CS 246—Assignment 5, Group Project (Spring 2017)
Sorcery

Sean Harrap

Due Date 1: Monday, July 17th, 5pm
Due Date 2: Tuesday, July 25th, 11:59pm

DO NOT EVER SUBMIT TO MARMOSET WITHOUT COMPILING AND TESTING FIRST. If your final submission doesn't compile, or otherwise doesn't work, you will have nothing to show during your demo. Resist the temptation to make last-minute changes. They probably aren't worth it.

This project is intended to be doable by three people in two weeks. Because the breadth of students' abilities in this course is quite wide, exactly what constitutes two weeks' worth of work for three students is difficult to nail down. Some groups will finish quickly; others won't finish at all. We will attempt to grade this assignment in a way that addresses both ends of the spectrum. You should be able to pass the assignment with only a modest portion of your program working. Then, if you want a higher mark, it will take more than a proportionally higher effort to achieve, the higher you go. A perfect score will require a complete implementation. If you finish the entire program early, you can add extra features for a few extra marks.

Above all, MAKE SURE YOUR SUBMITTED PROGRAM RUNS. The markers do not have time to examine source code and give partial correctness marks for non-working programs. So, no matter what, even if your program doesn't work properly, make sure it at least does something.

1 The Basics

In this project, you will produce the game Sorcery, a card game based on collectible card games such as "Hearthstone: Heroes of Warcraft" and "Magic: the Gathering." If you have never played either of these games before, Hearthstone is free and might help you familiarize yourself with the genre.

Sorcery is played on the terminal, with commands issued via standard input or supplied from a file and output printed to standard output.

Watch out: This project involves a number of details, and going about it in the wrong order may cause you to spend lots of time while making relatively little progress in terms of marks. You should read the advice section at the end before starting to implement your project to get some guidelines on how to approach it.

1.1 Basic Definitions

The following words have special meanings in Sorcery:

- **Card:** *Cards* are the basic objects in Sorcery, making up players' decks, hands, and graveyards.
- **Deck:** The players' *deck* is a collection of cards which they may *draw* from.

- **Board:** A player's *board* is a collection of cards which they have played and which have not been moved to another zone.
- **Graveyard:** A player's *graveyard* is a collection of minions which have died.
- **Hand:** The players' *hand* is a collection of up to 5 cards which they may play.
- **Draw:** To *draw*, a player takes a card from their deck and puts it into their hand. A player may only draw if their hand is not full.
- **Owner:** The *owner* of a card is the player whose hand, deck, graveyard, or board it is in.
- **Type:** A card's *type* is one of "minion," "enchantment," "ritual," or "spell."
- **Minion:** A *minion* is a card representing a character or creature which will help you achieve victory.
- **Die:** When a minion *dies*, it is moved from its owner's board to their graveyard.
- **Magic:** A player's *magic* is the main resource they use to play cards and use special abilities.
- **Trigger:** *Triggers* are effects which occur when certain conditions are met.
- **Active Player:** The *active player* is the player whose turn it is. The other player is the *inactive player* (or non-active player).
- **Shuffle:** *Shuffling* a deck randomizes the order of the cards in the deck.
- **APNAP Order:** *APNAP order*, or "active player, non-active player order" is the default order in which simultaneous effects occur (for example, the order in which Blizzard damages minions or "a minion enters play" triggers activate). The order is:

  1. First, the minions owned by the active player, in left-to-right (oldest-played to newest-played) order.
  2. Next, the ritual owned by the active player.
  3. Finally, repeat the above two steps in the same order for the non-active player.

1.2 Basic Gameplay

The game's objective is to reduce the opposing player's life to 0, at which point the game ends. The game begins by first asking both players for their names. It then shuffles both player's decks. Once the decks are shuffled, the game begins with player 1. Both players start with 20 life, 4 cards in their hand, and 3 magic.

For the rest of the game, players alternate turns. A player's turn consists of the following:

- The player gains 1 magic. There is no limit to the amount of magic a player may have.
- The player draws a card if their deck is nonempty and their hand is not full. Otherwise, this step is simply skipped.
- Any "At the start of the turn" effects occur.
- The player is allowed to take actions until they pass.
- Any "At the end of the turn" effects occur.

Exception: Player 1 does not gain magic or draw a card on their first turn.

2 Cards

Every card has a name and a cost. Other than that, card effects are determined by their type.

2.1 Spells

Spells are the simplest type of card. A spell simply changes the game in some way (such as by increasing its caster's life by 5 or killing a chosen minion) and is then removed from the game.

2.2 Minions

Minions are the main card type in the game and a player's primary means of achieving victory. When a minion is played, it is moved from the player's hand to one of the player's 5 minion slots on the board. If all 5 slots are occupied, the minion cannot be played. Minions occupy the leftmost spots on the board and new minions are always added to the right of older minions.

In addition to the attributes they share with other card types, minions have *attack* and *defence* values. Attack and defence represent a minion's combat strength. If a minion's defence is ever 0 or less it immediately dies and is moved from the board to the top of the graveyard.

Definition: An *x/y minion* is a minion with x attack and y defence.

Minions may take one action per turn (including the turn they enter play), where an action is any of:

- Attacking the opposing player. The opposing player loses life equal to the attack value of the minion.
- Attacking an opposing minion. Both minions damage one another: minion A reduces minion B's defence by minion A's attack, and then minion B damages A in the same way.
- Using an *activated ability*.

If a minion ever leaves play in any way, its stats are restored to their starting values.

Note: *Minions with 0 attack can still attack.*

Example: *Brad plays Air Elemental (a 1/1) and Nomair then attacks it with his Earth Elemental (a 4/4), killing the Air Elemental and reducing the Earth Elemental's defence by 1. The Air Elemental in Brad's graveyard is a 1/1, and Nomair's Earth Elementai s a 4/3. Brad then plays Unsummon to return Nomair's Earth Elemental to Nomair's deck. Nomair's Earth Elemental is now a 4/4*

In addition to attack, defence, and actions, some minions also have abilities. Abilities are divided into two categories:

- *Activated abilities* cost magic and an action to use, and work similar to playing a spell card.
- *Triggered abilities* activate for free whenever a certain condition is met.

Due to space concerns in the text interface a minion can only have one ability.

Question: How could you design activated abilities in your code to maximize code reuse?

2.3 Enchantments

Enchantments are modifications that can be played on minions. An enchantment can modify any aspect of a minion: some possibilities include modifying attack and defence values or granting new abilities.

If a minion is enchanted by multiple enchantments, they are applied in oldest-to-newest order. For example, a 2/2 minion enchanted first with a $+1/+1$ enchantment and then a $*2/*2$ enchantment has 6 attack and defence, while if it was first enchanted with the $*2/*2$ enchantment it would have 5 attack and defence.

If an enchantment grants a minion a new activated ability, the minion's old activated ability may not be used. If multiple enchantments grant a minion activated abilities, only the newest enchantment's ability may be used.

If at any time a minion leaves the board (for example, it dies or is returned to its owner's deck), all enchantments on that minion are removed from the game.

Example: *Sean plays Apprentice Summoner (a 1/1) and then plays Giant Strength (an enchantment which gives the Apprentice Summoner +2/+2) on it. Ten plays Unsummon on Sean's Apprentice Summoner, returning it to Sean's deck. The Giant Strength enchantment is removed from the game, and next time Sean draws and plays his Apprentice Summoner it will be a 1/1.*

Question: What design pattern would be ideal for implementing enchantments? Why?

2.4 Rituals

Rituals are special cards with a *triggered ability*, an *activation cost* and a number of *charges*. Every time the ritual's triggered ability activates, it expends a number of charges equal to its activation cost to do the effect. If it does not have enough charges left to activate, the ability's effect simply does not occur, but the ritual remains on the board.

A player may only have one ritual on the board at any one time. If they play a second ritual while one is already active, the old ritual is removed from the game.

Example: *Brad has a Dark Ritual in play (whose triggered ability grants him a magic at the start of his turn) with an activation cost of 1 and 2 charges left, while Nomair has a Dark Ritual in play with an activation cost of 1 and 0 charges left. On Brad's next turn he gains an extra magic from his Dark Ritual, reducing its charges to 1. On Nomair's next turn, his Dark Ritual does not grant him an extra magic and remains at 0 charges.*

3 Triggers

There are four triggers which can be used by triggered abilities or other game rules. Triggers only activate on cards that are currently on the board. If multiple triggers activate, they activate in APNAP order. If a minion is already dead, triggers targeting that minion (such as Fire Elemental or Standstill) do not occur.

3.1 At the start of your turn

When a player's turn starts, all triggered abilities titled "at the start of your turn" (or similar) on cards on that player's board activate. These occur immediately after the player gains magic and draws a card (if applicable), and before the board is displayed for that turn.

3.2 At the end of your turn

When a player's turn ends, all triggered abilities titled "at the end of your turn" (or similar) on cards on that player's board activate.

3.3 Whenever a minion enters play

When a minion is placed on the board by any means (for example, played from a player's hand or created by a spell) all cards on the board with this trigger activate in APNAP order, including the minion entering play.

3.4 Your choice

The last trigger is unspecified: you need to design one more of your own as part of a custom minion. It can be anything other than the above three, and you get to pick the rules about how edge cases are resolved.

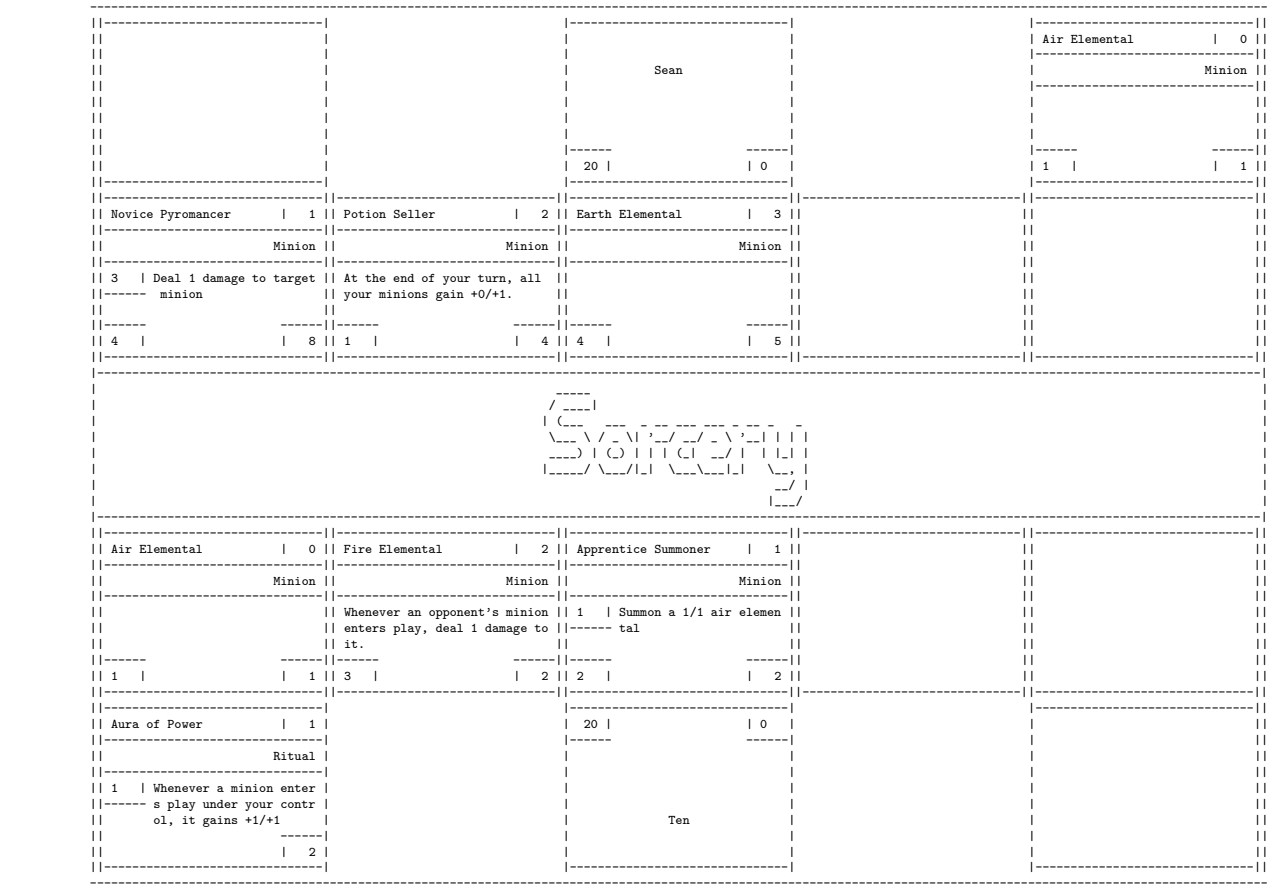
Example: *Sean has both a Fire Elemental and a Standstill in play, and Nomair plays an Air Elemental. Due to APNAP order, the Fire Elemental's trigger activates first, killing the Air Elemental. Since the Air Elemental is no longer in play, Standstill doesn't activate and doesn't have to spend charges.*

4 The Display

All information in Sorcery is displayed in a text-based manner.

4.1 The Board

An example Sorcery board follows:

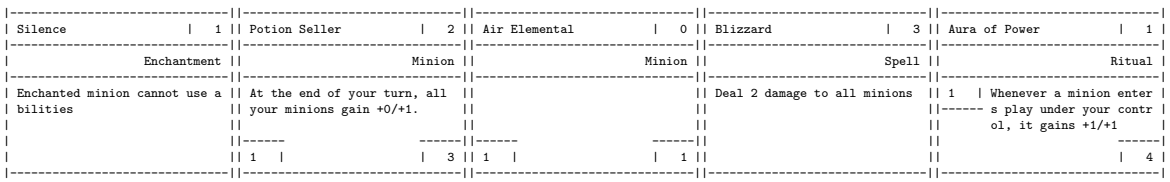


Note: *You are given two options for drawing the board: the simple style in this document or a fancier style using unicode characters. Both styles are available in the files* `ascii_graphics.cc/h` *provided to you. You are encouraged to use the fancy style, but the simple one is provided in case the characters in the fancy style do not render properly on your terminal.*

The top and bottom left cards are player 1 and 2's rituals respectively, while the top and bottom right cards are their graveyards. The middle-top row of 5 cards are player 1's minions, while the middle-bottom row are player 2's minions. Any empty slots are filled with a blank rectangle. The top and bottom centre boxes represent the players themselves: the left value is the player's life, the right value is their magic, and the centre value is their name. Minions are left-justified: if a minion dies, minions to the right of it should move left to fill the hole. The graveyard displays the top minion (the minion which entered it most recently) if it is not empty.

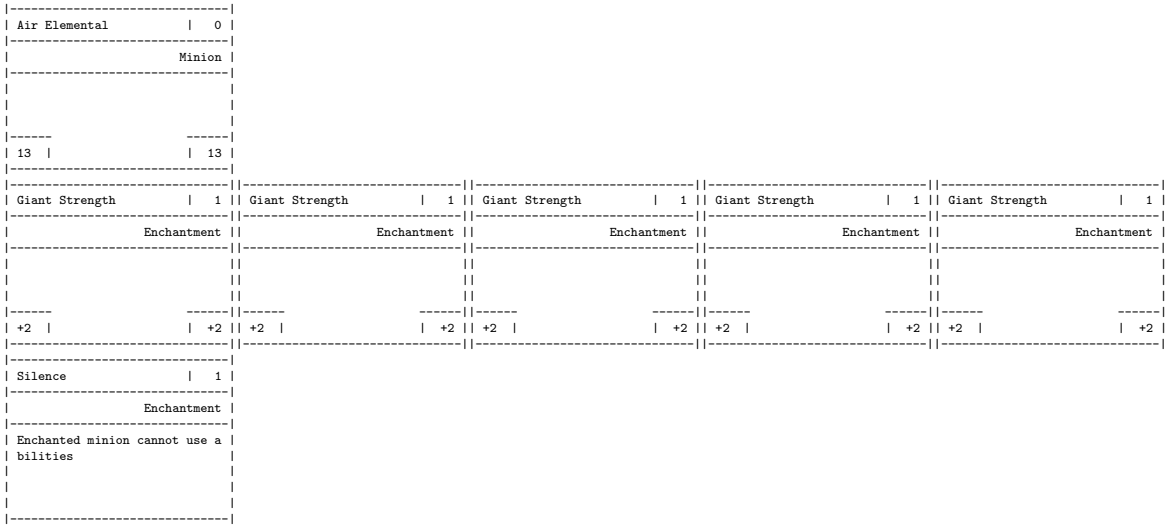
4.2 The Hand

To display a player's hand, simply display the cards within it in a row. For example:



4.3 Inspecting a minion

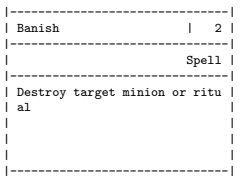
To inspect a minion, display the minion (in the exact same way it appears on the board), and then on a new line display its enchantments, oldest to newest, five per line. For example, if Silence was the newest enchantment played:



4.4 Individual cards

Whenever a card needs to be displayed in any of the previous areas, it should be displayed as follows:

4.4.1 Spells



The card's name is "Banish", indicated in the top left, and its cost is 2, indicated in the top right. Its type is listed in the middle box and a description of its effect is in the lower box

4.4.2 Minions

Minions can be displayed in one of two ways, depending on whether they have an activated ability or not.

A minion without an activated ability is displayed as follows:

```
|-------------------------------|
| Potion Seller           |   2 |
|-------------------------------|
|                        Minion |
|-------------------------------|
| At the end of your turn, all  |
| your minions gain +0/+1.      |
|                               |
|------                  ------|
| 1   |                  |   3 |
|-------------------------------|
```

In this case, the Potion Seller has a triggered ability, displayed in its effect box. The only difference from the layout of the spell card above is the presence of an attack box in the lower left and defence box in the lower right. If the minion has no triggered ability, the effect box will simply be empty.

If the minion has an activated ability, there is an additional box to describe its cost:

```
|-------------------------------|
| Apprentice Summoner     |   1 |
|-------------------------------|
|                        Minion |
|-------------------------------|
| 1   | Summon a 1/1 air elemen |
|------ tal                     |
|                               |
|------                  ------|
| 1   |                  |   1 |
|-------------------------------|
```

Question: Suppose we found a solution to the space limitations of the current user interface and wanted to allow minions to have any number and combination of activated and triggered abilities. How might you design your code and the user interface to accomodate this?

4.4.3 Enchantments

Enchantments are laid out similar to minions: the parts of the minion that they modify are highlighted by their layout. For example, the enchantment on the left adds 2 to a minion's attack and defence and the enchantment on the right prevents it from using abilities.

```
|-------------------------------||-------------------------------|
| Giant Strength          |   1 || Silence                |   1 |
|-------------------------------||-------------------------------|
|                   Enchantment ||                   Enchantment |
|-------------------------------||-------------------------------|
|                               || Enchanted minion cannot use a |
|                               || bilities                      |
|                               ||                               |
|------                  ------||                               |
| +2  |                  |  +2 ||                               |
|-------------------------------||-------------------------------|
```

4.4.4 Rituals

Rituals are laid out similarly to minions with abilities, except without an attack box. The ritual's activation cost is located where the ability cost would be located for a minion, while its number of remaining charges is located where the minion's defence would be.

```
|-------------------------------|
| Dark Ritual             |   0 |
|-------------------------------|
|                         Ritual |
|-------------------------------|
| 1   | At the start of your tu |
|------ rn, gain 1 magic        |
|                               |
|                         ------|
|                         |   5 |
|-------------------------------|
```

5 Commands and Command Line Arguments

5.1 Commands

During a player's turn they may issue the following commands. Each command must be issued on its own line. A command may have extra whitespace before and after every word in it.

5.1.1 help

The `help` command displays a message describing the commands and their formats. You may use the following help message or create your own:

```
Commands: help -- Display this message.
          end  -- End the current player's turn.
          quit -- End the game.
          attack minion other-minion -- Orders minion to attack other-minion.
          attack minion -- Orders minion to attack the opponent.
          play card [target-player target-card] -- Play card, optionally targeting target-card owned by target-player.
          use minion [target-player target-card] -- Use minion's special ability, optionally targeting target-card owned by target-player.
          inspect minion -- View a minion's card and all enchantments on that minion.
          hand -- Describe all cards in your hand.
          board -- Describe all cards on the board.
```

5.1.2 end

The `end` command ends the current player's turn. A player may end their turn at any time.

5.1.3 quit

The `quit` command ends the game immediately with no winner. Ctrl+D has the same effect.

5.1.4 draw

The `draw` command draws a card, similar to the effect if the player just started their turn. This command is only available in `-testing` mode.

5.1.5 discard

The `discard i` command discards the `i`th card in the player's hand, simply removing it from their hand (the card does not go to the graveyard, trigger leave play effects or anything else). This command is only available in `-testing` mode.

5.1.6 attack

The `attack` command follows one of two formats:

- `attack i` orders minion `i` to attack the opposing player, where 1 is the leftmost minion and 5 is the rightmost minion.
- `attack i j` orders the active player's minion `i` to attack the inactive player's minion `j`, where both `i` and `j` are as above.

5.1.7 play

The `play` command follows one of two formats:

- `play i` plays the `i`th card in the active player's hand with no target. For example, this can be used to play minions, rituals, and spells with no targets. Once again `i` ranges from `1` to `5`.

- `play i p t` plays the `i`th card in the active player's hand on card `t` owned by player `p`. `p` may be equal to `1` or `2` to represent player `1` or `2` respectively. `t` is either `1, 2, 3, 4, 5` (the `i`th minion owned by player `p`) or `r` (the ritual owned by player `p`). This can be used to play enchantments and spells with targets.

5.1.8 use

The `use` command follows the same format as the `play` command and has the same meaning, except that `i` refers to the `i`th minion owned by the current player, and the command orders that minion to use its activated ability on the provided target (or on no target).

5.1.9 inspect

The `inspect i` command inspects the `i`th minion owned by the active player, as described in the "inspecting a minion" subsection of the Display section.

5.1.10 hand

The `hand` command displays the active player's hand, as described in the "hand" subsection of the Display section.

5.1.11 board

The `board` command displays the board, as described in the "board" subsection of the Display section.

5.2 Command Line Arguments

The following command line arguments may be specified to Sorcery in any order:

5.2.1 -deck1 and -deck2

The `-deck1` *filename* argument specifies that player 1's deck will be supplied in *filename*. `-deck2` works similarly but for player 2. If either player's deck is not specified using one of these commands, that player should use the file `default.deck` to specify their deck, which can be assumed to exist in the current directory. Deck files are simply a list of card names, one per line: see the provided `default.deck` for an example.

Note that the first card in the deck is the first card that will be drawn. So, when using `default.deck` and `-testing` mode, each player should start with the cards on the first five lines, and the first card they draw should be the card on the 6th line. There is no limit on deck size: a deck consists of precisely the cards listed in the file.

5.2.2 -init

The `-init` *filename* arguments specifies that the game will be initialized using *filename*. Filename consists of a sequence of commands to read from standard input before prompting the user for additional input (this includes player names). For example, if *filename* contains:

```
Sean
Ten
play 1
play 1
```

Then the game will begin with Player 1 named Sean, Player 2 named Ten, and Sean attempting to play the first two cards from his hand. After this, play continues using input from standard input as normal.

5.2.3 -testing

The `-testing` argument enables testing mode, changing gameplay in four ways:

- If a player attempts to play a card or activate an ability and does not have enough magic to do so, their magic is simply set to `0` and they play the card or activate the ability as if they had enough magic.
- Players may now use the `discard i` command to discard the `i`th card in their hand.
- Players may now use the `draw` command to draw a card.
- Decks are no longer randomized at the beginning of the game.

These two commands do not need to be described in the `help` command.

Question: Suppose you chose to implement graphics as a bonus feature. How could you make supporting two (or more) interfaces at once easy while requiring minimal changes to the rest of the code?

6 Individual Card Descriptions

By default, Sorcery includes the cards listed in this section. Once you are done the main game, you're welcome to add more cards out of interest or as a bonus.

6.1 Minions

Air Elemental	0	Minion		Earth Elemental	3	Minion		Fire Elemental	2	Minion		Potion Seller	2	Minion
								Whenever an opponent's minion enters play, deal 1 damage to it.				At the end of your turn, all your minions gain +0/+1.		
1	1			4	4			2	2			1	3	

Novice Pyromancer	1	Minion	Deal 1 damage to target minion	Apprentice Summoner	1	Minion	Summon a 1/1 air elemental	Master Summoner	3	Minion	Summon up to three 1/1 air elementals
1	1		1	1			2	3			

Notes:

- The Apprentice and Master Summoner abilities cannot be used if their owner already has 5 minions on the board.
- Master Summoner's ability may be used if there is room for at least one more minion but not all three. In that case, it simply summons enough to fill the board.
- Novice Pyromancer and Fire Elemental do not take damage from the minions they damage with their abilities.

6.2 Spells

Banish	2	Unsummon	1	Recharge	1
Spell		Spell		Spell	
Destroy target minion or ritual		Put target minion on the bottom of its owner's deck		Your ritual gains 3 charges	
Disenchant	1	Raise Dead	1	Blizzard	3
Spell		Spell		Spell	
Destroy the top enchantment on target minion		Resurrect the top minion in your graveyard		Deal 2 damage to all minions	

Notes:

- Recharge and Raise Dead cannot be played if the ritual slot or graveyard respectively are empty. Raise Dead cannot be played if there is no room for new minions.
- "Resurrect" means move from the graveyard to the board.
- Disenchant cannot be played on a minion with no enchantments.

6.3 Enchantments

Giant Strength	1	Magic Fatigue	0	Silence	1
Enchantment		Enchantment		Enchantment	
		Enchanted minion's activated ability costs 2 more		Enchanted minion cannot use abilities	
+2	+2				

Notes:

- Silence and Magic Fatigue can be played on minions with no activated ability, in which case they do nothing but remain on the minion as an enchantment.

6.4 Rituals

Dark Ritual	0	Aura of Power	1	Standstill	3
Ritual		Ritual		Ritual	
1	At the start of your turn, gain 1 magic	1	Whenever a minion enters play under your control, it gains +1/+1	2	Whenever a minion enters play, destroy it
	5		4		4

Notes:

- Standstill affects your own minions.
- Due to APNAP order, if you play a 1/1 (such as Air Elemental) with Aura of Power in play while your opponent has Fire Elemental in play, your 1/1 will gain the +1/+1 first, and therefore survive as a 2/1 once it takes damage from Fire Elemental.

6.5 Custom cards

In addition to the cards listed above, you are expected to implement four more cards with a reasonable amount of flexibility.

- There must be one card of each type (minion, spell, enchantment, ritual).
- The enchantment must add a new activated ability to the minion it enchants.
- Either the minion or the ritual must introduce a fourth kind of trigger.
- The remaining two new cards must introduce a new nontrivial mechanic (of similar difficulty to the previous two requirements). The new mechanic should require some interesting application of what you've learned in this course. The new rules may work however you wish so long as they do not change the behaviour of the rest of the game.

Question: Describe your additions to the game. How do you expect them to effect your organization of Sorcery's code compared to if they were not there? Which object-oriented programming concepts were used in your solution?

7 Advice

Sorcery is a serious project which will take both time and some clever software engineering to complete. Since we can only assign marks to working components of your program, this section contains advice on how to go about tackling the project. Finally, this section has some advice for bonus marks.

7.1 Tackling the project

This section provides some advice on the order in which you might want to approach Sorcery to maximize the number of marks you get for the work you've completed:

Watch out: To test most of the functionality of Sorcery we will require the `-testing` and `-init` arguments to work properly. Make sure the features described by those arguments are available!

1. Decide on the basic classes you will use in your program, and their high-level relationships.
2. Implement players (only having names for now), the game loop, and the `-init` command line argument. Make each command simply echo itself for now, so that you can verify that all of these work correctly.
3. Implement skeleton functionality to load decks from a file called `default.deck`. If you cannot get deck loading from a file working, start by instead hardcoding the players' decks to start with the provided `default.deck` (make sure the cards are listed in the same order as they are in that file!). We will use that file for all of our provided tests, except those relying on deck loading. If you get the custom cards implemented but do not manage to get deck loading working, make sure you add them to the **end** of the deck so that they do not interfere with our tests.
4. Implement abstract cards and the ability for a player to have a hand of cards, including giving each player a deck and the functionality to draw from that deck. Implement the ability for players to start and end their turn, including drawing a card at the start of their turn if their hand isn't full.
5. Implement minions with no activated or triggered abilities, and allow them to attack players (with no limit on the number of actions per turn). Keep in mind that they will need to be enchantable later.
6. Implement spells which interact with minions.
7. Allow minions to attack other minions.

8. Implement rituals and triggered abilities.
9. Implement simple enchantments, such as enchantments that modify the attack of a minion.
10. Implement activated abilities.
11. Implement details that have been left out thus far (magic, actions, etc).
12. Implement the more complicated remaining cards.

7.2 Bonus content

If you have extra time and want to earn some bonus marks, there are plenty of things you can do to improve Sorcery. Here are a few ideas to get you started.

Watch out: While working on bonus content can be fun, it's generally much more time consuming and worth far fewer marks than the main project is. Make sure you don't start the bonus until your base game is complete!

1. Use the XWindow class we provided, or a proper graphics library, to add graphics to your project.
2. Add new cards with new and interesting effects, similar to how you designed four new custom cards.
3. Play some collectible card games and implement some other ideas from them!

Bonus content which earns high marks should involve difficult extensions or exhibit interesting object-oriented design.

7.3 Polish

For most of the marks we are more concerned with checking that you have implemented the features specified in this document, and an otherwise fairly barebones implementation will do. However, projects earning high marks should provide a polished user interface: mistaken should provide feedback in the form of error messages and the user interface should be as easy to use as the medium allows (for example, it should be somehow clear whose turn it is at any given time).

8 Provided Files

You are given `default.deck` as well as `ascii_graphics.cc` and `ascii_graphics.h`. The `ascii_graphics` library provides a number of utility functions and variable definitions to help you draw an interface which looks like the one in this document without too much work. It doesn't do anything on its own: it's designed to be called from your code.

`default.deck` provides all of the cards that you are required to implement, except for the four which you will design yourself. If you modify it to include your custom cards, make sure to add them to the end to not break any tests during the demo!

9 Preparing for the Demo

You are free to run the demo however you'd like: the TAs will have a checklist that they'll use as you go, and anything you haven't covered by the end they'll guide you through. We'll provide a number of test files (designed to be used with `-init` and `default.deck`) to streamline the process if you'd like, but you'll need to decide how to properly demonstrate the custom cards you added.

10 Submission Instructions

See **project_guidelines.pdf** for important instructions including what to submit when, bonus, plan of attack and final design document.

11 A Note on Random Generation

To complete the pseudo-random number generation portion of this project you have two options available to you. The rand and srand functions in <cstdlib> generate a random number and seed the random generator respectively (typically, time() from <ctime> is used for the seed). Alternatively, you can use the prng class from prng.h, which provides an encapsulated random number generation algorithm. Either is fine for this project.