Main is pretty simple, it only initializes the random number generator, gets the instance of GameManager, starts the game with GameManager's startGame() function, and asks the user whether he would like to battle again once the first battle has ended. If the user presses y or Y, GameManager's startGame() is called again. If the user presses n or N the game ends, otherwise you get an error stating that your choice is invalid.

Since here in main the GameManager is called for the first time, it is also the one and only time GameManager is instantiated. This is because the GameManager class is a singleton and since we only need one GameManager throughout the game and we need to access it from different parts in the program, I thought this design pattern would work great for it. I learned it from my CIS-18B class, Java Programming, and although the syntax is a little different it was hard to implement it.

```
/*
 * Jose Sandoval
 * CIS-17C: C++ Programming
 * November 11, 2014
 * Project #1: Card Fight Game
 * Description: A fighting game that uses cards.
 */

// System libraries
#include <ctime>

// Header files
#include "GameManager.h"

int main()
{
	// Initialize the random number generator.
	srand(static_cast<unsigned int>(time(0)));

	// Instatiate GameManager object and start game.
	GameManager::getInstance()->startGame();

	char answer;

	do
	{
		cout << "Would you like to battle again? (Y/N)" << endl;
		cin >> answer;

		switch (answer)
		{
		case 'y':
		case 'Y':
			GameManager::getInstance()->startGame();
			break;
		case 'n':
		case 'N':
```

```
                cout << "Goodbye!" << endl;
                break;
            default:
                cout << "ERROR: Not a valid choice!\n\n" << endl;
            }
        } while (answer != 'n' && answer != 'N');

        return 0;
}
```

The singleton GameManager class is where everything in the program occurs. Its private members are a static pointer of an instance of itself, a gameState variable which is actually an Enum, the vector for players has not been implemented yet but I hope it would help me to reduce repeated code by eliminating code that is specific to each player. Right now there is a pointer to a player and a pointer to an enemy, or AI, which both are objects of the Player class. There's also a GUI object that takes care of drawing the "graphics" in the game, mostly boxes for the cards, stats and banner. There's a map of characters where the name of a character is stored as the string and the universe the character comes from is stored as a value. There's a stack of Items which represents a stack of special cards that contain items that can help the player. Those items are Health Packages which increase the health of the player who uses it, Energy packages which increase the energy of the player who uses it, Cards Swap item which replaces the cards on your current hand for new cards, very useful for when you have low attack cards, and the Special Attack item which allows the player to perform a special attack. These items can be obtained only when the player or AI use an Item Card. The GameManager constructor, its copy constructor and overloaded equals operator are private because of singleton design pattern requirements. The cleanup() function would be the equivalent of a destructor and it would delete the pointers of the player and enemy.

The public members of the GameManager class are a static function called getInstance() that returns a pointer to the GameManager object so it can be called from anywhere in the program. A void setGameState() function that takes a GameState enum as a parameter and sets the gameState variable of the GameManager. A getGameState() function that returns the gameState of the game. There are also getPlayer() and getEnemy() functions that return pointers to the player and enemy. A getCharacters() function that returns a map of the available characters in the game. A getGUI() function that returns a pointer to the gui of the GameManager class. And finally void functions such as startGame() which starts a game, gameOver() which ends the game and announces the winner, startBattle() which starts a battle, compareCards() which compares the chosen cards by the player and enemy, replaceCards() which replaces the cards used by the player and enemy, and battleMenu() which displays the battle menu.

```
/*
 * Jose Sandoval
 * CIS-17C: C++ Programming
 * November 11, 2014
 * Project #1: Card Fight Game
 * Description: A fighting game that uses cards.
 */

#ifndef GAMEMANAGER_H
#define GAMEMANAGER_H

#include <fstream>
#include <stack>

#include "Player.h"
#include "GUI.h"
#include "GameState.h"

class GameManager
{
private:
```

```
        static GameManager* _instance;
        GameState::GameState gameState;
        vector<Player*> players;
        Player* player;
        Player* enemy;
        GUI gui;
        map<string, string> characters;
        stack<Item> items;
        GameManager();
        GameManager(const GameManager&);
        GameManager& operator=(const GameManager&);
        void cleanUp();
public:
        static GameManager* getInstance();
        void setGameState(GameState::GameState);
        GameState::GameState getGameState();
        Player* getPlayer();
        Player* getEnemy();
        map<string, string> getCharacters();
        GUI* getGUI();
        void startGame();
        void gameOver();
        void startBattle();
        void compareCards();
        void replaceCards();
        void battleMenu();
};

#endif // GAMEMANAGER_H



/*
 * Jose Sandoval
 * CIS-17C: C++ Programming
 * November 11, 2014
 * Project #1: Card Fight Game
 * Description: A fighting game that uses cards.
 */

#include "GameManager.h"

GameManager* GameManager::_instance = NULL;

// Constructor for GameManager singleton class.
GameManager::GameManager()
{
        // Set gamestate to GAMEOVER
        gameState = GameState::GAMEOVER;

        // Create variables for reading in
        // the Characters.txt file
        string line;
        ifstream data("Characters.txt");
        vector<string> information;
```

```
	// Read in Characters.txt data.
	if (data.is_open())
	{
		// Save the data to the information vector.
		while (getline(data, line, ','))
			information.push_back(line);

		// Close the Characters.txt file.
		data.close();
	}
	else
		cout << "ERROR: File could not be opened!" << endl;

	// Fill the characters map with the data from vector.
	for (int i = 0; i < information.size(); i += 2)
		characters[information.at(i)] = information.at(i + 1);
}

// cleanUp function would be the destructor
// for the GameManager class.
void GameManager::cleanUp()
{
	delete player;
	delete enemy;
}

// Returns the one and only instance of
// GameManager.
GameManager* GameManager::getInstance()
{
	// If instance is NULL instantiate
	// a GameManager object.
	if (_instance == NULL)
		_instance = new GameManager;

	return _instance;
}

// Sets the gameState of the game.
void GameManager::setGameState(GameState::GameState gameState)
{
	this->gameState = gameState;
}

// Returns the gameState of the game.
GameState::GameState GameManager::getGameState()
{
	return gameState;
}

// Returns a pointer to the player.
Player* GameManager::getPlayer()
{
	return player;
```

```
}

// Returns a pointer to the enemy.
Player* GameManager::getEnemy()
{
	return enemy;
}

// Returns a pointer to the characters map.
map<string, string> GameManager::getCharacters()
{
	return characters;
}

// Returns a pointer to the gui.
GUI* GameManager::getGUI()
{
	return &gui;
}

// Starts the game.
void GameManager::startGame()
{
	// Change the gameState to PLAYING
	gameState = GameState::PLAYING;

	// Display game banner
	gui.displayBanner();

	// Instantiate player objects.
	// The passed argument indicates whether
	// the player is AI (Artificial Intelligence)
	player = new Player(false);
	enemy = new Player(true);

	// Instantiate items stack.
	Item* anItem;
	int itemsStackSize = 25;
	for (int item = 0; item < itemsStackSize; item++)
	{
		anItem = new Item(rand() % 4 + 1);
		items.push(*anItem);
	}

	// Start Battle
	startBattle();
}

// Ends the game and displays who won the battle.
void GameManager::gameOver()
{
	cout << "\n" << setw(25) << "GAME OVER!" << endl;

	// Check what player won based on who has a health of 0.
	if (player->getCharacter()->getHealth() == 0)
```

```
			cout << setw(25) <<enemy->getCharacter()->getName() << " WINS!!!" <<
endl;
		else
			cout << setw(25) << player->getCharacter()->getName() << " WINS!!!" <<
endl;

		// Calls the destructor to clean up memory.
		cleanUp();
}

// Starts the battle.
void GameManager::startBattle()
{
		// Display the chosen characters.
		cout << "\n\n" << setw(30) << "THE BATTLE STARTS!\n"
		     << setw(20) << player->getCharacter()->getName()
		        << " VS " << enemy->getCharacter()->getName()
		        << "\n\n\n";

		// Game loop.
		do
		{
			// Display stats.
			gui.drawStats();

			// Display battle menu.
			battleMenu();

			// TEMPORARY FIX FOR STOPPING AFTER SPECIAL
			// ATTACK KILLS OPPONENT.
			if (gameState != GameState::GAMEOVER)
			{
				// Displays the cards chosen by
				// the players.
				gui.drawChosenCards();

				// Compares the cards to see who will
				// attack.
				compareCards();
			}
		} while (gameState != GameState::GAMEOVER);

		// Displays the stats one last time
		// before calling gameOver.
		gui.drawStats();

		// Calls gameOver to finish the game.
		gameOver();
}

// Compares the chosen cards by the players
// to see who will attack.
void GameManager::compareCards()
{
		// If the player or the enemy chose an item
```

```
// card add the item to the bag of the one
// who chose it.
if (player->getChosenCard()->isItemCard()
      || enemy->getChosenCard()->isItemCard())
{
      // If the player is the one who chose the item
      // card show him what he got and add it to
      // his bag.
      if (!player->isPlayerAI()
            && player->getChosenCard()->isItemCard())
      {
            cout << "\nYou got item: "
                  << items.top().getType() << endl;

            player->addItem(items.top());
            items.pop();
      }

      // If the enemy is the one who chose the
      // item card just add it to his bag.
      if (enemy->getChosenCard()->isItemCard())
      {
            enemy->addItem(items.top());
            items.pop();
      }
}

// If the player chose a card with a higher attack
// then he is the one who attacks.
if (player->getChosenCard()->getAttack()
    > enemy->getChosenCard()->getAttack())
{
      cout << "\n" << setw(25)
            << "Player ATTACKS!" << endl;

      // Have the enemy take the attack of the player's
      // chosen card and the defense of his own card.
      enemy->getCharacter()
            ->defend(player->getChosenCard()->getAttack(),
            enemy->getChosenCard()->getDefense());
}

// If the attack value on the player's chosen card
// is the same as the attack value on the enemy's
// chosen card and they are not item cards, then
// it is a DRAW and both players have to choose
// another card.
else if (player->getChosenCard()->getAttack()
          == enemy->getChosenCard()->getAttack()
          && !player->getChosenCard()->isItemCard())
{
      cout << "\n" << setw(25)
            << "DRAW! Choose another card!" << endl;

      replaceCards();
```

```
			player->attack();
			enemy->attack();
			gui.drawChosenCards();
			compareCards();
		}
		// If none of the cases above is true it means
		// that enemy has a higher attack value on his
		// chosen card so he attacks.
		else
		{
			// If the enemy's chosen card is not an item
			// have the player take the damage.
			if (!enemy->getChosenCard()->isItemCard())
			{
				cout << "\n" << setw(25)
					<< "Enemy ATTACKS!" << endl;

				// Have the player take the attack value of the enemy's
				// chosen card and the defense of his own card.
				player->getCharacter()
					->defend(enemy->getChosenCard()->getAttack(),
					player->getChosenCard()->getDefense());
			}
		}

		// Replace the chosen cards.
		replaceCards();

		cout << "\n";
}

// Replaces the cards chosen by the players.
void GameManager::replaceCards()
{
		player->getHand()->replaceCard();
		enemy->getHand()->replaceCard();
}

void GameManager::battleMenu()
{
		// Create choice variable.
		int choice;

		// Battle menu for a player who is not AI.
		if (!player->isPlayerAI())
		{
			do
			{
				cout << "Choose an option:\n"
					<< "1) Attack\n"
					<< "2) Items\n\n"
					<< "Choice: ";
				cin >> choice;

				switch (choice)
```

```
                {
                case 1:
                        // Player attacks.
                        player->attack();
                        break;
                case 2:
                        // If the bag of the layer is not
                        // empty look at his items.
                        if (!player->getItems()->empty())
                        {
                                // Display the items inside
                                // the player's bag.
                                player->viewItems();

                                cout << "\nChoose an option:\n"
                                        << "1) Apply item\n"
                                        << "2) Go back\n\n"
                                        << "Choice: ";
                                cin >> choice;

                                // Apply item.
                                if (choice == 1)
                                {
                                        // Create pointer that points to the
                                        // player's bag.
                                        map<string, int>* items = player->getItems();

                                        do
                                        {
                                                cout << "\nEnter item number: ";
                                                cin >> choice;
                                        } while (choice < 0 || choice > items-
>size());

                                        // Create an iterator and move it where the
chosen
                                        // item is located in the map.
                                        auto iterator = items->begin();
                                        advance(iterator, (choice - 1));

                                        // If the chosen item is a Cards Swap,
                                        // a Health Package, or an Energy
                                        // Package use it.
                                        if (iterator->first == "Cards Swap" ||
                                                iterator->first == "Health Package" ||
                                                iterator->first == "Energy Package")
                                                player->applyItem(choice);
                                        else
                                        {
                                                // If the chosen item is a Special
Attack,
                                                // make sure the player has the minimum
                                                // energy to perform it.
                                                if (player->getCharacter()->getEnergy()
> 10
```

```
                                                && iterator->first == "Special
Attack")
                                                player->applyItem(choice);
                                          else
                                          {
                                                cout << "You do not have the
minimum"
                                                      << " energy needed for a
special attack.\n";
                                          }
                                    }

                                    // Set choice equal to 0 to break out of
loop.
                                    choice = 0;
                              }
                        }
                        // Otherwise tell him he has not items.
                        else
                              cout << "\nYou have no items in your bag." << endl;
                        break;
                  default:
                        cout << "ERROR: Not a valid choice." << endl;
                  }

                  cout << "\n\n";

            } while (choice != 1);
      }

      // AI decisions code.
      if (enemy->isPlayerAI())
      {
            // Create a variable for the chosen item.
            int chosenItem = 0;

            int count = 0;

            // If the bag of the enemy is not empty
            // have him look at its contents.
            if (!enemy->getItems()->empty())
            {
                  // Create pointer that points to the
                  // enemy's bag.
                  map<string, int>* items = enemy->getItems();

                  // Iterate through the enemy's bag.
                  for (auto iterator = items->begin();
                        iterator != items->end(); iterator++)
                  {
                        count++;

                        // If item is a health Package.
                        if (iterator->first == "Health Package")
                        {
```

```
                                    // Create minimum health threshold variable.
                                    int minHealthThreshold = 15;

                                    // Check if the health of the player is
                                    // below the minimum health threshold,
                                    // if it is use a health package.
                                    if (enemy->getCharacter()->getHealth()
                                          < minHealthThreshold)
                                          chosenItem = count;
                              }

                              // If item is an Energy Package.
                              if (iterator->first == "Energy Package")
                              {
                                    // Use the energy package only when
                                    // the energy is less than the maximum
                                    // energy.
                                    if (enemy->getCharacter()->getEnergy()
                                          < enemy->getCharacter()->getMaxEnergy())
                                          chosenItem = count;
                              }

                              // If item is Cards Swap.
                              if (iterator->first == "Cards Swap")
                                    chosenItem = count;

                              // If item is Special Attack.
                              if (iterator->first == "Special Attack")
                              {
                                    // Miimum energy needed for a special attack.
                                    int minEnergyNeeded = 10;

                                    // If the energy of the enemy is greater than or
                                    // equal to the minimum energy needed then use it.
                                    if (enemy->getCharacter()->getEnergy() >=
minEnergyNeeded)
                                          chosenItem = count;
                              }
                        }

                        // Apply the chosen item.
                        if (chosenItem != 0)
                              enemy->applyItem(chosenItem);
                        // If an item was not chosen then attack.
                        else
                              enemy->attack();
                  }
                  // If the enemy's bag is empty attack.
                  else
                        enemy->attack();
            }
}
```

```
/*
 * Jose Sandoval
 * CIS-17C: C++ Programming
 * November 11, 2014
 * Project #1: Card Fight Game
 * Description: A fighting game that uses cards.
 */

#ifndef PLAYER_H
#define PLAYER_H

// System Libraries
#include <iterator>
#include <iomanip>
#include <map>

// Header Files
#include "Hand.h"
#include "Item.h"
#include "Character.h"

using namespace std;

class Player
{
private:
	Hand hand;
	Card* chosenCard;
	Character* character;
	bool isAI;
	map<string, int> items;
public:
	Player(bool);
	~Player();
	bool isPlayerAI();
	Character* getCharacter();
	Card* getChosenCard();
	Hand* getHand();
	map<string, int>* getItems();
	void attack();
	void addItem(Item);
	void viewItems();
	void applyItem(int);
};

#endif // PLAYER_H

#include "Player.h"
#include "GameManager.h"

// Overloaded constructor for the Player class.
Player::Player(bool isAI)
{
	// Set the AI variable for the player.
	this->isAI = isAI;
```

```
	// Make a local map of characters for further code readability.
	map<string, string> characters = GameManager::getInstance()->getCharacters();

	// Create a variable to store the number of the chosen character.
	int chosenCharacter;

	// If the player is not AI, show a list of characters
	// to choose from.
	if (!isAI)
	{
		// Count variable is used for character enumeration.
		int count = 0;

		cout << "Choose a character:\n\n" << "   NAME:"
			<< setw(25) << "UNIVERSE:" << endl;

		// Iterate through the characters map and print
		// them out so the player can choose one.
		for (auto iterator = characters.begin();
			iterator != characters.end(); iterator++)
		{
			count++;
			cout << count << ") "  << iterator->first
				<< setw(25) << iterator->second << endl;
		}

		// Receive chosen character player input.
		do
		{
			cout << "\nChosen characters: ";
			cin >> chosenCharacter;
		} while (chosenCharacter > characters.size() || chosenCharacter < 1);
	}
	// AI decision code for choosing a character.
	else
		chosenCharacter = rand() % 8 + 1;

	// Create an iterator and move it where the chosen
	// character is located in the map.
	auto iterator = characters.begin();
	advance(iterator, (chosenCharacter - 1));

	// Instantiate a character object.
	character = new Character(iterator->first);
}

// Destructor for the player class.
Player::~Player()
{
	delete character;
}

// Returns wheter the player is AI.
bool Player::isPlayerAI()
```

```
{
	return isAI;
}

// Returns a pointer to the card
// chosen by the player.
Card* Player::getChosenCard()
{
	return chosenCard;
}

// Returns a pointer to the character
// of the player.
Character* Player::getCharacter()
{
	return character;
}

// Returns a pointer to the hand
// of the player.
Hand* Player::getHand()
{
	return &hand;
}

// Returns whether the player has
// items in his bag.
map<string, int>* Player::getItems()
{
	return &items;
}

// Prints out the cards on the hand of the
// player and allows him to choose a card.
void Player::attack()
{
	// Create a variable for the chosen card.
	int selectedCard = 0;

	// If the player is not AI ask him for to
	// choose a card from his hand.
	if (!isAI)
	{
		do
		{
			cout << endl;

			// Print the cards on the hand of the
			// player.
			hand.displayHand();

			cout << "\nChoose a card: ";
			cin >> selectedCard;
		} while (selectedCard > 5 || selectedCard < 1);
```

```
			// Decrease the chosenCard variable so
			// that it matches with the subscripts
			// of the deque container.
			selectedCard--;
		}
		// AI decision code for choosing the best card
		// in the hand of the player.
		else
		{
			Card* currentCard;

			int bestCardAttack = 0;
			int bestCardDefense = 0;

			// Variable used to stop choosing cards
			// if an item card has already been chosen.
			bool itemCardChosen = false;

			// Loop through the 5 cards on the hand.
			for (int card = 0; card < 5; card++)
			{
				// Get a card.
				currentCard = hand.getCard(card);

				// If it is an item card, choose it.
				if (currentCard->isItemCard())
				{
					selectedCard = card;
					itemCardChosen = true;
				}

				// Check which car has the best attack and
				// best defense so it can be chosen as long as
				// an item card has not been chosen.
				if (bestCardAttack < currentCard->getAttack() && !itemCardChosen)
				{
					bestCardAttack = currentCard->getAttack();
					bestCardDefense = currentCard->getDefense();
					selectedCard = card;

					if (bestCardDefense < currentCard->getDefense())
					{
						bestCardDefense = currentCard->getDefense();
						selectedCard = card;
					}
				}
			}
		}

		// Points the currentCard pointer to
		// the chosen card.
		chosenCard = hand.getCard(selectedCard);
}

// Adds an item to the player's bag.
```

```
void Player::addItem(Item item)
{
	items[item.getType()] += 1;
}

// Displays the items on the player's bag.
void Player::viewItems()
{
	int count = 0;

	cout << "\n\nItems in your bag: " << endl;

	for (auto iterator = items.begin(); iterator != items.end(); iterator++)
	{
		count++;
		cout << count << ") " << iterator->first
			<< "     Quantity: " << iterator->second << endl;
	}
}

// Applies an item from the player's bag.
void Player::applyItem(int item)
{
	// Create an iterator and move it where the chosen
	// item is located in the map.
	auto iterator = items.begin();
	advance(iterator, (item - 1));

	// Create a used item variable and get the
	// type of the used item.
	string usedItem = iterator->first;

	// Create a variable for a special attack.
	string specialAttack;

	// If the used item is a special attack
	// display the special attacks of the
	// player's character.
	if (usedItem == "Special Attack")
	{
		// I NEED TO FINISH IMPLEMENTING THIS
		// BY OUTSOURCING SPECIAL ATTACKS TO
		// A .TXT FILE.

		int choice;
		int attack;
		int energyCost;

		if (!isAI)
		{
			do
			{
				cout << "\n\nChoose a Special Attack:\n"
					<< "1) Kamehameha		A: 25 E: 30\n"
					<< "2) Spirit Bomb		A: 40 E: 50\n"
```

```
                        << "3) Dragon Fist       A: 20 E: 20\n"
                        << "4) Ki Blast          A: 15 E: 10\n\n"
                        << "Choice: ";
                cin >> choice;

                cout << "\n\n" << character->getName()
                        << " used ";

                switch (choice)
                {
                case 1:
                        cout << "Kamehameha." << endl;
                        attack = 25;
                        energyCost = 30;
                        break;
                case 2:
                        cout << "Spirit Bomb." << endl;
                        attack = 40;
                        energyCost = 50;
                        break;
                case 3:
                        cout << "Dragon Fist." << endl;
                        attack = 20;
                        energyCost = 20;
                        break;
                case 4:
                        cout << "Ki Blast." << endl;
                        attack = 15;
                        energyCost = 10;
                        break;
                default:
                        cout << "ERROR: Not a valid choice" << endl;
                        choice = 0;
                }
        } while (choice == 0 || character->getEnergy() < energyCost);

        GameManager::getInstance()->getEnemy()
                ->getCharacter()->defend(attack, 0);
        getCharacter()->decreaseEnergy(energyCost);
        GameManager::getInstance()->getGUI()->drawStats();
}
else
{
        choice = rand() % 4 + 1;

        cout << "\n\n" << character->getName()
                << " used ";

        switch (choice)
        {
        case 1:
                cout << "Kamehameha." << endl;
                attack = 25;
                energyCost = 30;
                break;
```

```
                case 2:
                        cout << "Spirit Bomb." << endl;
                        attack = 40;
                        energyCost = 50;
                        break;
                case 3:
                        cout << "Dragon Fist." << endl;
                        attack = 20;
                        energyCost = 20;
                        break;
                case 4:
                        cout << "Ki Blast." << endl;
                        attack = 15;
                        energyCost = 10;
                        break;
                default:
                        break;
                }

                GameManager::getInstance()->getPlayer()
                        ->getCharacter()->defend(attack, 0);
                character->decreaseEnergy(energyCost);
                GameManager::getInstance()->getGUI()->drawStats();
        }
    }

    // If the used item is a health package.
if (usedItem == "Health Package")
    {
        // Randomly choose the health contained
        // in the health package.
        int extraHealth = rand() % 11 + 5;

        // Display how much health was in the
        // health package.
        cout << "\n\nYour health package contained "
                << extraHealth << " health." << endl;

        // Add the health to the character.
        character->addHealth(extraHealth);

        // Display stats to see the added health.
        cout << "\n\n";
        GameManager::getInstance()->getGUI()->drawStats();
    }

    // If the used item is an energy package.
    if (usedItem == "Energy Package")
    {
        // Randomly choose the energy contained
        // in the energy package.
        int extraEnergy = rand() % 10 + 5;

        // Display how much energy was in the
        // energy package.
```

```
			cout << "\n\nYour energy package contained "
				<< extraEnergy << " energy." << endl;

			 // Add the energy to the character.
			character->addEnergy(extraEnergy);

			// Display stats to see the added energy.
			cout << "\n\n";
			GameManager::getInstance()->getGUI()->drawStats();
		}

		// If the used item is a cards swap item.
		if (usedItem == "Cards Swap")
		{
			// Display the old cards.
			cout << "\n\nYour old cards: " << endl;
			hand.displayHand();

			// Replace the cards on the hand
			// for new cards.
			for (int card = 0; card < 5; card++)
			{
				// Always get the first card
				// because when the card is replaced
				// the next card moves to the first spot.
				hand.getCard(0);
				hand.replaceCard();
			}

			// Display the new cards.
			cout << "\n\nYour new cards: " << endl;
			hand.displayHand();
		}

		// If the player has more than one of the
		// same item in his bag reduce the quantity
		// by one when the item is used.
		if (iterator->second > 1)
			items[usedItem] -= 1;
		// Otherwise delete the item from the bag.
		else
			items.erase(usedItem);
}
```

```
/*
 * Jose Sandoval
 * CIS-17C: C++ Programming
 * November 11, 2014
 * Project #1: Card Fight Game
 * Description: A fighting game that uses cards.
 */

#ifndef HAND_H
#define HAND_H

// System Libraries
#include <deque>
#include <string>
#include <iostream>

// Header Files
#include "Deck.h"

using namespace std;

class Hand
{
private:
	int chosenCard;
	Deck* deck;
	deque<Card> hand;
public:
	Hand();
	~Hand();
	Card* getCard(int);
	void replaceCard();
	void displayHand();
};

#endif // HAND_H



/*
 * Jose Sandoval
 * CIS-17C: C++ Programming
 * November 11, 2014
 * Project #1: Card Fight Game
 * Description: A fighting game that uses cards.
 */

#include "Hand.h"
#include "GameManager.h"

// Constructor for Hand class.
Hand::Hand()
{
	// Create a deck with 200 cards.
	deck = new Deck(200);
```

```
	// Fill the hand with 5 cards from the deck.
	for (int card = 0; card < 5; card++)
		hand.push_back(deck->getCard());
}

// Destructor for Hand class.
Hand::~Hand()
{
	delete deck;
}

// Get the chosen card from the hand.
Card* Hand::getCard(int chosenCard)
{
	this->chosenCard = chosenCard;

	return &hand.at(chosenCard);
}

// Replaces the chosen card from the hand
// with the next card on the deck.
void Hand::replaceCard()
{
	hand.erase(hand.begin() + chosenCard);
	hand.push_back(deck->getCard());
}

// Displays the cards on the hand.
void Hand::displayHand()
{
	GameManager::getInstance()->getGUI()->drawHandCards(&hand);
}
```

```
/*
 * Jose Sandoval
 * CIS-17C: C++ Programming
 * November 11, 2014
 * Project #1: Card Fight Game
 * Description: A fighting game that uses cards.
 */

#ifndef DECK_H
#define DECK_H

// System Libraries
#include <queue>

// Header Files
#include "Card.h"

using namespace std;

class Deck
{
private:
	int size;
	queue<Card> cards;
public:
	Deck(int);
	~Deck();
	int getSize();
	Card getCard();
};

#endif //DECK_H
```

```
/*
 * Jose Sandoval
 * CIS-17C: C++ Programming
 * November 11, 2014
 * Project #1: Card Fight Game
 * Description: A fighting game that uses cards.
 */

#include "Deck.h"

// Overloaded constructor of Deck class.
Deck::Deck(int size)
{
	this->size = size;

	for (int card = 0; card < size; card++)
	{
		Card* aCard = new Card(rand() % 9 + 1, rand() % 5 + 1);
		cards.push(*aCard);
	}
```

```
}

// Destructor for Deck class.
Deck::~Deck()
{
}

// Returns the size of the deck.
int Deck::getSize()
{
	return size;
}

// Returns the first card on the deck.
Card Deck::getCard()
{
	Card card = cards.front();
	cards.pop();
	return card;
}
```

```
/*
 * Jose Sandoval
 * CIS-17C: C++ Programming
 * November 11, 2014
 * Project #1: Card Fight Game
 * Description: A fighting game that uses cards.
 */

#ifndef CARD_H
#define CARD_H

class Card
{
private:
	int attack;
	int defense;
	bool isSpecialCard;
public:
	Card(int attack, int defense);
	~Card();
	int getAttack();
	int getDefense();
	bool isItemCard();
};

#endif //CARD_H
```

```
/*
 * Jose Sandoval
 * CIS-17C: C++ Programming
 * November 11, 2014
 * Project #1: Card Fight Game
 * Description: A fighting game that uses cards.
 */

#include "Card.h"

// Overloaded constructor for Card class.
Card::Card(int attack, int defense)
{
	// Set attack and defense of the card,
	// and make it a not special card.
	this->attack = attack;
	this->defense = defense;
	isSpecialCard = false;

	// Make it a special card if attack
	// value equals defense value and
	// set attack and defense values to 0.
	if (attack == defense)
	{
		isSpecialCard = true;
		this->attack = 0;
```

```
            this->defense = 0;
        }
}

// Destructor for Card class.
Card::~Card()
{
}

// Returns the attack value of the card.
int Card::getAttack()
{
        return attack;
}

// Returns the defense value of the card.
int Card::getDefense()
{
        return defense;
}

// Returns whether the card is a special card.
bool Card::isItemCard()
{
        return isSpecialCard;
}
```

```
/*
 * Jose Sandoval
 * CIS-17C: C++ Programming
 * November 11, 2014
 * Project #1: Card Fight Game
 * Description: A fighting game that uses cards.
 */

#ifndef CHARACTER_H
#define CHARACTER_H

// System Libraries
#include <string>

using namespace std;

class Character
{
private:
	string name;
	int health;
	int energy;
	int maxHealth;
	int maxEnergy;
public:
	Character(string);
	~Character();
	string getName();
	int getHealth();
	int getEnergy();
	int getMaxHealth();
	int getMaxEnergy();
	void addHealth(int);
	void addEnergy(int);
	void decreaseEnergy(int);
	void defend(int, int);
};

#endif // CHARACTER_H
```

```
/*
 * Jose Sandoval
 * CIS-17C: C++ Programming
 * November 11, 2014
 * Project #1: Card Fight Game
 * Description: A fighting game that uses cards.
 */

#include "Character.h"
#include "GameManager.h"

// Overloaded constructor for Character class.
Character::Character(string name)
```

```
{
	this->name = name;
	health = 50;
	energy = 50;
	maxHealth = health;
	maxEnergy = energy;
}

// Destructor for Character class.
Character::~Character()
{
}

// Returns the name of the character.
string Character::getName()
{
	return name;
}

// Returns the health of the character.
int Character::getHealth()
{
	return health;
}

// Returns the energy of the character.
int Character::getEnergy()
{
	return energy;
}

// Returns the maximum health of the character.
int Character::getMaxHealth()
{
	return maxHealth;
}

// Returns the maximum energy of the character.
int Character::getMaxEnergy()
{
	return maxEnergy;
}

// Adds health to the character when the
// player has used a health package.
void Character::addHealth(int addedHealth)
{
	// If the health is less than the
	// maximum health, add health.
	if (health < maxHealth)
		health += addedHealth;

	// If health goes over the maximum
	// health, set the health equal to
	// the maximum health.
```

```
	if (health > maxHealth)
		health = maxHealth;
}

// Adds energy to the character when the
// player has used an energy package.
void Character::addEnergy(int addedEnergy)
{
	// If the energy is less than the
	// maximum energy, add energy.
	if (energy < maxEnergy)
		energy += addedEnergy;

	// If the energy goes over the
	// maximum energy, set the energy equal
	// to the maximum energy.
	if (energy > maxEnergy)
		energy = maxEnergy;
}

// Decreases the energy of the character
// when the character has used a
// special attack.
void Character::decreaseEnergy(int usedEnergy)
{
	energy -= usedEnergy;
}

// The character defends himself
void Character::defend(int damage, int defense)
{
	// Take the damage off the defense.
	defense -= damage;

	// If the defense is less than or
	// equal to zero, take the remaining
	// damage off the health.
	if (defense <= 0)
		health += defense;

	// If the health of the character is less than
	// or equal to zero, set the health to 0 and
	// set the gameState to GAMEOVER.
	if (health <= 0)
	{
		health = 0;
		GameManager::getInstance()->setGameState(GameState::GAMEOVER);
	}
}
```

```
/*
 * Jose Sandoval
 * CIS-17C: C++ Programming
 * November 11, 2014
 * Project #1: Card Fight Game
 * Description: A fighting game that uses cards.
 */

#ifndef GUI_H
#define GUI_H

// System Libraries
#include <string>
#include <iostream>

// Header Files
#include "Hand.h"

using namespace std;

class GUI
{
private:
	char upperLeftCorner = 201;
	char upperRightCorner = 187;
	char lowerLeftCorner = 200;
	char lowerRightCorner = 188;
	char horizontal = 205;
	char vertical = 186;
	string graphic;
public:
	GUI();
	~GUI();
	string drawTopBox(int);
	string drawBottomBox(int);
	void drawStats();
	void drawChosenCards();
	void drawHandCards(deque<Card>*);
	void displayBanner();
};

#endif // GUI_H
```

```
/*
 * Jose Sandoval
 * CIS-17C: C++ Programming
 * November 11, 2014
 * Project #1: Card Fight Game
 * Description: A fighting game that uses cards.
 */

#include "GUI.h"
#include "GameManager.h"
```

```
GUI::GUI()
{
}

GUI::~GUI()
{
}

string GUI::drawTopBox(int horizontalSize)
{
	graphic = "";
	graphic += upperLeftCorner;
	for (int x = 0; x < horizontalSize; x++)
		graphic += horizontal;
	graphic += upperRightCorner;

	return graphic;
}

string GUI::drawBottomBox(int horizontalSize)
{
	graphic = "";
	graphic += lowerLeftCorner;
	for (int x = 0; x < horizontalSize; x++)
		graphic += horizontal;
	graphic += lowerRightCorner;

	return graphic;
}

void GUI::drawStats()
{
	cout << drawTopBox(46) << endl;

	for (int i = 1; i < 4; i++)
	{
		cout << char(186);
		cout << "     ";
		if (i == 1)
		{
			cout << GameManager::getInstance()->getPlayer()->getCharacter()-
>getName()
				<< "           "
				<< GameManager::getInstance()->getEnemy()->getCharacter()-
>getName();
		}
		if (i == 2)
		{
			cout << "Health: " << GameManager::getInstance()->getPlayer()-
>getCharacter()->getHealth()
				<< "            "
				<< "Health: " << GameManager::getInstance()->getEnemy()-
>getCharacter()->getHealth();
		}
```

```
			if (i == 3)
			{
				cout << "Energy: " << GameManager::getInstance()->getPlayer()-
>getCharacter()->getEnergy()
					<< "             "
					<< "Energy: " << GameManager::getInstance()->getEnemy()-
>getCharacter()->getEnergy();
			}
			cout << "       ";
			cout << char(186) << endl;
		}

		cout << drawBottomBox(46) << "\n\n";
}

void GUI::drawChosenCards()
{
		string output;

		output += "\nPlayer chose card:     ";
		output += "Enemy chose card:\n";

		for (int line = 0; line <= 12; line++)
		{
			if (line % 2 == 1)
				output += "\n     ";
			if (line % 2 == 0)
				output += "             ";

			if (line == 1 || line == 2)
			{
				output += drawTopBox(6);
				output += "   ";
			}
			if (line == 3 || line == 4 || line == 9 || line == 10)
			{
				output += char(186);
				output += "      ";
				output += char(186);
				output += "   ";
			}
			if (line == 5 || line == 6)
			{
				output += char(186);
				if (line == 5)
				{
					if (GameManager::getInstance()->getPlayer()-
>getChosenCard()->isItemCard())
						output += " Item";
					else
						output += " A: " +
to_string(GameManager::getInstance()
						->getPlayer()->getChosenCard()->getAttack());
				}
				else
```

```
                        {
                                if (GameManager::getInstance()->getEnemy()-
>getChosenCard()->isItemCard())
                                        output += " Item";
                                else
                                        output += " A: " +
to_string(GameManager::getInstance()->
                                        getEnemy()->getChosenCard()->getAttack());
                        }
                        output += " ";
                        output += char(186);
                        output += "  ";
                }
                if (line == 7 || line == 8)
                {
                        output += char(186);
                        if (line == 7)
                        {
                                if (GameManager::getInstance()->getPlayer()-
>getChosenCard()->isItemCard())
                                        output += " Card";
                                else
                                        output += " D: " +
to_string(GameManager::getInstance()
                                        ->getPlayer()->getChosenCard()->getDefense());
                        }
                        else
                        {
                                if (GameManager::getInstance()->getEnemy()-
>getChosenCard()->isItemCard())
                                        output += " Card";
                                else
                                        output += " D: " +
to_string(GameManager::getInstance()
                                        ->getEnemy()->getChosenCard()->getDefense());
                        }

                        output += " ";
                        output += char(186);
                        output += "  ";
                }
                if (line == 11 || line == 12)
                {
                        output += drawBottomBox(6);
                        output += "  ";
                }
        }

        cout << output << endl;
}

void GUI::drawHandCards(deque<Card>* hand)
{
        string output;
        int card = 0;
```

```
	for (int line = 1; line <= 42; line++)
	{
		if (line % 6 == 0)
			output += "\n";

		if (line < 6)
			output += "Card #" + to_string(line) + "    ";
		if (line > 6 && line < 12)
		{
			output += char(201);
			for (int lengthSize = 0; lengthSize < 6; lengthSize++)
				output += char(205);
			output += char(187);
			output += "  ";
		}
		if ((line > 12 && line < 18) || (line > 30 && line < 36))
		{
			output += char(186);
			output += "      ";
			output += char(186);
			output += "  ";
		}
		if (line > 18 && line < 24)
		{
			card++;
			output += char(186);

			if (hand->at(card - 1).isItemCard())
				output += " Item";
			else
				output += " A: " + to_string(hand->at(card -
1).getAttack());
			output += " ";
			output += char(186);
			output += "  ";
		}
		if (line > 24 && line < 30)
		{
			card++;
			output += char(186);
			if (hand->at(card - 6).isItemCard())
				output += " Card";
			else
				output += " D: " + to_string(hand->at(card -
6).getDefense());
			output += " ";
			output += char(186);
			output += "  ";
		}
		if (line > 36 && line < 42)
		{
			output += char(200);
			for (int lengthSize = 0; lengthSize < 6; lengthSize++)
				output += char(205);
```

```
                output += char(188);
                output += "  ";
            }
        }

        cout << output;
}

// Displays the game banner.
void GUI::displayBanner()
{
        cout <<
"================================================================================\n"
            << "                    ULTIMATE CARD FIGHT GAME\n"
            <<
"================================================================================\n" <<
endl;
}
```

```
/*
 * Jose Sandoval
 * CIS-17C: C++ Programming
 * November 11, 2014
 * Project #1: Card Fight Game
 * Description: A fighting game that uses cards.
 */

#ifndef ITEM_H
#define ITEM_H

#include <string>

using namespace std;

class Item
{
private:
	string type;
public:
	Item(int);
	~Item();
	string getType();
};

#endif // ITEM_H
```

```
/*
 * Jose Sandoval
 * CIS-17C: C++ Programming
 * November 11, 2014
 * Project #1: Card Fight Game
 * Description: A fighting game that uses cards.
 */

#include "Item.h"

// Overloaded constructor for Item class.
Item::Item(int itemType)
{
	switch (itemType)
	{
	case 1:
		type = "Special Attack";
		break;
	case 2:
		type = "Health Package";
		break;
	case 3:
		type = "Energy Package";
		break;
	case 4:
		type = "Cards Swap";
```

```
			break;
		default:
			break;
		}
}

// Destructor for Item class.
Item::~Item()
{
}

// Returns the type of the item.
string Item::getType()
{
		return type;
}
```

```
namespace GameState
{
		enum GameState
		{
			GAMEOVER,
			PLAYING
		};
}
```