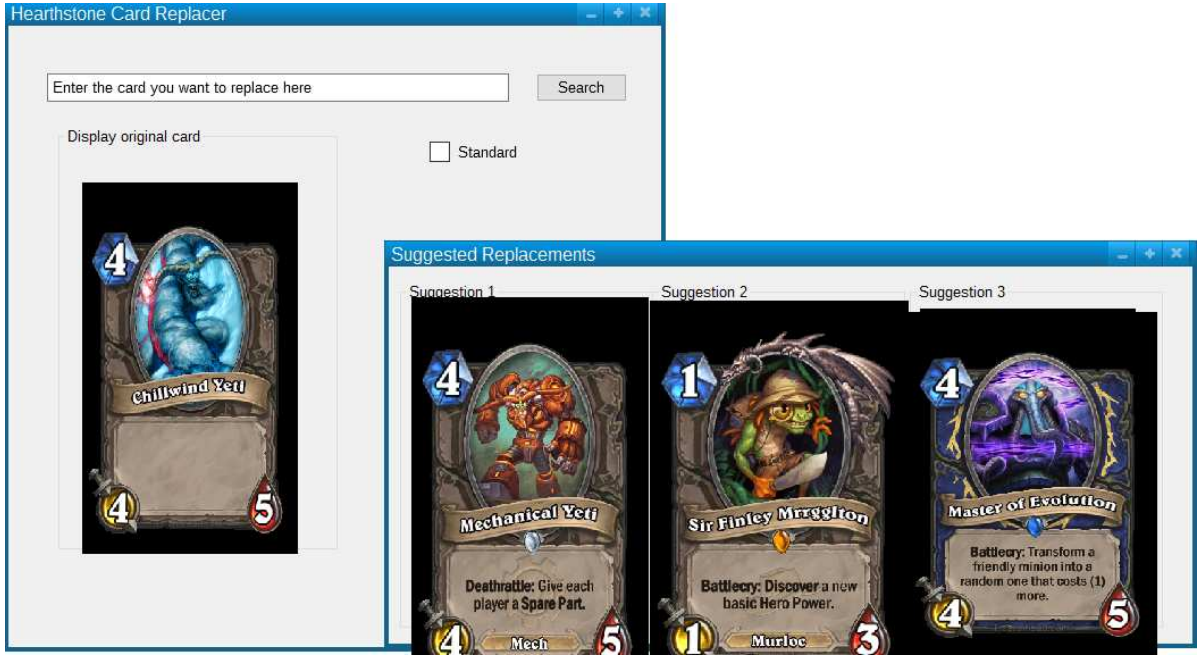
Specification

The purpose of this program is to find replacement Hearthstone cards. This is done by taking the card name the user inputs and entering it into the Hearthstone API (market.mashape.com/omgvamp/hearthstone) to find the attributes (e.g. mana cost, attack, health, class) of the card. The program then stores these attributes and finds replacements by comparing them with cards of the same type. If too few cards are found, the program broadens the scope of the search. After the search is complete, the program displays the images of the 3 best replacements cards.

Analysis I



This is a mock up of our GUI design. The user will input the name of a card and press the button to call our event handler which will look up the input card, run our comparison algorithm, and return a new window frame showing the three best matches.

Analysis II

To look up the cards, we will use OMGVAMP's Hearthstone API hosted at market.mashape.com/omgvamp/hearthstone. We will use the CURL library to retrieve card data from the API and write it into a file, then use Niels Lohmann's JSON library to parse the file into data structures we can use in our comparisons. The GUI will be made using FLTK's Cairo library.

Design I

- compareMinions
  - If user's card is a minion, retrieve a list of legal minions
  - Iterate through the list comparing each card to the user's to determine fitness
  - If the cost of both cards is equal, the fit is increased by 3
  - Otherwise, fit is reduced by 3 times the difference
  - If the health of both cards is equal, the fit is increased by 3
  - Otherwise, fit is reduced by 2 times the difference
  - If the attack of both cards is equal, the fit is increased by 3
  - Otherwise, fit is reduced by 3 times the difference

Design II

- compareSpell
  - If user's card is a spell, retrieve a list of legal spells
  - Iterate through the list, comparing each card's text to determine fitness
  - Tokenize the text of the card into individual words and compare the words to the user's card's text
  - If a word in the card's text matches one in the user's, fit is increased by 3
  - If a matching word is a mechanic keyword, fit is increased by an additional 5
  - Fit is reduced by the square of the difference in cost

Design III

- compareWeapons
  - If user's card is a minion, retrieve a list of legal minions
  - Iterate through the list comparing each card to the user's to determine fitness
  - If the cost of both cards is equal, the fit is increased by 3
  - Or if the user's card costs more, fit is reduced by the difference squared
  - Or if the user's card costs less, fit is reduced by 2 times the square of the difference
  - If the durability of the user's weapon is lower, the fit is increased by 2 times the difference
  - Otherwise, fit is reduced by 3 times the difference
  - If the attack of user's card is lower, the fit is increased by the difference
  - Otherwise, fit is reduced by 2 times the difference

Implementation I

Main
Goal: Start the GUI and pass control to FLTK

```
#include "lab.h"

int main()
{
    makeSearchWindow()->show();
    Fl::run();
}
```

Implementation II

getCard

Goal: Retrieve data for a single card from the Hearthstone API

- Access API using Mashape key
- Use JSON library to parse data
- Store card data in a struct

```
#include "lab.h"
const string js = "Accept: application/json";
const string key =
"X-Mashape-Key: q9bHltcHV2mshsWdmYro91TZDWyCp1vFXk8jsnHs80kSDWR3Co";

Card getCard(string query)
{
    Card card;
    string output;
```

Implementation III

```
struct curl_slist *slist1 = NULL;
slist1 = curl_slist_append(slist1, js.c_str());
slist1 = curl_slist_append(slist1, key.c_str());

string cmd = "https://omgvamp-hearthstone-v1.p.mashape.com/cards/sear

while (query.find(" ")!=string::npos)
    query.replace(query.find(" "), 1, "%20");

string search = cmd + query + "?collectible=1";

CURL* handle = curl_easy_init();
curl_easy_setopt(handle,CURLOPT_URL,search.c_str());
curl_easy_setopt(handle,CURLOPT_HTTPHEADER, slist1);
curl_easy_setopt(handle,CURLOPT_WRITEFUNCTION,handleData);
curl_easy_setopt(handle,CURLOPT_WRITEDATA,&output);
```

Implementation IV

```
curl_easy_perform(handle);

while (!(output.back() == '}'||output.back() == ']') ) output.pop_back();
//cout << output << endl;

json ji = json::parse(output);
if (output.find("Card not found") != string::npos){
    cout << "error card not found" << endl;
    card.name = "error";
}
else
{
    if (!ji[0]["name"].is_null()) card.name = ji[0]["name"];
    if (!ji[0]["cardSet"].is_null()) card.cardSet = ji[0]["cardSet"];
    if (!ji[0]["type"].is_null()) card.type = ji[0]["type"];
    if (!ji[0]["text"].is_null()) card.text = ji[0]["text"];
    if (!ji[0]["playerClass"].is_null())
```

Implementation V

```
            card.playerClass = ji[0]["playerClass"];
        if (!ji[0]["race"].is_null()) card.race = ji[0]["race"];
        if (!ji[0]["img"].is_null()) card.img = ji[0]["img"];
        if (!ji[0]["health"].is_null()) card.health = ji[0]["health"];
        if (!ji[0]["attack"].is_null()) card.attack = ji[0]["attack"];
        if (!ji[0]["durability"].is_null())
            card.durability = ji[0]["durability"];
        if (!ji[0]["cost"].is_null()) card.cost = ji[0]["cost"];
    }
    return card;
}
```

Implementation VI

getData
Goal: Retrieve a set of card data to compare against the search card

- Access the Hearthstone API's type endpoint for the type matching the search card
- Use snippets from CURL documentation to write retrieved data into a file
- Use JSON library to parse file into a list of card structs

Implementation VII

```
#include "lab.h"
const string js = "Accept: application/json";
const string key =
"X-Mashape-Key: q9bHltcHV2mshsWdmYro91TZDWyCp1vFXk8jsnHs80kSDWR3Co";

size_t handleData(void* ptr, size_t size, size_t nmemb, void* g)
{
    *(static_cast<string*>(g)) += (static_cast<char*>(ptr));
    return size * nmemb;
}

size_t write_data(void *ptr, size_t size, size_t nmemb, FILE *stream)
{
    size_t written = fwrite(ptr, size, nmemb, stream);
    return written;
}
```

Implementation VIII

```
list<Card> getData(string query, string endpoint)
{
    list<Card> cardlist;

    string search = "https://omgvamp-hearthstone-v1.p.mashape.com/cards/";
    search = search + endpoint + "/";
    while (query.find(" ")!=string::npos)
        query.replace(query.find(" "), 1, "%20");
    search = search + query + "?collectible=1";

    struct curl_slist *slist1 = NULL;
    slist1 = curl_slist_append(slist1, js.c_str());
    slist1 = curl_slist_append(slist1, key.c_str());

    FILE *carddata;
    CURL* curl = curl_easy_init();
    if (curl) {
```

Implementation IX

```
    carddata = fopen("carddb.txt","wb");
    curl_easy_setopt(curl, CURLOPT_URL, search.c_str());
    curl_easy_setopt(curl,CURLOPT_HTTPHEADER, slist1);
    curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, write_data);
    curl_easy_setopt(curl, CURLOPT_WRITEDATA, carddata);
    curl_easy_perform(curl);
    curl_easy_cleanup(curl);
    fclose(carddata);
}

ifstream ifs ("carddb.txt");
json ji = json::parse(ifs);

for (int i = 0; i < ji.size(); i++)
{
    Card card;
    if (!ji[i]["name"].is_null()) card.name = ji[i]["name"];
```

Implementation X

```
    if (!ji[i]["cardSet"].is_null()) card.cardSet = ji[i]["cardSet"];
    if (!ji[i]["type"].is_null()) card.type = ji[i]["type"];
    if (!ji[i]["text"].is_null()) card.text = ji[i]["text"];
    if (!ji[i]["playerClass"].is_null())
        card.playerClass = ji[i]["playerClass"];
    if (!ji[i]["race"].is_null()) card.race = ji[i]["race"];
    if (!ji[i]["img"].is_null()) card.img = ji[i]["img"];
    if (!ji[i]["health"].is_null()) card.health = ji[i]["health"];
    if (!ji[i]["attack"].is_null()) card.attack = ji[i]["attack"];
    if (!ji[i]["durability"].is_null())
        card.durability = ji[i]["durability"];
    if (!ji[i]["cost"].is_null()) card.cost = ji[i]["cost"];
    card.fit = 0;
    cardlist.push_back(card);
}
```

Implementation XI

```
        return cardlist;
    }
```

Implementation XII

compareMinions

Goal: Assign scores to possible replacement minions

- Iterate through the list of minions provided by getData
- Compare the cost, health, and attack of the search card to each minion in the list
- Return a list of legal replacements with similarity scores calculated

```
#include "lab.h"

list<Card> compareMinions(Card theirCard)
{
    list<Card> cardSet = getData("Minion", "types");
    list<Card> fitList;
```

Implementation XIII

```
for (list<Card>::iterator it=cardSet.begin(); it != cardSet.end(); it++)
{
    if ( ( (*it).playerClass == theirCard.playerClass ||
            (*it).playerClass == "Neutral" )
        && (*it).name != theirCard.name)
        {
        int x = 0;

        if ((*it).cost - theirCard.cost > 0)
            x -= 3*((*it).cost - theirCard.cost);
        else if ((*it).cost - theirCard.cost < 0)
            x += 3*((*it).cost - theirCard.cost);
        else
            x +=3;

        if ((*it).attack - theirCard.attack > 0)
            x -= (*it).attack - theirCard.attack;
```

Implementation XIV

```
            else if ((*it).attack − theirCard.attack < 0)
                x += (*it).attack − theirCard.attack;
            else
                x +=3;

            if ((*it).health − theirCard.health > 0)
                x −= 2*((*it).health − theirCard.health);
            else if ((*it).health − theirCard.health < 0)
                x += 2*((*it).health − theirCard.health);
            else
                x +=3;

            (*it).fit = x;
            fitList.push_front((*it));
        }
    }
        //for(list<Card>::iterator it=fitList.begin();it!=fitList.end();++it);
```

Implementation XV

```
        //cout<<(*it).name<< " " << (*it).fit << endl;
    return fitList;
}
```

Implementation XVI

compareSpell
Goal: Assign scores to possible replacement spells

- Iterate through the list of spells provided by getData
- Turn the text value of the card into tokens
- Compare the text of each spell in the list to the search card by matching tokens
- Return a list of legal replacements with similarity scores calculated

Implementation XVII

```
#include "lab.h"

list<Card> compareSpell(Card inputCard)
{
    list<Card> spells = getData("Spell", "types");
    list<Card> outList;
    for (list<Card>::iterator it=spells.begin(); it != spells.end(); ++it)
    {
        if ((*it).playerClass == inputCard.playerClass
            && (*it).name != inputCard.name){
            string text = inputCard.text;
            text.erase(remove(text.begin(), text.end(), ','), text.end());
            text.erase(remove(text.begin(), text.end(), '.'), text.end());
            list<string> words = tokenize(text);
    //cout << text << "...";
    //cout << words.size() << endl;
```

Implementation XVIII

```
            for(list<string>::iterator wit=words.begin();wit!=words.end();wit++)
                {
                    if ((*it).text.find(*wit) != string::npos) {
                        (*it).fit += 3;
                        if( (*wit).find("<b>") != string::npos) (*it).fit += 5;
                    }//cout << *it << " " << fit << endl;
                }
            (*it).fit -= pow((inputCard.cost - (*it).cost), 2);
            outList.push_back(*it);
        }
    }
    return outList;
}

list<string> tokenize(string str)
{
    list<string> words;
```

Implementation XIX

```
    istringstream iss(str);
    while (iss)
    {
        string substring;
        iss >> substring;
        words.push_back(substring);
    }
    return words;
}
```

Implementation XX

compareWeapons

Goal: Assign scores to possible replacement weapons

- Iterate through the list of weapons provided by getData
- Compare the cost, durability, and attack of the search card to each weapon in the list
- Return a list of legal replacements with similarity scores calculated

```
#include "lab.h"

list<Card> compareWeapons(Card theirCard)
{
    list<Card> data = getData("Weapon", "types");
    list<Card> fitList;
    for (list<Card>::iterator it=data.begin(); it != data.end(); it++)
```

Implementation XXI

```
{
    //cout << (*it).name << " " << (*it).playerClass << endl;
    if ( (*it).playerClass == theirCard.playerClass
        && (*it).name != theirCard.name)
    {
        if ((*it).cost - theirCard.cost > 0)
            (*it).fit -= 2*pow((*it).cost - theirCard.cost, 2);
        else if ((*it).cost - theirCard.cost < 0)
            (*it).fit -= pow((*it).cost - theirCard.cost, 2);
        else
            (*it).fit +=3;

        if ((*it).attack - theirCard.attack > 0)
            (*it).fit += (*it).attack - theirCard.attack;
        else
            (*it).fit += 2 * ((*it).attack - theirCard.attack);
```

Implementation XXII

```
            if ((*it).durability − theirCard.durability > 0)
                (*it).fit += 2 * ((*it).durability − theirCard.durability);
            else
                (*it).fit += 3 * ((*it).durability − theirCard.durability);

            fitList.push_back((*it));
        }
        }for(list<Card>::iterator it=fitList.begin();it!=fitList.end();++it)
        //cout << (*it).name << " " << (*it).playerClass <<" "<< (*it).fit <
    return fitList;
}
```

Implementation XXIII

cullCards

Goal: Find the three most similar cards to the search card

- Iterate through the list of cards provided by the compare functions
- Compare the similarity scores against each other, keeping the three highest
- Return a list of the three cards with the highest similarity score

```
#include "lab.h"

list<Card> cullCards(list<Card> cardSet)
{
Card acard = getCard("murloc tinyfin");
Card bcard, ccard;
acard.fit = -1000;
```

Implementation XXIV

```
bcard.fit = −1000;
ccard.fit = −1000;
list<Card> topThree;


for (list<Card>::iterator it = cardSet.begin(); it != cardSet.end(); it++){
//cout << (*it).name << " " << (*it).fit <<endl;
    if((*it).fit > acard.fit){
        ccard = bcard;
        bcard = acard;
        acard = *it;
    }
    else if((*it).fit > bcard.fit){
        ccard = bcard;
        bcard = *it;
    }
    else if((*it).fit > ccard.fit)
```

Implementation XXV

```
        ccard = *it;
}

topThree.push_back(acard);
topThree.push_back(bcard);
topThree.push_back(ccard);
return topThree;
}
```

Implementation XXVI

GUIWindow

Goal: Create a GUI for the user to enter a search query to find replacement cards

- Create a Cairo window using FLTK
- Create a user input field and a field to hold the image
- Send user input to cbSearch function when user presses button

```
#include "lab.h"

Fl_Input * sq;
Fl_Cairo_Window * gw;
Fl_Box * cw;
Fl_Button * sr;
void cbSearch(Fl_Button*,void*);
```

Implementation XXVII

```
Fl_Cairo_Window * makeSearchWindow()
{
    int width = 310;
    int height = 500;

    gw = new Fl_Cairo_Window (width,height);
    gw->color(FL_WHITE);
    gw->label("Card Replacement Searcher");

    cw = new Fl_Box(FL_FLAT_BOX,0,85, width,415,"");
    cw->color(FL_WHITE);

    sq = new Fl_Input(.025* width, .055*height, .55*width, .05*height, "Card");
    sq->align(FL_ALIGN_TOP);

    sr = new Fl_Button(.65*width, .075 *height, .3*width, .07*height, "Search")
    sr->callback((Fl_Callback*)cbSearch);
```

Implementation XXVIII

```
        return gw;
    }
```

Implementation XXIX

makeDisplayWindow
Goal: Create a window to display the top three replacement cards

- Create a Cairo window using FLTK
- Create three boxes to hold card images
- Call compare function matching type of user's card
- Call cullCards to cull comparisons to three best fits
- Retrieve card images from list returned by cullCards and display in order of fit

Implementation XXX

```
#include "lab.h"

Fl_Cairo_Window * dw;
Fl_Box * b1;
Fl_Box * b2;
Fl_Box * b3;
extern Fl_Input * sq;

Fl_Cairo_Window * makeDisplayWindow(Card inputCard)
{
    list<Card> cardlist;
    list<Card> topThree;

    dw = new Fl_Cairo_Window (921,415);
    dw->color(FL_WHITE);
    dw->label("Recommended Cards");
```

Implementation XXXI

```
b1 = new Fl_Box(FL_FLAT_BOX,0,0, 307,415,"");
b1->color(FL_WHITE);

b2 = new Fl_Box(FL_FLAT_BOX,307,0, 307,415,"");
b2->color(FL_WHITE);

b3 = new Fl_Box(FL_FLAT_BOX,614,0, 307,415,"");
b3->color(FL_WHITE);

if(inputCard.type == "Minion")
      cardlist = compareMinions(inputCard);
else if(inputCard.type == "Spell")
    cardlist = compareSpell(inputCard);
else if(inputCard.type == "Weapon")
    cardlist = compareWeapons(inputCard);

topThree = cullCards(cardlist);
```

Implementation XXXII

```
Card one = topThree.front();
topThree.pop_front();
Card two = topThree.front();
topThree.pop_front();
Card three = topThree.front();

FILE *img1;
CURL* curl = curl_easy_init();
if (curl) {
    img1 = fopen("one.png","wb");
    curl_easy_setopt(curl, CURLOPT_URL, one.img.c_str());
    curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, write_data);
    curl_easy_setopt(curl, CURLOPT_WRITEDATA, img1);
    curl_easy_perform(curl);
    curl_easy_cleanup(curl);
    fclose(img1);
}
```

Implementation XXXIII

```
FILE *img2;
CURL* curl2 = curl_easy_init();
if (curl2) {
    img2 = fopen("two.png","wb");
    curl_easy_setopt(curl2, CURLOPT_URL, two.img.c_str());
    curl_easy_setopt(curl2, CURLOPT_WRITEFUNCTION, write_data);
    curl_easy_setopt(curl2, CURLOPT_WRITEDATA, img2);
    curl_easy_perform(curl2);
    curl_easy_cleanup(curl2);
    fclose(img2);
}

FILE *img3;
CURL* curl3 = curl_easy_init();
if (curl3) {
    img3 = fopen("three.png","wb");
```

Implementation XXXIV

```
        curl_easy_setopt(curl3, CURLOPT_URL, three.img.c_str());
        curl_easy_setopt(curl3, CURLOPT_WRITEFUNCTION, write_data);
        curl_easy_setopt(curl3, CURLOPT_WRITEDATA, img3);
        curl_easy_perform(curl3);
        curl_easy_cleanup(curl3);
        fclose(img3);
    }

    b1->image(new Fl_PNG_Image("one.png"));
    b1->color(FL_WHITE);
    b1->redraw();
    b2->image(new Fl_PNG_Image("two.png"));
    b2->color(FL_WHITE);
    b2->redraw();
    b3->image(new Fl_PNG_Image("three.png"));
    b3->color(FL_WHITE);
    b3->redraw();
```

Implementation XXXV

```
        return dw;
    }
```

Implementation XXXVI

cbSearch
Goal: Call getCard to find data on card entered by user and display its image

- Send user input to getCard
- Use CURL to download card image
- Display image in search window
- Call makeDisplayWindow

Implementation XXXVII

```
#include "lab.h"

void cbSearch(Fl_Button*,void*)
{
    string s = sq->value();
    Card ci = getCard(s);

    if(ci.name == "error")
    {
        if(dw) dw->hide();
        cw->image(new Fl_Image(0,0,0));
        cw->label("error: card not found");
        cw->redraw();
    }

    else
    {
```

Implementation XXXVIII

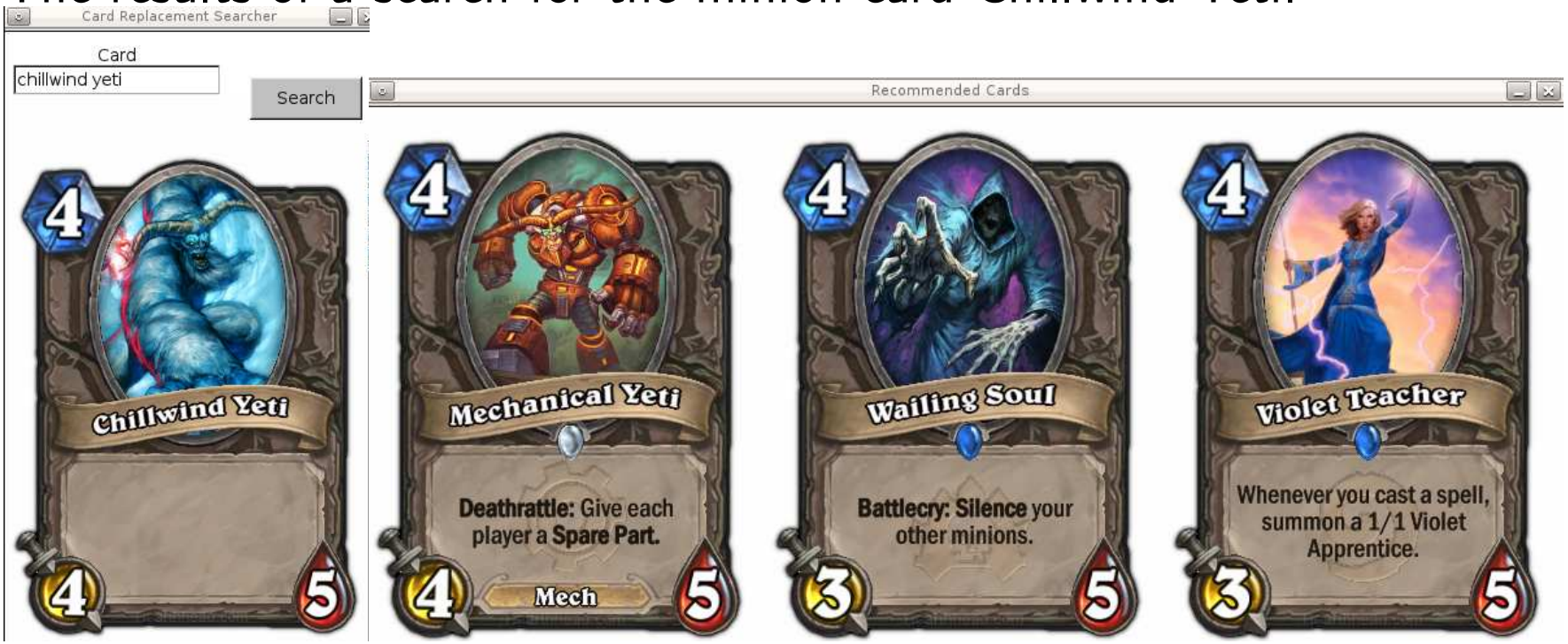
```
FILE *picture;
CURL* curl = curl_easy_init();
if (curl) {
    picture = fopen("card.png","wb");
    curl_easy_setopt(curl, CURLOPT_URL, ci.img.c_str());
    curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, write_data);
    curl_easy_setopt(curl, CURLOPT_WRITEDATA, picture);
    curl_easy_perform(curl);
    curl_easy_cleanup(curl);
    fclose(picture);
}
cw->label();
cw->image(new Fl_PNG_Image("card.png"));
cw->redraw();

if(dw) dw->hide();
makeDisplayWindow(ci)->show();
```

Implementation XXXIX

```
            dw->redraw();
        }
    }
```

Test I

The results of a search for the minion card Chillwind Yeti.



Test II

The results of a search for the spell card Frost Nova.

Test III

The results of a search for the weapon card Arcanite Reaper.