Read/Write	-
Left	Int	Read/Write	-
Right	Int	Read/Write	-
bottomLeft	Int	Read/Write	-
bottomRight	Int	Read/Write	-
Type	String	Read/Write	getType() setType()
Guard	GreatSwordsman[]	Read/Write	putNativeGuards()
Order	GreatSwordsman[]	Read/Write	putNativeOrder()
Rogues	GreatSwordsman[]	Read/Write	putNativeRogues()
Soldiers	GreatSwordsman[]	Read/Write	putNativeSoldiers()
Ghosts	Ghost[]	Read/Write	putGhosts()
ghostTile	Boolean	Read/Write	isGhostTile() setGhostTile()

Procedure Dictionary

Procedure Name	**Return Type**	**Parameter Name**	**Parameter Type**	**Responsibilities**
isGhostTile	Boolean	-	-	Does this tile have ghosts
setGhostTile	Void	ghostTile	Boolean	Put ghosts in a tile
getType	String	-	-	Return value
setType	Void	type	String	Set value
putPlayer	Void	Player1	Player	Put player in tile
removePlayer	Void	Player1	Player	Remove player from tile
putDenizen	Void	monster	Denizen	Put denizen in tile
removeDenizen	Void	monster	Denizen	Remove denizen from tile
getWarning	YellowChit	-	-	Return value
getTreasure	GoldChit	-	-	Return value
setWarning	Void	warning	YellowChit	Set value
setSound	Void	s	RedChit	Set value
setTreasure	Void	s	GoldChit	Set value
setLostCastle	Void	-	-	Set value
setLostCity	Void	-	-	Set value
setGuardHouse	Void	I, tile	Int, MapTiles	Set value
putNativeGuards	Void	tile	MapTiles	Put guards

setHouse	Void	I, valley	Int, MapTiles	Set value
putNativeSoldiers	Void	tile	MapTiles	Put soldiers
setInn	Void	I, valley	Int, MapTiles	Set value
putNativeRogues	Void	valley	MapTiles	Put rogues
setChapel	Void	I, valley	Int, MapTiles	Set value
putNativeOrder	Void	valley	MapTiles	Put Order
setGhosts	Void	I, valley	Int, MapTiles	Put ghosts
putGhosts	Void	Valley, clearin	MapTiles, int	Put ghosts
setSoundTreasureCheat	Void	mapChit	MapChits	Set value
secretRoads	Void	currentTile	Int	Unlock secret paths

PlayerActions

Name	ID	Description	Inheritance
PlayerActions	OS-22	Functions to handle player actions	None

Data Member Dictionary

Variable Name	Type	Status	Procedures
-	-	-	-

Procedure Dictionary

Procedure Name	Return Type	Parameter Name	Parameter Type	Responsibilities
restingAction	Void	Player	Player	Player chooses to rest.
lootingAction	Void	Player, cheating, map, currentTile, currentClearing	Player, Boolean, Map, Int, Int	Player chooses to loot
Loot	Void	Player, map, cheating, treasure, shinyStuff	Player, Map, Boolean, MapChits, Items[]	Player chooses to loot.
locatingAction	Void	Player, cheating, map, currentTile	Player, Boolean, Map, Int	Player chooses to locate.
moveAction	Void	Player, cheating, map, view	Player, Boolean, Map, GUI	Player chooses to move.

Treasure

Name	ID	Description	Inheritance
Treasure	OS-23	Valuables found in treasure sites	Items

Data Member Dictionary

Variable Name	Type	Status	Procedures
Gt_Treasure	Boolean	Read/Write	isGt_Treasure() setGt_Treasure()
Lg_Treasure	Boolean	Read/Write	-
Fame_value	Int	Read/Write	-
Notoriety_value	Int	Read/Write	-
Gold_price	Int	Read/Write	-
Intact_price	Int	Read/Write	-
Damaged_price	Int	Read/Write	-
Destroyed_price	Int	Read/Write	-

Procedure Dictionary

Procedure Name	Return Type	Parameter Name	Parameter Type	Responsibilities
isGt_Treasure	Boolean	-	-	Determine if treasure is great
setGt_Treasure	Void	gt_Treasure	Boolean	Set the treasure to great

Weapon

Name	ID	Description	Inheritance
Weapon	OS-24	The armaments used by characters, monsters and natives	Items

Data Member Dictionary

Variable Name	Type	Status	Procedures
alerted	Boolean	Read/Write	setAlert() getAlert()
weaponLength	Int	Read/Write	-
Missile	Boolean	Read/Write	-
alertedHarm	Harm	Read/Write	-
unalertedHarm	Harm	Read/Write	-
Hands	Int	Read/Write	-
goldCost	Int	Read/Write	-

Procedure Dictionary

Procedure Name	Return Type	Parameter Name	Parameter Type	Responsibilities
setalert	Void	v	Boolean	Set the alert value to new value v
getAlert	Boolean	-	-	Get the alert value
missileRoll	Int	-	-	Handle the effects on harm from a missile weapon
missileRollCheat	Int	-	-	Handle the effects on harm from a missile weapon in cheat mode

ArmorDialog

Name	ID	Description	Inheritance
ArmorDialog	OS-25	Ask user to pick direction	None

Data Member Dictionary

Variable Name	Type	Status	Procedures
-	-	-	-

Procedure Dictionary

Procedure Name	Return Type	Parameter Name	Parameter Type	Responsibilities
getArmor	Void	Player	Player	Ask user to pick direction

CombatDialog

Name	ID	Description	Inheritance
CombatDialog	OS-26	Ask user to pick direction and strength	None

Data Member Dictionary

Variable Name	Type	Status	Procedures
-	-	-	-

Procedure Dictionary

Procedure Name	Return Type	Parameter Name	Parameter Type	Responsibilities
getAttack	Void	Player	Player	Ask user to pick direction and strength

EvadeDialog

Name	ID	Description	Inheritance
EvadeDialog	OS-27	Ask user to pick direction and strength	None

Data Member Dictionary

Variable Name	Type	Status	Procedures
-	-	-	-

Procedure Dictionary

Procedure Name	Return Type	Parameter Name	Parameter Type	Responsibilities
getEvasion	Void	Player	Player	Ask user to pick direction and strength

GUI

Name	ID	Description	Inheritance
GUI	OS-28	The display for the user	None

Data Member Dictionary

Variable Name	Type	Status	Procedures
Game	Game	Read/Write	-
Map	Map	Read/Write	displayClearing() getSoundTreasureCheat() buildCityCheat() buildCastleCheat() getWarningCheat()

			updateMap()
serverIP	String	Read/Write	Connect()
Client	Client	Read/Write	Connect() ACTION_START()
Possibilities	String[]	Read/Write	avaiableChars()
Move	Boolean	Read/Write	mousePressed() getNewLocation()
Pause	Boolean	Read/Write	mousePressed() recordTurn()
clickedLocation	Int	Read/Write	displayClearing() getNewLocation()
MainWindow	JFrame	Read/Write	initilizeWindow()
Map	JPanel	Read/Write	tileBuilder() genValley() genWoods() genOther() buildMap() initilizeWindow() initPlayers()
scrollPane	JScrollPane	Read/Write	initilizeWindow()
mapTiles	JLabel[]	Read/Write	buildMap()
clearingTiles	JLabel[][]	Read/Write	genValley() genWood() genOther() buildBuildings() buildMap()
tileCount	Int	Read/Write	tileBuilder() buildMap()
Players	JPanel	Read/Write	createPlayer() chooseStart() numOfPlayers() doYouWantToTrade() whichSearchTable() displayClearing() initilizeWindow()
Date	JPanel	Read/Write	initilizeWindow()
Instruction	JPanel	Read/Write	recordTurn() initilizeWindow()
jlPlayers	JList	Read/Write	initilizeWindow()
spPlayers	JScrollPane	Read/Write	initilizeWindow()
startButton	JButton	Read/Write	initilizeWindow()
dLabel	JLabel	Read/Write	chandeDate() initilizeWindow()
moveLabel	JLabel	Read/Write	recordTurn() initilizeWindow()
X	Int	Read/Write	-

Y	Int	Read/Write	-
tileX	Int	Read/Write	-
tileY	Int	Read/Write	-
Player	JLabel[]	Read/Write	initLabels() initPlayers()
playerX	Int	Read/Write	mousePressed() getPlayerX() setPlayerX()
playerY	Int	Read/Write	mousePressed() getPlayerY() setPlayerY()

Procedure Dictionary

Procedure Name	**Return Type**	**Parameter Name**	**Parameter Type**	**Responsibilities**
getMap	Map	-	-	Return value
initLabels	Void	-	-	Initiate the labels
setMap	Void	M	Map	Set to new value
Connect	Void	p	Player	Connect the client
main	Void	Args	String[]	Start the game
tileBuilder	Void	Path, xOffset, yOffset	String, int, int	Create map tiles
createPlayer	String	-	-	Create a new player
chooseStart	Int	currPlayer	Player	Choose where to start
numOfPlayers	Int	-	-	Ask users
generateClearing	Void	-	-	Generate clearing
genValley	Void	See code	See code	Build pictures
genWoods	Void	See code	See code	Build pictures
genOther	Void	See code	See code	Build pictures
buildBuildings	Void	S, tile, clearing	String, int, int	Build the buildings
mouseClicked	Void	e	MouseEvent	Mouse functions
mousePressed	Void	e	MouseEvent	Mouse functions
mouseReleased	Void	e	MouseEvent	Mouse functions
mouseEntered	Void	e	MouseEvent	Mouse functions
mouseExited	Void	e	MouseEvent	Mouse functions

refresh	Void	labelText	String	Refresh display
recordTurn	Void	Player, phasesAvailable, gameMap	Player, int, Map	Ask user to build turn
hideMapChits	void	-	-	Hide chits
revealMapChits	Void	I	Int	Reveal chits
trading	Void	gameMap, player1	Map, Player	How to trade
doYouWantToTrade	Boolean	-	-	Ask user
whichSearchTable	String	-	-	Pick which search table
revealTreasure	Void	-	-	Reveal the treasure
displayScore	Void	Gamers	Player[]	Display the score
displayClearing	Void	S	String	Display the clearing
getNewLocation	Int	-	-	Return value
setMove	Void	Moving	Boolean	Set to new value
cheatMode	Boolean	-	-	Do you want to engage
getSoundTreasureCheat	MapChits	tileType, tileNum	String, int	Return value
buildCityCheat	Void	-	-	Build
buildCastleCheat	Void	-	-	Build
getWarningCheat	YellowChit	tileType, tileNum	String, int	Return value
diceAnswer	int	-	-	Let user pick
pickLocationsDwellingsCheat	Void	See code	See code	Pick where the buildings are
updateMap	Void	m	Map	Update the Map
convertNameToPosition	Int[]	Q	String[]	Return name
changeDate	Void	s	String	Change
fightWho	Player	Clearing	Clearing	Who do you want
selectFightGear	Void	player	Player	Pop up for fight
getPlayerX	Int	-	-	Return value
getPlayerY	Int	-	-	Return value
setPlayerX	Void	x	Int	Set to new value
setPlayerY	Void	y	Int	Set to new value
buildMap	Void	-	-	Build map
initializeWindow	Void	-	-	Make window
initPlayers	Void	-	-	Initialize players
avaiableChars	Void	X	Socket	Display

				message
combatMessage	Void	message	String	Display message
selectFatigue	String	Person	Player	Pick fatigue chit
displayMessage	String	Message	String	Pop-up
ACTION_START()	Void	-	-	Send
lootChoices	Boolean	-	-	What do you want to find
whatFound	Boolean	currentTile	Int	What do you want to find
victoryPoints	Int	String, values	String, int[]	Victory points

ServerGUI

Name	ID	Description	Inheritance
ServerGUI	OS-29		None

Data Member Dictionary

Variable Name	Type	Status	Procedures
-	-	-	-

Procedure Dictionary

Procedure Name	Return Type	Parameter Name	Parameter Type	Responsibilities
showServerIP	Void	String	String	Display Message

CheatGame

Name	ID	Description	Inheritance
CheatGame	OS-30	Play a game while being able to choose die rolls and starting location of map chits	Game

Data Member Dictionary

Variable Name	Type	Status	Procedures
View	GUI	Read/Write	CheatGame() startCheatGame() doTurnCheat() createPlayers()
Players	Player[]	Read/Write	startCheatGame() createPlayers()
numOfPlayers	Int	Read/Write	CheatGame()

			startCheatGame() createPlayers()
map	Map	Read/Write	CheatGame() startCheatGame() doTurnCheat()

Procedure Dictionary

Procedure Name	Return Type	Parameter Name	Parameter Type	Responsibilities
startCheatGame	Void	-	-	Start the gamein cheat mode
doTurnCheat	Void	Player	Player	A single player turn
createPlayers	Void	-	-	Create and assign players

Client

Name	ID	Description	Inheritance
Client	OS-31	Each player has a client if they did not host the game	None

Data Member Dictionary

Variable Name	Type	Status	Procedures
SOCK	Socket	Read/Write	Run()
INPUT	Scanner	Read/Write	Run() RECEIVE()
gui	GUI	Read/Write	-
SEND	Scanner	Read/Write	-
OUT	PrintWriter	Read/Write	Run() SEND()
player	Player	Read/Write	createPlayer()

Procedure Dictionary

Procedure Name	Return Type	Parameter Name	Parameter Type	Responsibilities
Run	Void	-	-	Start the scanner
CheckStream	Void	-	-	Keep looking for input
RECEIVE	Void	-	-	Check for message from server
SEND	Void	x	String	Send message to

				server
createPlayer	void	message	String	Build a player

Game

Name	ID	Description	Inheritance
Game	OS-32	Play a normal game	None

Data Member Dictionary

Variable Name	Type	Status	Procedures
View	GUI	Read/Write	Game() startGame() gameOver() combat() doTurn() createPlayers() determineStart()
Players	Player[]	Read/Write	startGame() gameOver() combat() createPlayers()
numOfPlayers	Int	Read/Write	Game() startGame() gameOver() combat() createPlayers()
map	Map	Read/Write	Game() startGame() combat() doTurn()

Procedure Dictionary

Procedure Name	Return Type	Parameter Name	Parameter Type	Responsibilities
startGame	Void	-	-	Do 28 days
gameOver	Void	-	-	Display score
Combat	Void	cheating	Boolean	The player enters combat
doTurn	Void	player	Player	The player does a turn
createPlayers	Void	-	-	Create array of players
shufflePlayers	Void	Players2	Player[]	Randomize the players array

determineStart	Int	S, player	String, Player	Put the player in a dwelling

Player

Name	ID	Description	Inheritance
Player	OS-33	The client	None

Data Member Dictionary

Variable Name	Type	Status	Procedures
greatTreasureVicPoint	Int	Read/Write	recordVictoryRequirments() calculateScore()
fameVicPoint	Int	Read/Write	recordVictoryRequirments() calculateScore()
notorietyVicPoint	Int	Read/Write	recordVictoryRequirments() calculateScore()
goldVicPoint	Int	Read/Write	recordVictoryRequirments() calculateScore()
greatTreasureVicPointNeeded	Int	Read/Write	recordVictoryRequirments() calculateScore()
fameVicPointNeeded	Int	Read/Write	recordVictoryRequirments() calculateScore()
NotorietyVicPointNeeded	Int	Read/Write	recordVictoryRequirments() calculateScore()
goldVicPointNeeded	Int	Read/Write	recordVictoryRequirments() calculateScore()
instanceCounter	Int	Read/Write	-
Playernum	Int	Read/Write	getPlayerNum()
amazonIcon	ImageIcon	Read/Write	-
bknightIcon	ImageIcon	Read/Write	-
captainIcon	ImageIcon	Read/Write	-
dwarfIcon	ImageIcon	Read/Write	-
elfIcon	ImageIcon	Read/Write	-
swordsmanIcon	ImageIcon	Read/Write	-
Profile	Character	Read/Write	doAction() calculateScore() getProfile() getCurrentLocation() setCurrentLocation() doActionCheat() choiceOfFightChits() choiceOfFatigueWoundChits()
Hidden	Boolean	Read/Write	doAction() doActionCheat()

Alive	Boolean	Read/Write	-
effortThisRound	Int	Read/Write	-
combatAttackDirection	String	Read/Write	getCombatAttackDirection() setCombatAttackDirection()
Attack	CombatChit	Read/Write	getAttack() setAttack()
shieldDirection	String	Read/Write	setShieldDirection() getShieldDirection()
evadeDirection	String	Read/Write	getEvadeDirection() setEvadeDirection()
Evade	CombatChit	Read/Write	getEvade() setEvade()
phasesForToday	Int	Read/Write	getPhasesForToday() setPhasesForToday()
numPhases	Int	Read/Write	setPhaseActions() setPhaseActions()
phaseActions	String[]	Read/Write	getPhaseActions() getPhaseAction() setPhaseActions() setPhaseActions()

Procedure Dictionary

Procedure Name	Return Type	Parameter Name	Parameter Type	Responsibilities
setCombatAttackDirection	Void	combatAttackDirection	String	Set value
getCombatAttackDirection	String	-	-	Get value
getAttack	CombatChit	-	-	Get value
setAttack	Void	at	CombatChit	Set value
setShieldDirection	Void	direct	String	Set value
getShieldDirection	String	-	-	Get value
setEvadeDirection	Void	evadeDirection	String	Set value
getEvadeDirection	String	-	-	Get value
setEvade	Void	ev	CombatChit	Set value
getEvade	CombatChit	-	-	Get value
getPlayerNum	Int	-	-	Get value
setPhasesForToday	Void	phasesForToday	int	Set value
getPhasesForToday	Int	-	-	Get value
setPhaseActions	Void	phaseActions1	String	Set value
getPhaseActions	String[]	-	-	Get value
setPhaseActions	Void	phaseActions1, a	String, int	Set value
getPhasesActions	String	-	-	Get value
doAction	Void	Action, map, game	String, Map, Game	Do the actions

rearangeBelongings	Void	-	-	Not in use
isThereOthersInClearing	Boolean	Map, currentTile, currentClearing	Map, int, int	Check
recordVictoryRequirments	Void	-	-	Record
calculateScore	Int	-	-	Calculate and return
getProfile	Character	-	-	Get value
setCurrentLocation	Void	newLocation	int	Set value
getCurrentLocation	Int	-	-	Get value
doActionCheat	Void	Action, map, cheatGame	String, Map, CheatGame	Do the actions
choiceOfFightChits	Void	response	String	Figure out choices
choiceOfFatigueWoundChits	Void	Response, fatigue	String, boolean	Figure out choices
increaseFatigueWoundChits	Void	Action, fatigue	CombatChit, boolean	Figure out choices
resetFightGear	Void	-	-	Reset
setNeededVictoryPoints	Void	A,b,c,d	Int, int,int,int	Set value
getNeededVictoryPoints	String	-	-	Get value

Server

Name	ID	Description	Inheritance
Server	OS-34	Connects the clients to the game	None

Data Member Dictionary

Variable Name	Type	Status	Procedures
Characters	ArrayList<String>	Read/Write	-
ConnectionArray	ArrayList<Socket>	Read/Write	-
CurrentPlayers	ArrayList<Player>	Read/Write	-
PORT	Int	Read/Write	-

Procedure Dictionary

Procedure Name	Return Type	Parameter Name	Parameter Type	Responsibilities
Main	Void	args	String[]	Start the game

ServerController

Name	ID	Description	Inheritance
ServerController	OS-35	Controller for the server thread	None

Data Member Dictionary

Variable Name	Type	Status	Procedures
CurrentPlayers	ArrayList<Player>	Read/Write	handleMessage() startGame()
SOCK	Socket	Read/Write	ServerController() checkConnection() run()
INPUT	Scanner	Read/Write	run()
OUT	PrintWriter	Read/Write	Run() handleMessage() startGame()
	String	Read/Write	Run()
g	Game	Read/Write	startGame()

Procedure Dictionary

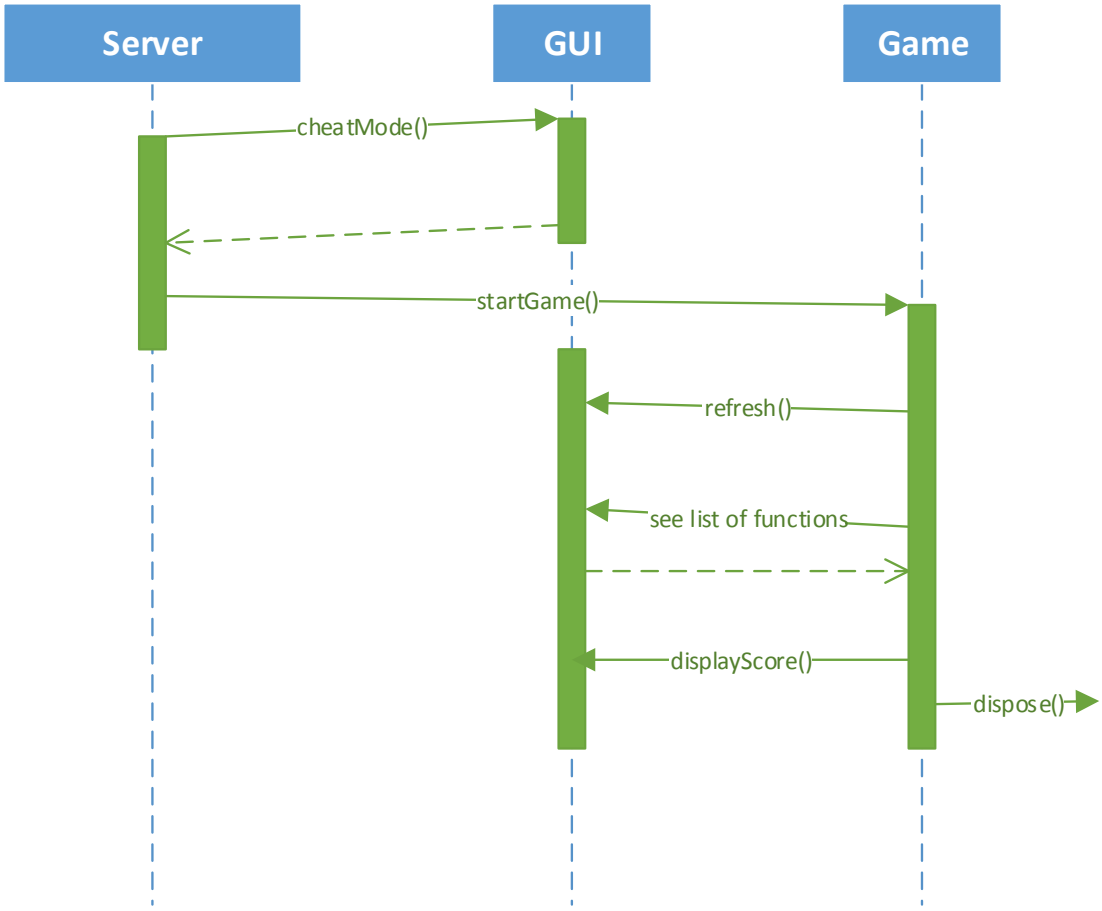
Procedure Name	Return Type	Parameter Name	Parameter Type	Responsibilities
checkConnection	Void	-	-	Check to see if still connected
Run	Void	-	-	Send a message to all clients
handleMessage	Void	Message	String	Message from the client
startGame	Void	-	-	Check if we have all players then start game

7 – Interaction Diagrams

Below are the UML Interaction Diagrams. Each corresponds to the previous bound use case maps in section 4.4.

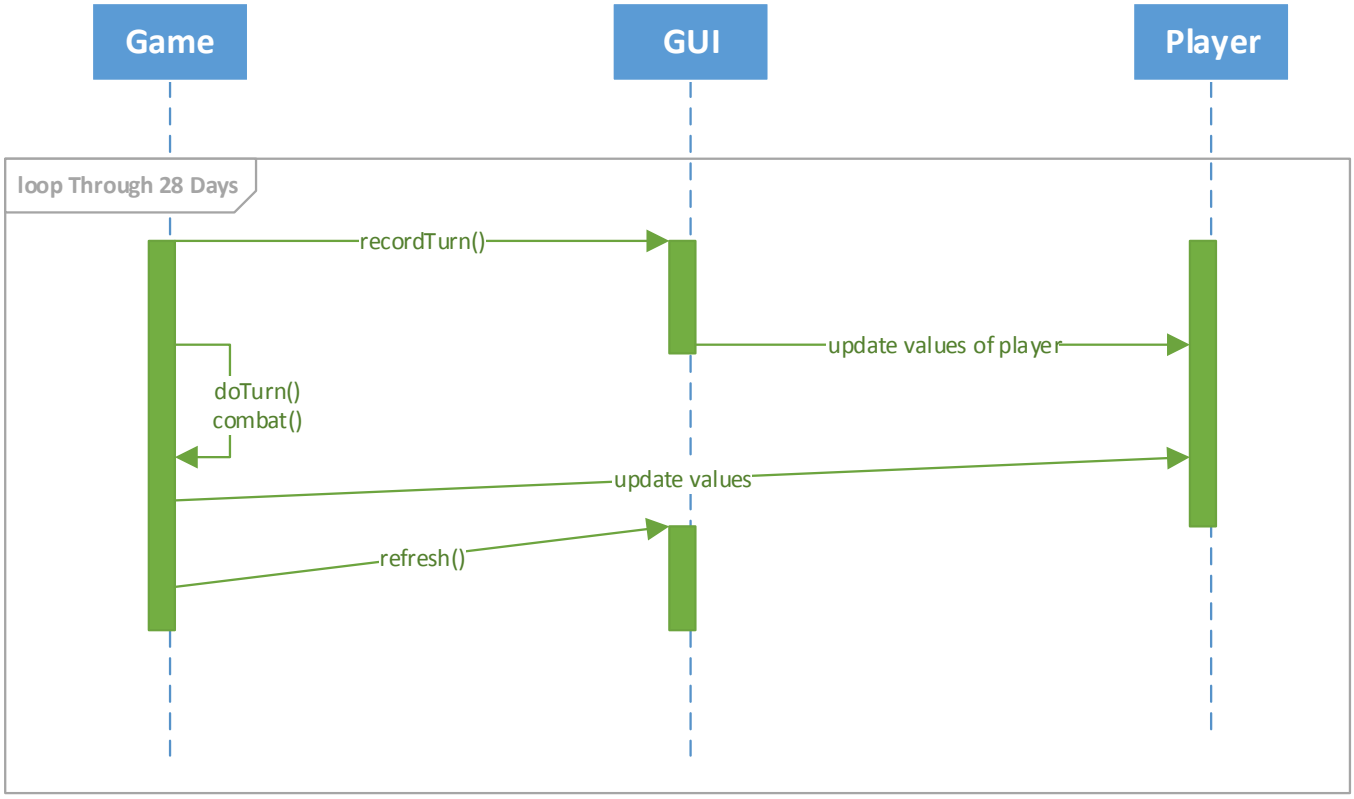
InDi-1

Corresponds to UC-1: A player starts and plays a game of Magic Realm.



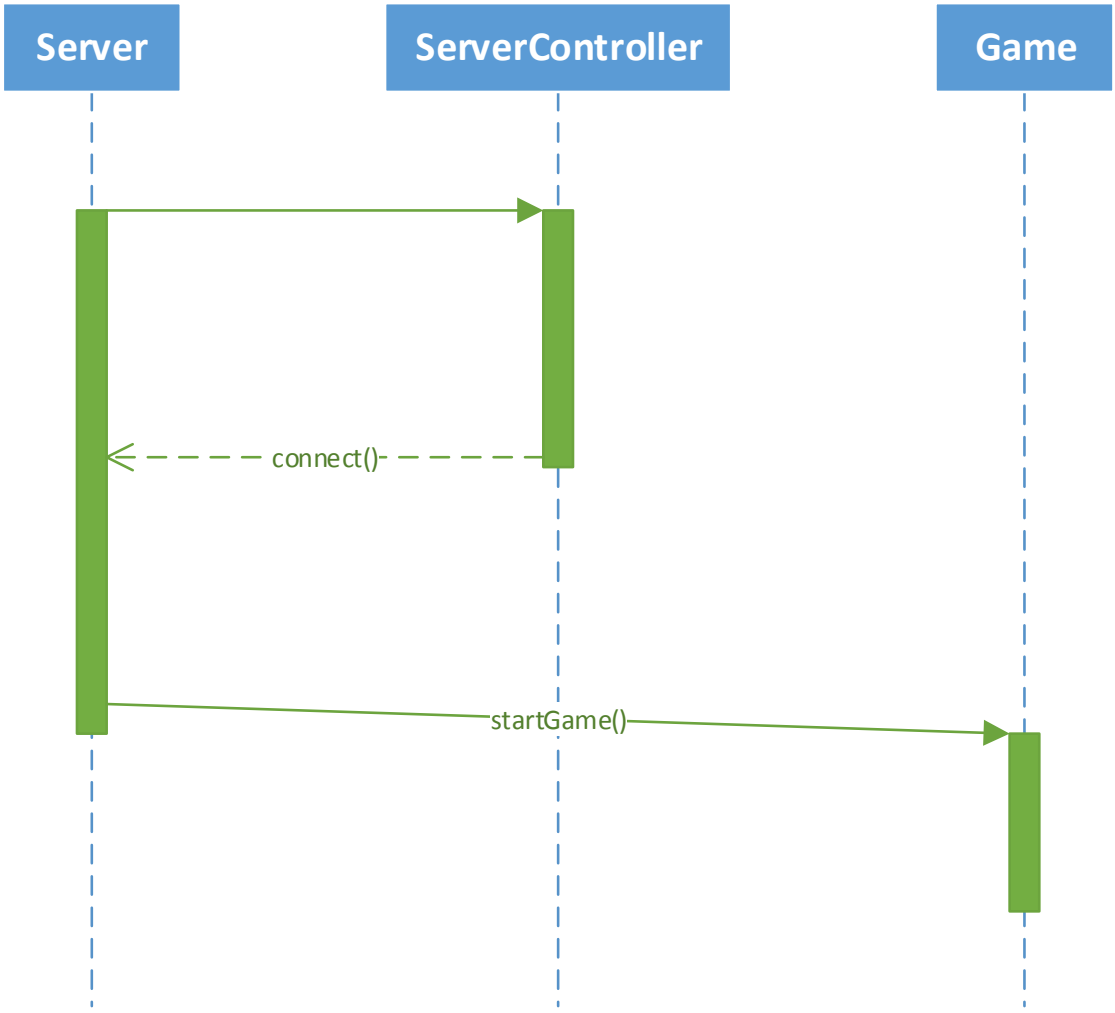
InDi-2

Corresponds to UC-2: During a game of Magic Realm all players progress through the day concurrently.



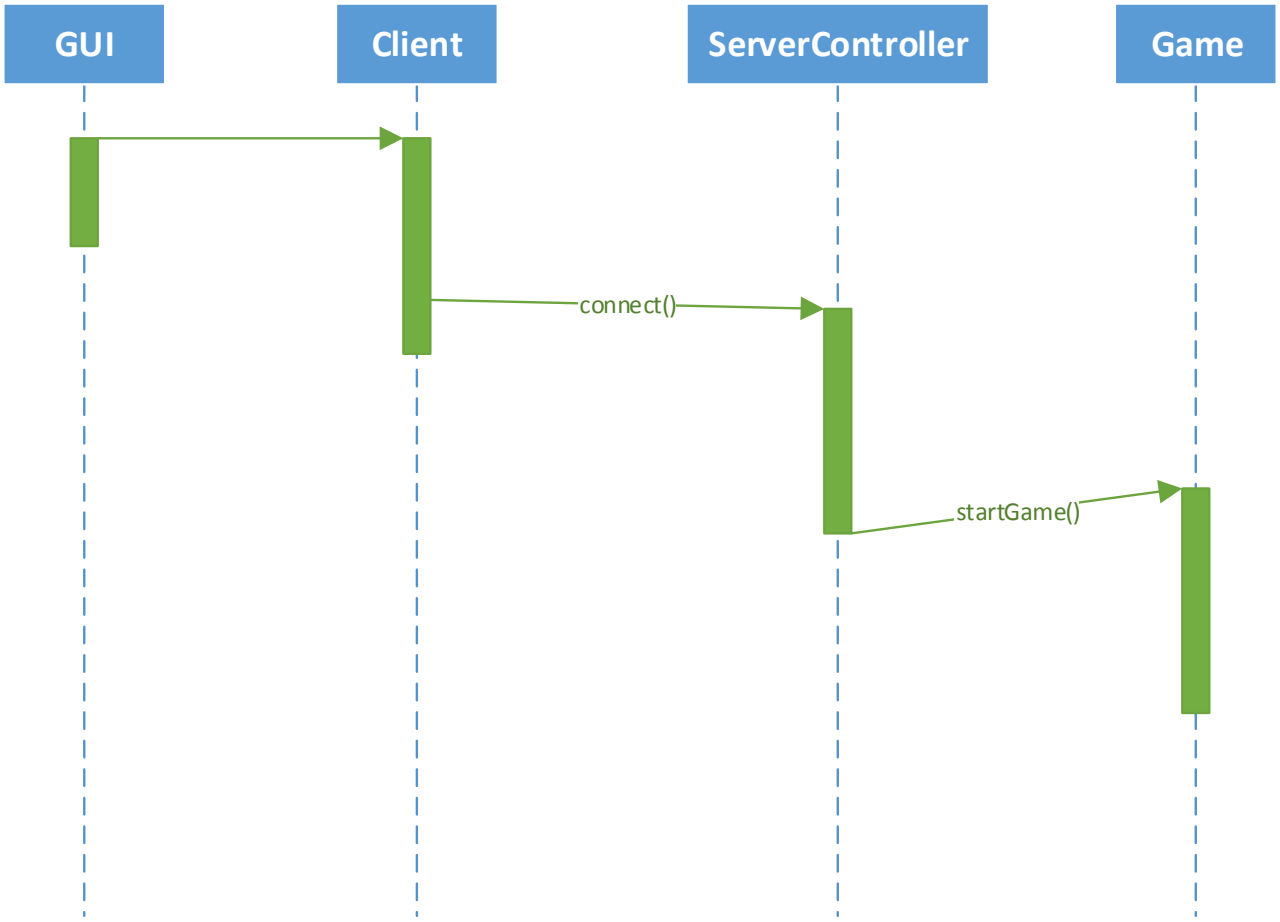
InDi-3

Corresponds to UC-3: A player sets up a game of Magic Realm.



InDi-4

Corresponds to UC-4: A player joins a game of Magic Realm as a client.



InDi-5

Corresponds to UC-5: A player fights against another player.