FINAL PROJECT
Go Fish v1.1

Produced by Taylor Nesby Fall 2014
For CIS-17C - 48956

Instructions

Go Fish follows the standard rules of the popular game go fish and can be launched via Go_Fish.exe! Click the Start Button to begin. This will instantiate a deck of cards created using a map and shuffled using the random shuffle algorithm within the STL. The goal of the game is to amass as many sets, which are groups of four cards with the same value, as possible before the cards run dry. The player with the highest score by this time wins. Each set is worth one point.

The game begins by dealing each player seven cards from the pool. Each player's hand is in the form of a deque. From here the human player must ask the computer for a card value they are looking for. The player must have at least one of that same card value already in their hand. An example of a card you could be dealt is 2H which indicates 2 of Hearts. The first character represents its value and the second its suit. Therefore the value is equal to two. Another example is JD which has a value of 11 and is diamond suited. Card values are as follows:

2=2	6=6	10=10
3=3	7=7	J=11
4=4	8=8	Q=12
5=5	9=9	K=13
		A=14

Once the human player has decided on which value to ask for, the value must be typed in and submitted for the computer to respond. If the computer owns any cards with the same value then they are handed over and the player is allotted guesses until a wrong guess occurs. On the other hand, if the computer does not have any cards with the asked value then the human player must draw a card from the pool (this adds a card to the human player's hand). Should the human player obtain a set, the program will automatically account for it by removing the cards in the set from the human player's hand and adding one to the human player score. Each of the human player's guesses is stored in a vector for the computer to draw upon. Guesses are removed when sets are placed.

It is then the computer's turn. The computer will follow the same rules mentioned in the paragraph above. Once their turn begins the AI will not ask for your input on whether or not you have a card: your opponent will simply take it and the score automatically updates. The player with the most sets wins.

Technical Features

My game was coded in C++ using Microsoft Visual Studios 2010. The main function, Player.h, is comprised of 1,050 lines. Tree.h is composed of 381 lines. Player.h contains 16 lines while card.h was created in 19 lines of code. In total my projects spans approximately 1,433 lines. Visual representations and much more can be found in the Doxygen generated report.

Coding concepts include the use of the following:

Concept	Functionality	Location
Map	Uses a map to initialize card suits and value.	375 in Form1.h
Vector	Holds the Player's guesses. This is to supplement the AI's actions should there be future expansions.	11 in Player.h
Algorithm	Shuffles the deck in random order.	382 in Form1.h
Deque	Acts as the Player's hand.	9 in Player.h
Recursion	If the computer guess right it will guess again until wrong.	675 and 774 in Form1.h
Tree	Keeps highscores.	Tree.h
Custom Classes	See below for more information.	
Custom Functions	See below for more information.	

Custom Functions

I also coded the following custom functions (references Form1.h unless otherwise noted):

- **map<string, int> makeDeck();**
  - Creates the initial deck. Declared at line 375. Defined at line 959.
- **void computerAction();**
  - Computer takes their turn. Implements recursion if guess is correct. Called at lines 448, 491, 503, 774, and 791. Defined at line 675.
- **void updateHand();**
  - Updates the UI showing the player's hand. Called at lines 406, 430, 468,497, 582, 610, 744, amd 829. Defined at line 853.
- **void updateScore();**
  - Updates the UI showing the game score. Called at lines 356, 431, 470, 499, 584, 612, 745, and 831. Defined at line 873.
- **bool findSet(Player &x, int v)**
  - Determines if there is a set in a given player's hand. Called at lines 411, 422, 460, 575, 738, and 813. Defined at line 888.
- **bool winner();**
  - Determines if there is a winner. Called at lines 472, 510, 585, 747, and 834. Defined at line 911.
- **void removeCards(Player &x, int v, string g)**
  - Removes cards from the deque of a Player structure. Called at lines 485, 602, 621, 763, 771, 788, 824, and 904. Defined at line 1,016.

The following functions rely on an "if clicked" scenario:

- **private: System::Void exitBtn_Click(System::Object^ sender, System::EventArgs^ e)**
  - Exits application. Found at line 346.
- **public: System::Void startBtn_Click(System::Object^ sender, System::EventArgs^ e)**

- Initializes and clears the playing board. Deals out cards. Found at line 351.

- **private: System::Void poolPic_Click(System::Object^ sender, System::EventArgs^ e)**
  - Draws a card, checks if there is a set, updates hand and score UI. Found at line 438.
- **private: System::Void submitBtn_Click(System::Object^ sender, System::EventArgs^ e)**
  - Checks if opposition has card and if so adds to hand & updates score and hand. Player can then guess again. Found at line 518.

Classes

- **card.h**
  - Utilizes a struct, titled card, to hold a card's value(int) and description aka suit (string)

```
using namespace std;
class card
{
public:
        string desc;
        int value;
        card(string d, int v)
        {
                desc=d;
                value=v;
        }
        string getDesc()
  {return desc;}
  int getValue()
  {return value;}
        ~card(void)
        {
        }
};
```

- **Player.h**
  - Utilizes a deque<card> of the aforementioned card class to hold the Player's cards thus creating a hand. Also includes a vector<int> to hold a Player's guesses.

```
#pragma once
#include "card.h"
#include <vector>
#include <deque>
class Player
{
public:
        //hand
  deque<card> hand;
  //guesses
  vector<int> guess;
        Player()
        {}
        virtual ~Player(void)
        {}
};
```

- **Tree.h**
  - Custom created tree to keep track of the high scores. It's memory lasts so long as the program is running. If a new highscore is detected a message detailing such is displayed.

```
#include <iostream>
#include <vector>
using namespace std;
class Tree
{
 private:
   struct Node
   {
      int data;
      int height;
      Node* left;
      Node* right;
   };
   Node* root;
   Node* worker;
public:
   //constructor
   Tree()
   {
      root=NULL;
   }
   //destructor
   ~Tree()
   {
      freeNode(root);
   }
   //frees the node
   void freeNode(Node* leaf)
   {
      if ( leaf != NULL )
      {
         freeNode(leaf->left);
         freeNode(leaf->right);
         delete leaf;
      }
   }
   //inserts a new leaf to the tree
   void insertLeaf(int v)
   {
      if(root==NULL)
         root=createLeaf(v);
      insertLeaf(root,v);
   }
```

```
//to add a node
void insertLeaf(Node* n, int v)
{
   //Check if tree is empty
   if(n==NULL)
      n=createLeaf(v); //add leaf
   else if(v<=n->data)
   {
      //If it is pointing to something already (recursive)
      if(n->left!=NULL)
      {
         insertLeaf(n->left,v);
      }
      //If it is not pointing to anything
      else
      {
         n->left=createLeaf(v);
         //balance
         balanceLeft(n,v);
      }
   }
   else if(v>n->data)
   {
       //If it is pointing to something already (recursive)
      if(n->right!=NULL)
      {
         insertLeaf(n->right,v);
      }
      //If it is not pointing to anything
      else
      {
         n->right=createLeaf(v);
         //balance
         balanceRight(n,v);
      }
   }
   n->height = max(height(n->left), height(n->right)) + 1;
}
//prints the tree data
vector<int> high(int c)
{
                  vector<int> highScores;
   if(root!=NULL)
   {
      switch(c)
      {
         case 1:
            highScores=preOrder(root);
```

```
            break;
         case 2:
            highScores=postOrder(root);
            break;
         case 3:
            highScores=inOrder(root);
                                                  break;
      }
   }
                  return highScores;
}
//creates a new node(leaf)
Node* createLeaf(int v)
{
   Node* temp=new Node;
   temp->data=v;
   temp->height=0;
   temp->left=NULL;
   temp->right=NULL;
   return temp;
}
//gets height of the node
int height(Node *n)
{
   return n == NULL ? -1 : n->height;
}
void balanceLeft(Node *n, int v)
{
   if (height(n->left) - height(n->right) == 2)
   {
      if (v < n->left->data)
         n = rotateWithLeftChild(n);
      else
         n = doubleWithLeftChild(n);
   }
}
void balanceRight(Node *n, int v)
{
   if (height(n->right) - height(n->left) == 2)
   {
      if (v > n->right->data)
         n = rotateWithRightChild(n);
      else
        n = doubleWithRightChild(n);
   }
}
//rotate tree node with left child
Node *rotateWithLeftChild(Node* k2)
```

```
{
   Node *k1 = k2->left;
   k2->left = k1->right;
   k1->right = k2;
   k2->height = max(height(k2->left), height(k2->right)) + 1;
   k1->height = max(height(k1->left), k2->height) + 1;
   return k1;
}
//rotate tree node with right child
Node *rotateWithRightChild(Node *k1)
{
   Node *k2 = k1->right;
   k1->right = k2->left;
   k2->left = k1;
   k1->height = max(height(k1->left), height(k1->right)) + 1;
   k2->height = max(height(k2->right), k1->height) + 1;
   return k2;
}
//double rotate
Node *doubleWithLeftChild(Node *k3)
{
   k3->left = rotateWithRightChild(k3->left);
   return rotateWithLeftChild(k3);
}
//double rotate
Node *doubleWithRightChild(Node *k1)
{
   k1->right = rotateWithLeftChild(k1->right);
   return rotateWithRightChild(k1);
}
//pre order transversal
vector<int> preOrder(Node* n)
{
                vector<int> highScores;
   if(n)
   {
      highScores.push_back(n->data);
      preOrder(n->left);
      preOrder(n->right);
   }
                return highScores;
}
//post order transversal
vector<int> postOrder(Node* n)
{
                vector<int> highScores;
   if (n)
   {
```

```
      postOrder(n->left);
      postOrder(n->right);
      highScores.push_back(n->data);
   }
                  return highScores;
}
//in order transversal
vector<int> inOrder(Node* n)
{
                  vector<int> highScores;
   if (n)
   {
      inOrder(n->left);
                        highScores.push_back(n->data);
      inOrder(n->right);
   }
                  return highScores;
}
//searches for a value in the tree
bool search(Node *r, int v)
{
   bool found = false;
   while ((r != NULL) && !found)
   {
      int rval = r->data;
      if (v < rval)
         r = r->left;
      else if (v > rval)
         r = r->right;
      else
      {
         found = true;
         break;
      }
      found = search(r, v);
   }
   return found;
}
//searches for a value in the tree-returns true if found, false if not found
bool search(int v)
{
   return search(root, v);
}
//deletes leaf and reorders
void removeNode(int v)
{
   removeNodep(v,root);
}
```

```
void removeNodep(int v, Node* parent)
{
   //If not empty
   if(root!=NULL)
   {
      //If found the data to delete
      if(root->data==v)
      {
         removeRootMatch();
      }
      //If not found
      else
      {
         //Left child
         if(v<parent->data&&parent->left!=NULL)
         {
            parent->left->data==v ?
            removeMatch(parent,parent->left,true):
            removeNodep(v,parent->left);
         }
         //Right child
         else if(v>parent->data&&parent->right!=NULL)
         {
            parent->right->data==v ?
            removeMatch(parent,parent->right,false):
            removeNodep(v,parent->right);
         }
         else
         {
            cout<<"The data "<<v<<" was not found in the tree."<<endl;
         }
      }
   }
   //If empty
   else
   {
      cout<<"The tree is empty."<<endl;
   }
}
void removeRootMatch()
{
   if(root!=NULL)
   {
      Node* delPtr=root;
      int rootData=root->data;
      int smallestright;
      //0 children
      if(root->left==NULL&&root->right==NULL)
```

```
        {
           root=NULL;
           delete delPtr;
        }
        //1 Child
        //  child on right
        else if(root->left==NULL&&root->right!=NULL)
        {
           root=root->right;
           delPtr->right=NULL;
           delete delPtr;
        }
        //  child on left
        else if(root->left!=NULL&&root->right==NULL)
        {
           root=root->left;
           delPtr->left=NULL;
           delete delPtr;
        }
        //2 Children
        else
        {
           smallestright=findsmallestp(root->right);
           removeNodep(smallestright,root);
           root->data=smallestright;
        }
     }
     else
     {
        cout<<"Can not remove root, tree is empty"<<endl;
     }
}
       void removeMatch(Node* parent, Node* match, bool left)
{
     if(root!=NULL)
     {
        Node* delPtr;
        int matchData=match->data;
        int smallestright;
        //0 Children
        if(match->left==NULL&&match->right==NULL)
        {
           delPtr=match;
           left==true?parent->left=NULL:parent->right=NULL;
           delete delPtr;
        }
        //1 Child
        //  Right child
```

```
        else if(match->left==NULL&&match->right!=NULL)
        {
           left==true?parent->left=match->right:parent->right=match->right;
           match->right=NULL;
           delPtr=match;
           delete delPtr;
        }
        //  Left child
        else if(match->left!=NULL&&match->right==NULL)
        {
           left==true?parent->left=match->left:parent->right=match->left;
           match->left=NULL;
           delPtr=match;
           delete delPtr;
        }
        //2 Children
        else
        {
           smallestright=findsmallestp(match->right);
           removeNodep(smallestright,match);
           match->data=smallestright;
        }
     }
     else
     {
        cout<<"Can not remove match. Tree is empty"<<endl;
     }
}
int findsmallestp(Node* ptr)
{
   //If empty
   if(root==NULL)
   {
      cout<<"Tree is empty."<<endl;
      return -1000;
   }
   //When not empty
   else
   {
      if(ptr->left!=NULL)
      {
         return findsmallestp(ptr->left);
      }
      else
      {
         return ptr->data;
      }
   }
```

```
                }
                    };
```

Form 1.h (the main function)

```
#pragma once
//standard libraries
#include <algorithm>
#include <cstdlib>
#include <map>
#include <sstream>
#include <ctime>
#include <string>
//classes
#include "Player.h"
#include "Tree.h"
//function prototypes
map<string, int> makeDeck();
void computerAction();
void updateHand();
void updateScore();
bool findSet(Player &,int);
bool winner();
void removeCards(Player &x, int v, string g);

namespace GoFish {

        using namespace System;
        using namespace System::ComponentModel;
        using namespace System::Collections;
        using namespace System::Windows::Forms;
        using namespace System::Data;
        using namespace System::Drawing;
        //
        // <summary>
        /// Summary for Form1
        /// </summary>
        vector<int> set;
        deque<card> pool;
        Player computer;
   Player person;
        bool wrong=true;
```

```
bool timeToDraw=false;
bool computerTurn=false;
bool timeToSubmit=false;
int personCount=0, computerCount=0;

public ref class Form1 : public System::Windows::Forms::Form
{

public:
	Form1(void)
	{
		InitializeComponent();
		//
		//TODO: Add the constructor code here
		//
	}

protected:
	/// <summary>
	/// Clean up any resources being used.
	/// </summary>
	~Form1()
	{
		if (components)
		{
			delete components;
		}
	}
private: System::Windows::Forms::Button^  startBtn;
private: System::Windows::Forms::Button^  exitBtn;

private: System::Windows::Forms::PictureBox^  poolPic;
private: System::Windows::Forms::Label^  poolLbl;
private: System::Windows::Forms::Label^  youLbl;
private: System::Windows::Forms::Label^  compPairsLbl;
private: System::Windows::Forms::Label^  youPairsLbl;
private: System::Windows::Forms::Button^  submitBtn;
private: System::Windows::Forms::TextBox^  guessBox;
private: System::Windows::Forms::Label^  messageLbl;
private: System::Windows::Forms::Label^  bannerLbl;
private: System::Windows::Forms::PictureBox^  pictureBox1;
```

```
        private: System::Windows::Forms::Label^  personCountLbl;
        private: System::Windows::Forms::Label^  computerCountLbl;

        protected:

        private:
                /// <summary>
                /// Required designer variable.
                /// </summary>
                System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code
                /// <summary>
                /// Required method for Designer support - do not modify
                /// the contents of this method with the code editor.
                /// </summary>
                void InitializeComponent(void)
                {
                        System::ComponentModel::ComponentResourceManager^
resources = (gcnew
System::ComponentModel::ComponentResourceManager(Form1::typeid));
                        this->startBtn = (gcnew System::Windows::Forms::Button());
                        this->exitBtn = (gcnew System::Windows::Forms::Button());
                        this->poolPic = (gcnew System::Windows::Forms::PictureBox());
                        this->poolLbl = (gcnew System::Windows::Forms::Label());
                        this->youLbl = (gcnew System::Windows::Forms::Label());
                        this->compPairsLbl = (gcnew System::Windows::Forms::Label());
                        this->youPairsLbl = (gcnew System::Windows::Forms::Label());
                        this->submitBtn = (gcnew System::Windows::Forms::Button());
                        this->guessBox = (gcnew System::Windows::Forms::TextBox());
                        this->messageLbl = (gcnew System::Windows::Forms::Label());
                        this->bannerLbl = (gcnew System::Windows::Forms::Label());
                        this->pictureBox1 = (gcnew System::Windows::Forms::PictureBox());
                        this->personCountLbl = (gcnew System::Windows::Forms::Label());
                        this->computerCountLbl = (gcnew System::Windows::Forms::Label());
                        (cli::safe_cast<System::ComponentModel::ISupportInitialize^
>(this->poolPic))->BeginInit();
                        (cli::safe_cast<System::ComponentModel::ISupportInitialize^
>(this->pictureBox1))->BeginInit();
                        this->SuspendLayout();
                        //
```

```
                    // startBtn
                    //
                    this->startBtn->AutoSize = true;
                    this->startBtn->BackColor =
System::Drawing::SystemColors::ControlDarkDark;
                    this->startBtn->Font = (gcnew System::Drawing::Font(L"Microsoft
Sans Serif", 14.25F, System::Drawing::FontStyle::Bold,
System::Drawing::GraphicsUnit::Point,
                            static_cast<System::Byte>(0)));
                    this->startBtn->ForeColor = System::Drawing::Color::LimeGreen;
                    this->startBtn->Location = System::Drawing::Point(915, 531);
                    this->startBtn->Name = L"startBtn";
                    this->startBtn->Size = System::Drawing::Size(75, 40);
                    this->startBtn->TabIndex = 0;
                    this->startBtn->Text = L"Start";
                    this->startBtn->UseVisualStyleBackColor = false;
                    this->startBtn->Click += gcnew System::EventHandler(this,
&Form1::startBtn_Click);
                    //
                    // exitBtn
                    //
                    this->exitBtn->BackColor =
System::Drawing::SystemColors::ControlDarkDark;
                    this->exitBtn->Font = (gcnew System::Drawing::Font(L"Microsoft
Sans Serif", 14.25F, System::Drawing::FontStyle::Bold,
System::Drawing::GraphicsUnit::Point,
                            static_cast<System::Byte>(0)));
                    this->exitBtn->ForeColor = System::Drawing::Color::Red;
                    this->exitBtn->Location = System::Drawing::Point(915, 577);
                    this->exitBtn->Name = L"exitBtn";
                    this->exitBtn->Size = System::Drawing::Size(75, 40);
                    this->exitBtn->TabIndex = 1;
                    this->exitBtn->Text = L"Exit";
                    this->exitBtn->UseVisualStyleBackColor = false;
                    this->exitBtn->Click += gcnew System::EventHandler(this,
&Form1::exitBtn_Click);
                    //
                    // poolPic
                    //
                    this->poolPic->BackColor = System::Drawing::Color::Transparent;
```

```
            this->poolPic->BackgroundImage =
(cli::safe_cast<System::Drawing::Image^
>(resources->GetObject(L"poolPic.BackgroundImage")));
            this->poolPic->ImageLocation = L"..\\card.jpg";
            this->poolPic->Location = System::Drawing::Point(782, 512);
            this->poolPic->Name = L"poolPic";
            this->poolPic->Size = System::Drawing::Size(76, 108);
            this->poolPic->TabIndex = 4;
            this->poolPic->TabStop = false;
            this->poolPic->Click += gcnew System::EventHandler(this,
&Form1::poolPic_Click);
            //
            // poolLbl
            //
            this->poolLbl->Anchor =
static_cast<System::Windows::Forms::AnchorStyles>((((System::Windows::Forms::Anchor
Styles::Top | System::Windows::Forms::AnchorStyles::Bottom)
                        | System::Windows::Forms::AnchorStyles::Left)
                        | System::Windows::Forms::AnchorStyles::Right));
            this->poolLbl->BackColor = System::Drawing::Color::Transparent;
            this->poolLbl->Font = (gcnew System::Drawing::Font(L"Microsoft
Sans Serif", 18, System::Drawing::FontStyle::Regular,
System::Drawing::GraphicsUnit::Point,
                        static_cast<System::Byte>(0)));
            this->poolLbl->ForeColor = System::Drawing::SystemColors::Control;
            this->poolLbl->Location = System::Drawing::Point(782, 481);
            this->poolLbl->Name = L"poolLbl";
            this->poolLbl->Size = System::Drawing::Size(76, 28);
            this->poolLbl->TabIndex = 7;
            this->poolLbl->Text = L"Pool";
            this->poolLbl->TextAlign =
System::Drawing::ContentAlignment::MiddleCenter;
            //
            // youLbl
            //
            this->youLbl->AutoSize = true;
            this->youLbl->BackColor =
System::Drawing::SystemColors::ControlDarkDark;
            this->youLbl->Font = (gcnew System::Drawing::Font(L"Palatino
Linotype", 20.25F, System::Drawing::FontStyle::Regular,
System::Drawing::GraphicsUnit::Point,
```

```
                    static_cast<System::Byte>(0)));
                this->youLbl->Location = System::Drawing::Point(81, 512);
                this->youLbl->Name = L"youLbl";
                this->youLbl->Size = System::Drawing::Size(148, 36);
                this->youLbl->TabIndex = 8;
                this->youLbl->Text = L"Your Cards";
                this->youLbl->TextAlign =
System::Drawing::ContentAlignment::MiddleCenter;
                //
                // compPairsLbl
                //
                this->compPairsLbl->BackColor =
System::Drawing::Color::Transparent;
                this->compPairsLbl->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 15.75F,
System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
                    static_cast<System::Byte>(0)));
                this->compPairsLbl->ForeColor = System::Drawing::Color::White;
                this->compPairsLbl->Location = System::Drawing::Point(12, 9);
                this->compPairsLbl->Name = L"compPairsLbl";
                this->compPairsLbl->Size = System::Drawing::Size(142, 50);
                this->compPairsLbl->TabIndex = 10;
                this->compPairsLbl->Text = L"Opponent Score";
                this->compPairsLbl->TextAlign =
System::Drawing::ContentAlignment::MiddleCenter;
                this->compPairsLbl->Click += gcnew System::EventHandler(this,
&Form1::compPairsLbl_Click);
                //
                // youPairsLbl
                //
                this->youPairsLbl->BackColor = System::Drawing::Color::Transparent;
                this->youPairsLbl->Font = (gcnew System::Drawing::Font(L"Microsoft
Sans Serif", 15.75F, System::Drawing::FontStyle::Regular,
System::Drawing::GraphicsUnit::Point,
                    static_cast<System::Byte>(0)));
                this->youPairsLbl->ForeColor = System::Drawing::Color::White;
                this->youPairsLbl->Location = System::Drawing::Point(185, 9);
                this->youPairsLbl->Name = L"youPairsLbl";
                this->youPairsLbl->Size = System::Drawing::Size(142, 50);
                this->youPairsLbl->TabIndex = 11;
                this->youPairsLbl->Text = L"Your Score";
```

```
            this->youPairsLbl->TextAlign =
System::Drawing::ContentAlignment::MiddleCenter;
            //
            // submitBtn
            //
            this->submitBtn->AutoSize = true;
            this->submitBtn->BackColor =
System::Drawing::SystemColors::ControlDarkDark;
            this->submitBtn->Cursor = System::Windows::Forms::Cursors::Cross;
            this->submitBtn->Font = (gcnew System::Drawing::Font(L"Microsoft
Sans Serif", 20.25F, System::Drawing::FontStyle::Regular,
System::Drawing::GraphicsUnit::Point,
                static_cast<System::Byte>(0)));
            this->submitBtn->ForeColor = System::Drawing::Color::LimeGreen;
            this->submitBtn->Location = System::Drawing::Point(558, 573);
            this->submitBtn->Name = L"submitBtn";
            this->submitBtn->Size = System::Drawing::Size(116, 43);
            this->submitBtn->TabIndex = 12;
            this->submitBtn->Text = L"Submit";
            this->submitBtn->UseVisualStyleBackColor = false;
            this->submitBtn->Click += gcnew System::EventHandler(this,
&Form1::submitBtn_Click);
            //
            // guessBox
            //
            this->guessBox->BackColor =
System::Drawing::SystemColors::ControlDarkDark;
            this->guessBox->Font = (gcnew System::Drawing::Font(L"Microsoft
Sans Serif", 18, System::Drawing::FontStyle::Regular,
System::Drawing::GraphicsUnit::Point,
                static_cast<System::Byte>(0)));
            this->guessBox->ForeColor = System::Drawing::Color::LimeGreen;
            this->guessBox->Location = System::Drawing::Point(495, 512);
            this->guessBox->Name = L"guessBox";
            this->guessBox->Size = System::Drawing::Size(257, 35);
            this->guessBox->TabIndex = 13;
            this->guessBox->Text = L"Enter Card Value Here";
            this->guessBox->TextAlign =
System::Windows::Forms::HorizontalAlignment::Center;
            //
            // messageLbl
```

```
                //
                this->messageLbl->BackColor = System::Drawing::Color::White;
                this->messageLbl->Font = (gcnew System::Drawing::Font(L"Microsoft
Sans Serif", 15.75F, System::Drawing::FontStyle::Regular,
System::Drawing::GraphicsUnit::Point,
                        static_cast<System::Byte>(0)));
                this->messageLbl->ForeColor = System::Drawing::Color::Black;
                this->messageLbl->Location = System::Drawing::Point(603, 74);
                this->messageLbl->Name = L"messageLbl";
                this->messageLbl->Size = System::Drawing::Size(250, 153);
                this->messageLbl->TabIndex = 14;
                this->messageLbl->Text = L"Welcome to a game of good old
fashioned Go Fish! The name\'s Ron and I\'ll be your "
                        L"opponent today. Good luck dog. You\'re gonna need it!";
                this->messageLbl->TextAlign =
System::Drawing::ContentAlignment::MiddleCenter;
                //
                // bannerLbl
                //
                this->bannerLbl->BackColor = System::Drawing::Color::LimeGreen;
                this->bannerLbl->Font = (gcnew System::Drawing::Font(L"Microsoft
Sans Serif", 14.25F, System::Drawing::FontStyle::Regular,
System::Drawing::GraphicsUnit::Point,
                        static_cast<System::Byte>(0)));
                this->bannerLbl->ForeColor = System::Drawing::Color::White;
                this->bannerLbl->Location = System::Drawing::Point(435, 9);
                this->bannerLbl->Name = L"bannerLbl";
                this->bannerLbl->Size = System::Drawing::Size(178, 50);
                this->bannerLbl->TabIndex = 15;
                this->bannerLbl->Text = L"Your Turn";
                this->bannerLbl->TextAlign =
System::Drawing::ContentAlignment::MiddleCenter;
                //
                // pictureBox1
                //
                this->pictureBox1->BackColor =
System::Drawing::Color::Transparent;
                this->pictureBox1->Image = (cli::safe_cast<System::Drawing::Image^
>(resources->GetObject(L"pictureBox1.Image")));
                this->pictureBox1->Location = System::Drawing::Point(540, 9);
                this->pictureBox1->Name = L"pictureBox1";
```

```
                    this->pictureBox1->Size = System::Drawing::Size(371, 311);
                    this->pictureBox1->TabIndex = 16;
                    this->pictureBox1->TabStop = false;
                    //
                    // personCountLbl
                    //
                    this->personCountLbl->Anchor =
static_cast<System::Windows::Forms::AnchorStyles>((((System::Windows::Forms::Anchor
Styles::Top | System::Windows::Forms::AnchorStyles::Bottom)
                                | System::Windows::Forms::AnchorStyles::Left)
                                | System::Windows::Forms::AnchorStyles::Right));
                    this->personCountLbl->AutoSize = true;
                    this->personCountLbl->BackColor =
System::Drawing::Color::Transparent;
                    this->personCountLbl->Font = (gcnew
System::Drawing::Font(L"Palatino Linotype", 72, System::Drawing::FontStyle::Regular,
System::Drawing::GraphicsUnit::Point,
                                static_cast<System::Byte>(0)));
                    this->personCountLbl->ForeColor = System::Drawing::Color::White;
                    this->personCountLbl->ImageAlign =
System::Drawing::ContentAlignment::TopCenter;
                    this->personCountLbl->Location = System::Drawing::Point(206, 74);
                    this->personCountLbl->Name = L"personCountLbl";
                    this->personCountLbl->Size = System::Drawing::Size(103, 129);
                    this->personCountLbl->TabIndex = 17;
                    this->personCountLbl->Text = L"0";
                    this->personCountLbl->TextAlign =
System::Drawing::ContentAlignment::MiddleCenter;
                    //
                    // computerCountLbl
                    //
                    this->computerCountLbl->Anchor =
static_cast<System::Windows::Forms::AnchorStyles>((((System::Windows::Forms::Anchor
Styles::Top | System::Windows::Forms::AnchorStyles::Bottom)
                                | System::Windows::Forms::AnchorStyles::Left)
                                | System::Windows::Forms::AnchorStyles::Right));
                    this->computerCountLbl->AutoSize = true;
                    this->computerCountLbl->BackColor =
System::Drawing::Color::Transparent;
```

```
            this->computerCountLbl->Font = (gcnew
System::Drawing::Font(L"Palatino Linotype", 72, System::Drawing::FontStyle::Regular,
System::Drawing::GraphicsUnit::Point,
                static_cast<System::Byte>(0)));
            this->computerCountLbl->ForeColor =
System::Drawing::Color::White;
            this->computerCountLbl->ImageAlign =
System::Drawing::ContentAlignment::TopCenter;
            this->computerCountLbl->Location = System::Drawing::Point(30, 74);
            this->computerCountLbl->Name = L"computerCountLbl";
            this->computerCountLbl->Size = System::Drawing::Size(103, 129);
            this->computerCountLbl->TabIndex = 18;
            this->computerCountLbl->Text = L"0";
            this->computerCountLbl->TextAlign =
System::Drawing::ContentAlignment::MiddleCenter;
            //
            // Form1
            //
            this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
            this->AutoScaleMode =
System::Windows::Forms::AutoScaleMode::Font;
            this->BackgroundImage = (cli::safe_cast<System::Drawing::Image^
>(resources->GetObject(L"$this.BackgroundImage")));
            this->ClientSize = System::Drawing::Size(1012, 631);
            this->Controls->Add(this->messageLbl);
            this->Controls->Add(this->bannerLbl);
            this->Controls->Add(this->guessBox);
            this->Controls->Add(this->submitBtn);
            this->Controls->Add(this->youPairsLbl);
            this->Controls->Add(this->compPairsLbl);
            this->Controls->Add(this->youLbl);
            this->Controls->Add(this->poolLbl);
            this->Controls->Add(this->poolPic);
            this->Controls->Add(this->exitBtn);
            this->Controls->Add(this->startBtn);
            this->Controls->Add(this->pictureBox1);
            this->Controls->Add(this->personCountLbl);
            this->Controls->Add(this->computerCountLbl);
            this->ForeColor = System::Drawing::Color::Transparent;
            this->Name = L"Form1";
            this->Text = L"Go Fish! ";
```

```
                        (cli::safe_cast<System::ComponentModel::ISupportInitialize^
>(this->poolPic))->EndInit();
                        (cli::safe_cast<System::ComponentModel::ISupportInitialize^
>(this->pictureBox1))->EndInit();
                        this->ResumeLayout(false);
                        this->PerformLayout();

                }
#pragma endregion
                //if the exit button is pressed then the application will exit
        private: System::Void exitBtn_Click(System::Object^  sender, System::EventArgs^
e)
                        {
                                Application::Exit();
                        }
                        //if the start/restart button is pressed then the application will clear
all past data and initialize the structures
        public: System::Void startBtn_Click(System::Object^  sender, System::EventArgs^
e)
                        {
                                //initialize and clear everything
                                personCount=0;
                                computerCount=0;
                                set.clear();
                                updateScore();
                                pool.clear();
                                person.hand.clear();
                                person.guess.clear();
                                computer.hand.clear();
                                computer.guess.clear();
                                if(startBtn->Text=="Start")
                                        startBtn->Text="Restart";
                                //welcome message
                                messageLbl->Text="Welcome to a game of good old
fashioned Go Fish! The name's Ron and I'll be your opponent today. Good luck dog. You're
gonna need it!";
                                //banner message
                                bannerLbl->Text="Your Turn";
                                bannerLbl->BackColor=Color::LimeGreen;
                                //hand label set to blank
                                youLbl->Text="";
```

```
                                //Initialize the random number generator
                                srand(static_cast<unsigned int>(time(0)));
                                //uses a map to initialize card suits and value
                                map<string,int> Deck=makeDeck();
                                //moves deck data from map to deque using iterator
                                for(map<string,int>::iterator
it=Deck.begin();it!=Deck.end();++it)
                                {
                                        pool.push_back(card(it->first,it->second));
                                }
                        //shuffles the deque wth shuffle algorithm
                                random_shuffle(pool.begin(),pool.end());
                                //distributes the cards to the AI and player (7 each) starting at
pool.begin
                                int i=0;
                                for(deque<card>::iterator it=pool.begin();i<14;++it)
                                {
                                        //computer gets evens
                                        if(i%2==0)
                                        {
                                                computer.hand.push_back(card(it->getDesc(),
it->getValue()));
                                        }
                                        //player gets odds
                                        else if(i%2==1)
                                        {
                                                person.hand.push_back(card(it->getDesc(),
it->getValue()));
                                        }
                                        i++;
                                }
                                //remove cards from the pool that were copied above
                                for(int i=0;i<14;i++)
                                {
                                        if(!pool.empty())
                                                pool.pop_front();
                                }
                                //sets the player card box to show all of the player's cards
                                updateHand();
                                //find sets
```

```
                                for(deque<card>::iterator
it=computer.hand.begin();it!=computer.hand.end();++it)
                                {
                                        int v=it->getValue();
                                        if(findSet(computer, v))
                                        {
                                                computerCount++;
                                                //add it to set
                                                set.push_back(v);
                                        }
                                }
                                //find sets
                                for(deque<card>::iterator
it=person.hand.begin();it!=person.hand.end();++it)
                                {
                                        int v=it->getValue();
                                        if(findSet(person, v))
                                        {
                                                personCount++;
                                                //add it to set
                                                set.push_back(v);
                                        }
                                }
                                ////sets the player card box to show all of the player's cards
                                updateHand();
                                updateScore();
                                timeToDraw=false;
                                computerTurn=false;
                                timeToSubmit=true;
                                wrong=true;
                        }
                        //if the player clicks on the draw button at the appropiate time a
new card from the pool of cards is added to the player's hand
private: System::Void poolPic_Click(System::Object^  sender, System::EventArgs^  e)
                {
                         int value;
                        //add a card to the player's hand
                        if(timeToDraw && !computerTurn && pool.empty() &&
!timeToSubmit)
                        {
                                bannerLbl->Text="Pool is empty.";
```

```
		timeToDraw=false;
		computerTurn=true;
		timeToSubmit=false;
		computerAction();
	}
	else if(timeToDraw && !pool.empty() && !timeToSubmit)
	{
		//assign card to person
		deque<card>::iterator it=pool.begin();
		person.hand.push_back(card(it->getDesc(), it->getValue()));
		//capture value
		value=it->getValue();
		//pop the card from the deck
		if(!pool.empty())
				pool.pop_front();
		if(findSet(person, value))
		{

			//add 1 to score
			personCount++;
			//add to set vector
			set.push_back(value);
			//update hand
			updateHand();
			//update score
			updateScore();
			//check if winner
			if(winner())
			{
				timeToDraw=false;
				computerTurn=false;
				timeToSubmit=false;
			}
			else
			{
				//no winner
				//remove values from guess

				for(int i=0;i<4;i++)
				{
					removeCards(person,value,"guess");
```

```
                                        }

                                                timeToDraw=false;
                                                computerTurn=true;
                                                timeToSubmit=false;
                                                computerAction();
                                                }
                                        }
                                        else//no set was found
                                        {
                                                //update hand
                                                updateHand();
                                                //update score
                                                updateScore();
                                                timeToDraw=false;
                                                computerTurn=true;
                                                timeToSubmit=false;
                                                computerAction();
                                        }
                                }
                                else
                                {
                                        bannerLbl->Text="Not time to draw.";
                                }
                                if(winner())
                                {
                                        timeToDraw=false;
                                        computerTurn=false;
                                        timeToSubmit=false;
                                }
                        }
                         //if submit button is pressed check to see if AI has the card the player input
private: System::Void submitBtn_Click(System::Object^  sender, System::EventArgs^  e)
                        {
                        int value;
                        int choice=rand()% 4 + 1;
                        int cardCount=0;
                        bool valid=false;
                        string model;
                        String^ MyString;
                        if(timeToSubmit)
```

```
		{
			bannerLbl->Text="Your Turn";
			bannerLbl->BackColor=Color::LimeGreen;
			//can only guess for card values that player ownes
			for(deque<card>::iterator
it=person.hand.begin();it!=person.hand.end();++it)
			{
				model=to_string(static_cast<long long>(it->getValue()));
				MyString = gcnew String(model.c_str());
				//if card is in player hand then continue
				if(MyString==guessBox->Text)
					valid=true;
			}
			if(valid)
			{
				for(deque<card>::iterator
toy=computer.hand.begin();toy!=computer.hand.end();++toy)
				{
					model=to_string(static_cast<long
long>(toy->getValue()));
					MyString = gcnew String(model.c_str());
					if(MyString==guessBox->Text)
					{
						//we have a match so..
						//add it to the player's hand

person.hand.push_back(card(toy->getDesc(),toy->getValue()));
						//capture value
						value=toy->getValue();
						cardCount++;
						wrong=false;
					}
				}

				if(!wrong)
				{
					switch(choice)
					{
						case 1:
							messageLbl->Text="Ah shucks. You got
me.";
```

```
                                        break;
                                        case 2:
                                                messageLbl->Text="You found me out.
Here's your card(s).";
                                        break;
                                        case 3:
                                                messageLbl->Text="You may have won
the battle but the war still rages.";
                                        break;
                                        case 4:
                                                messageLbl->Text="Wow, nice guess.
Here's you card(s).";
                                        break;
                                }
                                //check for a set of 4
                                if(findSet(person, value))
                                {
                                        //add 1 to score
                                        personCount++;
                                        //add set to set vector
                                        set.push_back(value);
                                        //update hand values shown
                                        updateHand();
                                        //update score
                                        updateScore();
                                        if(winner())
                                        {
                                                //prepares next move
                                                timeToDraw=false;
                                                timeToSubmit=false;
                                                computerTurn=false;
                                        }
                                        else
                                        {
                                                computerTurn=false;
                                                timeToDraw=false;
                                                timeToSubmit=true;
                                                //remove value from guess
                                                if(!person.guess.empty())
                                                {
                                                        for(int i=0; i<4;i++)
```

```
                                                {

removeCards(person,value,"guess");
                                                }
                                        }
                                }
                        }
                        else//it doenst find a set
                        {
                                //update hand values shown
                                updateHand();
                                //update score
                                updateScore();
                                computerTurn=false;
                                timeToDraw=false;
                                timeToSubmit=true;
                                if(!computer.hand.empty())
                                {
                                        //get rid of the matched cards owned by
the computer
                                        for(int i=0;i<cardCount;i++)
                                        {

removeCards(computer,value,"hand");
                                        }
                                }
                        }

                }
                else
                {
                        //there is no match
                        switch(choice)
                        {
                                case 1:
                                        messageLbl->Text="Nice try but I don't
have that card. Time for you to draw a card pal. Go fish.";
                                break;
                                case 2:
                                        messageLbl->Text="Nope. Go 'head and
go fish.";
```

```
                                        break;
                                        case 3:
                                                messageLbl->Text="Go Fish.";
                                        break;
                                        case 4:
                                                messageLbl->Text="Go fish. Tell me how
the water is when you're done.";
                                        break;
                                }
                                //convert text to int
                                int bad = Convert::ToInt32(guessBox->Text);
                                //add bad guess to the guess vector belonging to
person
                                person.guess.push_back(bad);
                                //prepares next move
                                timeToDraw=true;
                                timeToSubmit=false;
                                computerTurn=false;
                                wrong=true;
                        }
                        wrong=true;

                }
                //else it is not a valid guess and player must guess again
                else
                {
                        bannerLbl->Text="Must guess values you own.";
                        computerTurn=false;
                        timeToDraw=false;
                        timeToSubmit=true;
                        wrong=true;
                }

        }
        else
        {
                bannerLbl->Text="Not time to submit";
                wrong=true;
        }
    }
void computerAction()
```

```
{
	//initializes chatateristics of the computer's turn
	bool guessResult=false;
	int guessNum, num, random;
	bool alreadyTaken = true;
	int cardCount=0;
	bannerLbl->Text="Opponent's Turn";
	bannerLbl->BackColor=Color::Red;
	//AI-finds suitable guess


	//when it guesses again it can't be the same as last guess

	//if guess is wrong
	while(alreadyTaken)
	{
		//get guess
		random=rand() % computer.hand.size();
		int i=0;
		for(deque<card>::iterator
it=computer.hand.begin();it!=computer.hand.end();++it)
		{
			if(i==random)
				guessNum=it->getValue();
			i++;
		}
		//if guess is in the set vector then guess again
		if(!set.empty())
		{
			for(vector<int>::iterator s=set.begin();s!=set.end();++s)
			{
				if(guessNum==*s)
					alreadyTaken = true;
				else
					alreadyTaken = false;
			}
		}
		else
			alreadyTaken = false;
	}
		//continue. check to see if person has the guessNum
```

```
                        //string model=to_string(static_cast<long long>(guessNum));
                        //String^ MyString = gcnew String(model.c_str());
                        if(!person.hand.empty())
                        {
                                for(deque<card>::iterator
it=person.hand.begin();it!=person.hand.end();++it)
                                {
                                        if(guessNum==it->getValue())
                                        {
                                                //add person's card to the computer
hand

computer.hand.push_back(card(it->getDesc(),it->getValue()));
                                                guessResult=true;
                                                cardCount++;
                                        }
                                }
                        }
                        //check for a set, update
                        if (guessResult)
                        {
                                messageLbl->Text="Hey thanks for the card(s).";
                                //add guess to vector
                                computer.guess.push_back(guessResult);
                                //determine if computer has the set
                                if(findSet(computer, guessNum))
                                {
                                        //increase score
                                        computerCount++;
                                        //add set value to set vector
                                        set.push_back(guessNum);
                                        updateHand();
                                        updateScore();
                                        //check to see if there is a winner
                                        if(winner())
                                        {
                                                timeToDraw=false;
                                                computerTurn=false;
                                                timeToSubmit=false;
                                        }
                                        else//no winner
```

```
                                        {
                                                timeToDraw=false;
                                                computerTurn=true;
                                                timeToSubmit=false;
                                                //deletes computer's guesses for the set
                                                if(!computer.guess.empty())
                                                {
                                                        for(int i=0;i<4;i++)
                                                        {

removeCards(computer,guessNum,"guess");
                                                        }
                                                }
                                                //delete person's card from hand
                                                if(!person.hand.empty())
                                                {
                                                        for(int i=0;i<cardCount;i++)
                                                        {

removeCards(person,guessNum,"hand");
                                                        }
                                                }
                                                computerAction();
                                        }
                                }
                                //no set was found->no winner
                                else
                                {
                                        timeToDraw=false;
                                        computerTurn=true;
                                        timeToSubmit=false;
                                        //delete person's card from hand
                                        if(!person.hand.empty())
                                        {
                                                for(int i=0;i<cardCount;i++)
                                                {

removeCards(person,guessNum,"hand");
                                                }
                                        }
                                        computerAction();
```

```
                        }
                    }
                    else
                    //guess was wrong,, computer draws card, updates, player
turn
                    {
                        //computer draws
                        if(pool.empty())
                        {
                            bannerLbl->Text="Pool is empty.";
                            timeToDraw=true;
                            computerTurn=false;
                            timeToSubmit=true;
                        }
                        else
                        //not empty so draw a card
                        {
                            if(!pool.empty())
                            {
                                deque<card>::iterator it=pool.begin();
                                num=it->getValue();

computer.hand.push_back(card(it->getDesc(), it->getValue()));
                                if(findSet(computer, num))
                                {
                                    //add 1 to score
                                    computerCount++;
                                    //add to set vector
                                    set.push_back(num);
                                    //remove values from guess
                                    if(!computer.guess.empty())
                                    {
                                        for(int i=0;i<cardCount;)
                                        {

removeCards(computer,num,"guess");
                                        }
                                    }
                                }
                                //update hand
                                updateHand();
```

```
                                                //update score
                                                updateScore();
                                                if(!pool.empty())
                                                        pool.pop_front();
                                                if(winner())
                                                {
                                                        timeToDraw=false;
                                                        computerTurn=false;
                                                        timeToSubmit=false;
                                                }
                                                else
                                                {
                                                        bannerLbl->Text="Your Turn";

bannerLbl->BackColor=Color::LimeGreen;
                                                        timeToDraw=false;
                                                        computerTurn=false;
                                                        timeToSubmit=true;
                                                }
                                        }
                                }

                }
void updateHand()
{
        //update hand values shown
                                        youLbl->Text="";
                                        int i=0, perLine=8;
                                        if(!person.hand.empty())
                                        {
                                                for(deque<card>::iterator
it=person.hand.begin();it!=person.hand.end();++it)
                                                {
                                                        if(i%perLine==0 && i!=0)
                                                                youLbl->Text+="\n";
                                                        string model=it->getDesc();
                                                        String^ MyString = gcnew
String(model.c_str());
                                                        youLbl->Text+=MyString + " ";
                                                        i++;
```

```
                                        }
                                        youLbl->Text+="\n";
                                }
}
void updateScore()
{
        string model;
        String^ MyString;

        //update person score
        model=to_string(static_cast<long long>(personCount));
        MyString = gcnew String(model.c_str());
        personCountLbl->Text=MyString;

        //update computer score
        model=to_string(static_cast<long long>(computerCount));
        MyString = gcnew String(model.c_str());
        computerCountLbl->Text=MyString;

}
bool findSet(Player &x, int v)
{
        //test to find a set
        int i=0;
        for(deque<card>::iterator it=x.hand.begin();it!=x.hand.end();++it)
        {
                if(v==it->getValue())
                {
                        //we have a match so add 1 to the count
                        i++;
                }
        }
        if(i==4)
        {
                for(int i=0;i<4;i++)
                {
                        removeCards(x,v,"hand");
                }
                return true;
        }
        else
```

```
                return false;
}
bool winner()
{
        if(computer.hand.empty() || person.hand.empty())
                        {
                                Tree highScores;
                                vector<int> sorted;
                                //use score to determine winner
                                if(computerCount>personCount)
                                {
                                        bannerLbl->Text="Computer Wins!";
                                        messageLbl->Text="I win. Play again?\n";
                                        highScores.insertLeaf(computerCount);
                                        sorted=highScores.high(3);
                                        vector<int>::iterator it=sorted.end();
                                        it--;
                                        if(computerCount>=*it)
                                                messageLbl->Text="I win. Play again?\nNew
Highscore!";
                                }
                                else if(computerCount<personCount)
                                {
                                        bannerLbl->Text="You Win!";
                                        messageLbl->Text="Bow wow wow I lost! Play
again?\n";
                                        highScores.insertLeaf(personCount);
                                        sorted=highScores.high(3);
                                        vector<int>::iterator it=sorted.end();
                                        it--;
                                        if(personCount>=*it)
                                                messageLbl->Text="Bow wow wow I lost! Play
again?\nNew Highscore!";
                                }
                                else if(computerCount==personCount)
                                {
                                        bannerLbl->Text="Tie Game!";
                                        messageLbl->Text="We're evenly matched. Play
again?\n";
                                        highScores.insertLeaf(computerCount);
                                        sorted=highScores.high(3);
```

```
                                        vector<int>::iterator it=sorted.end();
                                        it--;
                                        if(computerCount>=*it)
                                                messageLbl->Text="We're evenly matched. Play
again?\nNew Highscore!";
                                }
                                return true;
                        }
        else
                return false;
}
private: System::Void compPairsLbl_Click(System::Object^  sender, System::EventArgs^  e)
{
                }
};}
map<string,int> makeDeck()
{
   //create map and fill with cards
   map<string,int> deck;
   int i=2;
   for(int count=0;count<52;count++)
   {
      string end;
      if(count<13)
      {
         end="C";
      }
      else if(count>=13 && count<26)
      {
         end="D";
      }
      else if(count>=26 && count<39)
      {
         end="S";
      }
      else if(count>=39 && count<52)
      {
         end="H";
      }
      if(i==11)
      {
```

```
            deck["J"+end]=i;
            i++;
            continue;
        }
        if(i==12)
        {
            deck["Q"+end]=i;
            i++;
            continue;
        }
        if(i==13)
        {
            deck["K"+end]=i;
            i++;
            continue;
        }
        if(i==14)
        {
            deck["A"+end]=i;
            i=2;
            continue;
        }
        string num;
        stringstream convert;
        convert << i;
        num = convert.str();
        deck[num + end]=i;
        i++;
    }
    return deck;
}
void removeCards(Player &x, int v, string g)
{
        if(g=="hand")
        {
                //remove cards from hand
                if(!x.hand.empty())
                {
                        for(deque<card>::iterator tif=x.hand.begin();tif!=x.hand.end();++tif)
                        {
                                int s=tif->getValue();
```

```
                                if(s==v)
                                {
                                        x.hand.erase(tif);
                                        break;
                                }
                        }
                }
        }
        else if(g=="guess")
        {
                //remove cards from guess
                if(!x.guess.empty())
                {
                        for(vector<int>::iterator tif=x.guess.begin();tif!=x.guess.end();++tif)
                        {
                                if(*tif==v)
                                {
                                        x.guess.erase(tif);
                                        break;
                                }
                        }
                }
        }
}
```