**Project 2**

UNO v2

CSC 17a 43950

Nicholas Cardinal

5/28/2015

**Introduction**

Title: UNO!

UNO is a card game.

The objective of UNO is to place all of the cards down that are in your hand.

There are two modes in this game, one where you can play head to head against your friends and another where you can play against an AI!

The game UNO is played by placing a card of the same number or color of the top card

in the discard pile. You begin with a hand of seven cards and are forced to draw when you cannot place the correct card. There are also special action cards you can use against the other player including Draw2 which forces the other played to add two cards to their hand, another being Skip which skips your opponents turn and another being a wild card which can be placed and any time and it allows you to change the color of the card atop the discard pile. The game is played this way until one player has placed his last card and they become the winner.

You can enjoy this game for hours with a friend or alone by yourself playing against the AI!

Summary

Project size: 1073 lines.

The number of variable: 13 listed

I used many concepts that were covered over the course of this class including classes, copy constructors, reading and writing from binary files, dynamic memory allocation, virtual functions, pure virtual functions, abstract classes, class inheritance, exception handling, polymorphism, overloaded operators. I failed to find a use for a template class.

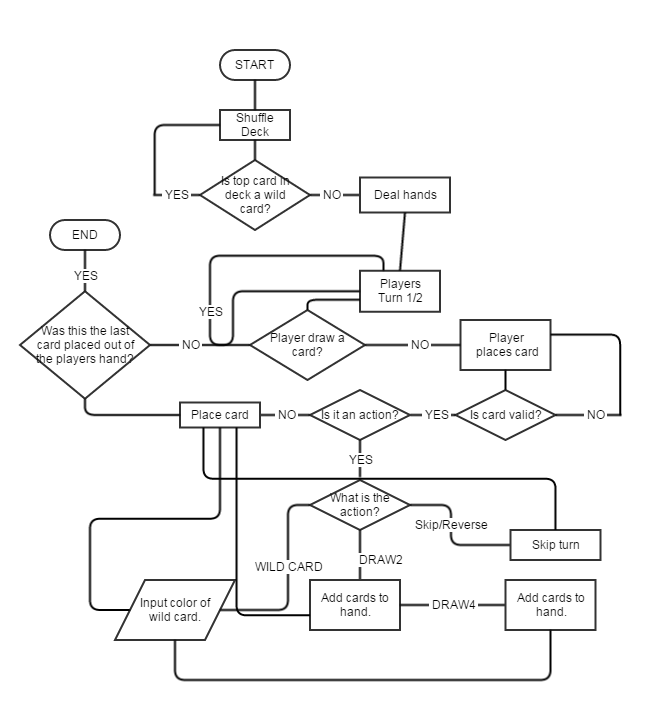
Description

I began by creating a class that holds the attributes of an UNO card including color, number, action and a bool value of wild. These attributes are used throughout many of the functions in order to compare cards, place cards and deal damage to the other player. I then created Deck class that is an array of Cards. I use this deck throughout the game with functions like draw card which removes a card from the deck and adds it to the players hand. Most of the games functionality is stored in the class titled “GameState” here is where I store the players individual hands, whose turn it is, and the contents of the discard pile. The GameState class handles everything from setting up the players hand, drawing cards, removing cards, and checking for various actions that can happen throughout the course of the game. The GameState class is the heart of the UNO game. The main game loop is run off of the GameState’s getWon function which is a function that checks whether either player has placed their last card, if one has won is set to true and the game ends. The game can be played against and AI which checks it’s hand for cards of the same type as the cards in the discard pile, it also places special action cards so that it behaves like a normal player would. It can also be played against a friend where the turn is incremented every time a player places a valid card.

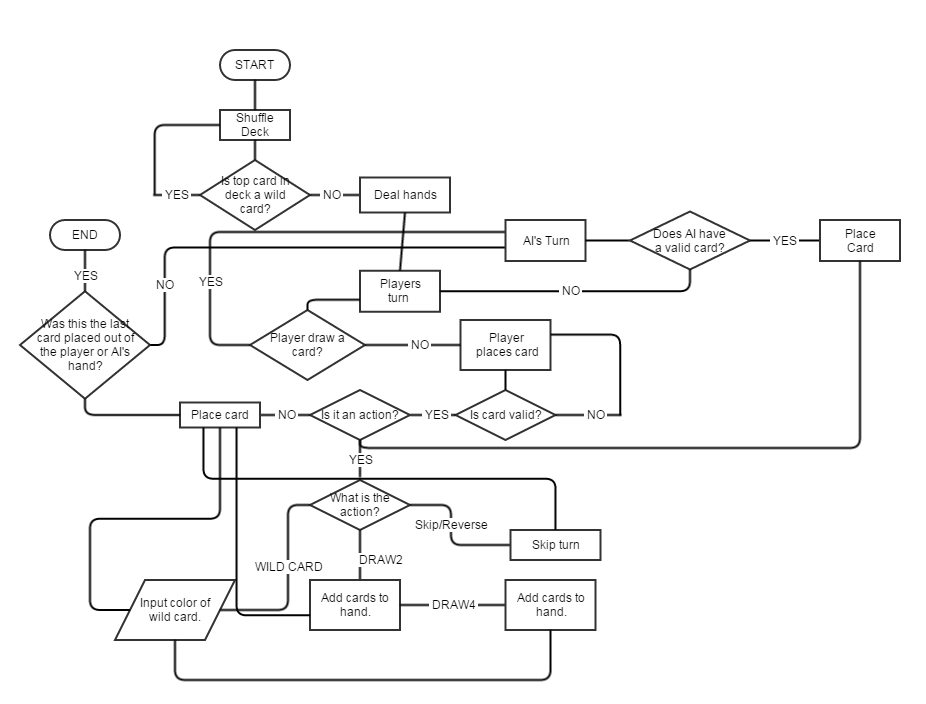
Variables

|  |  |  |  |
| --- | --- | --- | --- |
| Type | Name | Description | Location |
| Deck | cardDeck | Is an instance of the Deck class which stores an array of Cards that is used throughout the game to draw from. | Line 20, main. |
| GameState | UNO | This instance of the GameState class controls all functionality of the UNO game. | Line 19, main. |
| bool | accepted | is initialized to false but is set to true once the player placed a valid card. | Line 30, main. |
| Card\* | gameDeck | This is the internal instance of the deck accessed by cardDeck. | Line 14, Deck class. |
| Card\* | p1Hand, p2Hand | A pointer to the players individual hands. | Line 18,19, GameState class. |
| Card | dpPile | A instance of my Card structure to store the attributes of the card after it has been discarded. | Line 20, GameState class. |
| int | turn | Stores the players turn, is incremented and the players turn is checked by p % 2 being 0 or 1. | Line 16, GameState class. |
| bool | skipped | Is used to change the players turn if a skip or reverse card is placed. | Line 23, GameState class. |
| bool | won | Initialized to false, becomes true when one of the players places his last card. | Line 17, GameState class. |
| bool | isSkip | Initialized to false, is set to true when either the AI or the player place a skip card. | Line 335, GameState.cpp, AI function. |
| Card\* | phTemp | Based on whose turn it is this pointer is set to a players hand. | Used in many functions, GameState.cpp |
| Card \* | newHand | A dynamically allocated array used in as a copy when a player is forced to draw a card. | Line 201, GameState.cpp, addNewCard function. |
| fstream | file | Fstream declaration used to open the a file to store the name of the winner. | Line 538, GameState.cpp |

Player vs. Player Flowchart:

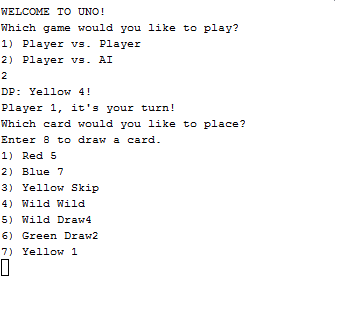


Player vs. AI Flowchart

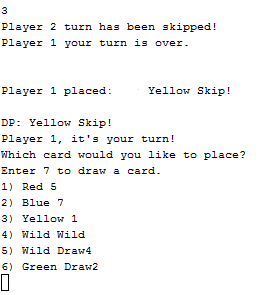


**Screen shots:**

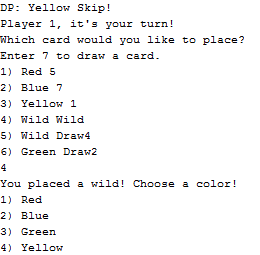
In the screenshot demo I’m only going to show the player vs. AI because it demonstrates the functionality of both types.



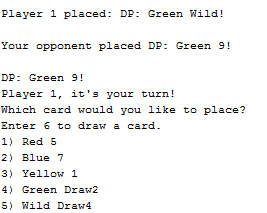
* The game begins by outputting the players hand and allowing them to pick a card, a card may be picked when it is the name color or number of that in the discard pile or “DP”.



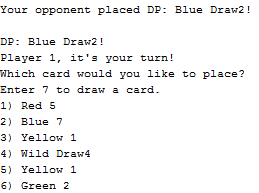
-This shot shows the functionality of the “Skip” card, my hand previously had the “Yellow Skip” in the 3rd place, I placed this card and it skipped the AI’s turn and returned to mine.



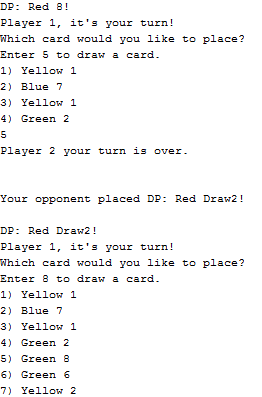
* In this shot I have placed card 4, a WILD, which allows me to change the color of the of the card on the discard pile.



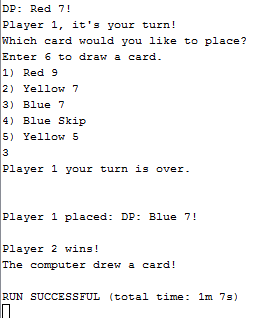
-I placed the wild card and selected the color green, it was then the AI’s turn and it placed a “Green 9” the Green 9 was then added to the discard pile and it again my turn.



-This shot shows the AI using a draw 2 card on me, where previously my hand had 4 cards in it now has 6 because the draw two card adds two cards to the opponent.



-This shot again demonstrates the use of the draw two card but also uses the draw function, I did not have a card to place so I used 5 to draw, the draw card button is always one higher than the size of the hand, I originally had 4 cards in my hand, I drew 1, the AI placed a draw 2 card which then made the size of my hand 7.



-In the end the AI placed its last card and was declared the winner and the program ends.

**Concepts**

**Memory allocation:**

This concept is used in my copy classes such as the addNewCard class declared on like 199.

This function reads in the players hand and dynamically allocates a new array of the size of the players hand originally plus one, the contents of the players hand is then copied in to the new array and then deleted. The new array now has one element more than the old array and that is where the card drawn from the deck is stored. The new hand is then reassigned to the players hand.

**Pointers with arrays and arrays of classes:**

This concept is illustrated mainly in my GameState and Deck classes, in these classes I define pointers to each players hands and a pointer to the deck used throughout the game. These pointers are used throughout the program to access the players hands and allow them to manipulate the deck.

**Reading and writing to binary files:**

Once the game is over, on line 283 the storeStats function is called passing in the name of the winner. The name is then written to a file where the user is then asked if they would like to see the stats from the previous game and if they choose to see them I read the winners name from the file and output how many turns it took them to win the game.

**Class Inheritance:**

The Player class is inherited from the abstract base class, Person. This allows the Player class to overwrite the pure virtual functions of the Person class so that it may use those functions to set the age and name of the player.

**Overloaded operators:**

I overloaded the stream operators for the Player class, it is used in lines 514 and 518 in the GameState.cpp file, the << operator is used to let the user know who the younger player is and to let them know who is starting the game.

**Copy constructors:**

In the Deck.h header file there is a copy constructor created on line 21. This constructor is used by passing in an instance of the Deck class and that deck is the assigned to the deck it was called on.

**Exception handling:**

When the function setAge() is call on line 507 in GameState.cpp the user is asked to enter their age, if the users input is anything but a number it will throw an exception telling the user that they have entered an invalid age and prompting them to enter again.

**Polymorphism: Virtual functions and Abstract Classes:**

The “Person.h.” class is a class that contains 4 pure virtual functions which are overridden by the “Player” class to store and retrieve the players name and age of each player during the game.

**Program:**

MAIN:

#include <cstdlib>

#include <iostream>

#include <ctime>

#include "Deck.h"

#include "Card.h"

#include "GameState.h"

#include "Player.h"

using namespace std;

int main(int argc, char\*\* argv) {

srand(time(0));

GameState UNO;

Deck cardDeck;

int choice = 0;

cardDeck.setUp();

cardDeck.shuffleDeck();

UNO.setHand(cardDeck);

UNO.setHand(cardDeck);

UNO.setDP(cardDeck.draw());

//UNO.nameSet();

choice = UNO.menu();

bool accepted = false; //Controls whether the card the user placed is accepted.

do{

if(choice == 1){

UNO.outputDP();

do{

accepted = UNO.checkCard(UNO.getHand(1),1,cardDeck);

}while(accepted == false);

UNO.outputDP();

accepted = false;

if(UNO.getWon() == false){

do{

accepted = UNO.checkCard(UNO.getHand(2),2,cardDeck);

}while(accepted == false);

}

accepted = false;

}

else{

UNO.setTurn(1);

UNO.outputDP();

do{

if(UNO.getSkip() == false){

accepted = UNO.checkCard(UNO.getHand(1),1,cardDeck);

UNO.setTurn(1);

}

else{

UNO.setSkip(false);

}

}while(accepted == false);

if(UNO.getSkip() == false){

UNO.AI(cardDeck);

}

else{

UNO.setSkip(false);

}

}

}while(UNO.getWon() == false);

return 0;

}

GameState.h

#include "Card.h"

#include "Deck.h"

#include "Player.h"

class GameState{

private:

int turn;

bool won;

Card \*p1Hand;

Card \*p2Hand;

Card dpPile[1];

int p1Size;

int p2Size;

bool skipped;

public:

//Constructor

GameState();

//Destructor

virtual ~GameState();

//Mutators/Accessors

bool getWon();

int getTurn();

void setTurn(int);

void setWon();

void setHand(Deck&);

Card\* getHand(int);

Card getDP();

void setDP(Card);

int getSize(Card\*);

//Game functionality.

void outputDP();

int cardChoice(Player,Player);

bool checkCard(Deck&,Player,Player);

void removeCard(int&,int,bool,Player, Player);

bool addNewCard(Deck&,bool,Player,Player);

bool checkAction(Deck&,int,Player,Player);

void checkWin(Player,Player);

void checkWild();

bool checkSkip(Player,Player);

void checkDrawAct(Deck&,Player,Player);

//AI Functions

void AI(Deck&);

int checkAI();

void AIDraw(Deck&);

void AIRemove(int);

void checkAIWild();

void checkDrawAI(Deck&);

void setSkip(bool);

bool getSkip();

//

bool operator==(const Card\*);

bool operator!=(const Card\*);

//File IO / Try-Catch function.

void setUp(Player&, Player&);

void nameSet();

void storeStats(Player);

//Display stats.

void statsOut(Player);

//Asks user if they want to check stats.

void checkStats(Player);

//Menu

int menu();

};

GameState.cpp

#include "Card.h"

#include "Deck.h"

#include "Player.h"

class GameState{

private:

int p;

bool won;

Card \*p1Hand;

Card \*p2Hand;

Card dpPile[1];

int p1Size;

int p2Size;

bool skipped;

public:

//Constructor

GameState();

//Destructor

virtual ~GameState();

//Mutators/Accessors

bool getWon();

int getTurn();

void setTurn(int);

void setWon();

void setHand(Deck&);

Card\* getHand(int);

Card getDP();

void setDP(Card);

int getSize(Card\*);

//Game functionality.

void outputDP();

int cardChoice();

bool checkCard(Card\*,int,Deck&);

void removeCard(Card\*,int&,int,int,bool);

bool addNewCard(int,Deck&,bool);

bool checkAction(Deck&,int);

void checkWin();

void checkWild();

bool checkSkip();

void checkDrawAct(Deck&);

//AI Functions

void AI(Deck&);

int checkAI();

void AIDraw(Deck&);

void AIRemove(int);

void checkAIWild();

void checkDrawAI(Deck&);

void setSkip(bool);

bool getSkip();

//Exception function for the try catch.

void exception(int);

//

bool operator==(const Card\*);

bool operator!=(const Card\*);

//File IO / Try-Catch function.

void nameSet();

//Menu

int menu();

};

GameState.cpp

#include <cstdlib>

#include <iostream>

#include <fstream>

#include "GameState.h"

#include "Deck.h"

#include "Card.h"

#include "Player.h"

using namespace std;

enum Action { blank, DRAW2, DRAW4, WILD, SKIP, REVERSE };

GameState::GameState(){

p1Size = 7;

p2Size = 7;

p1Hand = new Card[p1Size];

p2Hand = new Card[p2Size];

turn = 1;

won = false;

skipped = false;

}

GameState::~GameState(){

delete [] p1Hand;

delete [] p2Hand;

}

void GameState::setHand(Deck& gameDeck){

//Sets up player 1's hand.

for(int i = 0; i < 7; i++){

p1Hand[i] = gameDeck.draw();

}

//Sets up player 2's hand.

for(int i = 0; i < 7; i++){

p2Hand[i] = gameDeck.draw();

}

}

Card\* GameState::getHand(int p){

if(p==1)

return p1Hand;

if(p==2)

return p2Hand;

}

Card GameState::getDP(){

return dpPile[0];

}

void GameState::setDP(Card c){

//Sets card to DP.

dpPile[0].setColor(c.getColor());

dpPile[0].setNum(c.getNum());

dpPile[0].setIntAction(c.getIntAction());

dpPile[0].setStrAction(c.getStrAction());

dpPile[0].setWild(c.getWild());

}

void GameState::outputDP(){

//Used for outputting the discard pile to the user.

if(dpPile[0].getNum() < 0){

cout << "DP: " << dpPile[0].getColor() << " " << dpPile[0].getStrAction() << "!" << endl;

}

else{

cout << "DP: " << dpPile[0].getColor() << " " << dpPile[0].getNum() << "!" << endl;

}

}

int GameState::cardChoice(Player p,Player p2){

int choice = 0;

int size = 0;

Card\* phTemp;

//If it's player one's turn.

if(turn % 2 == 1){

size = p1Size;

phTemp = p1Hand;

cout << p.getName() << " it's your turn!" << endl;

}

//If it's player 2's turn.

if(turn % 2 == 0){

size = p2Size;

phTemp = p2Hand;

cout << p2.getName() << " it's your turn!" << endl;

}

cout << "Which card would you like to place?" << endl;

cout << "Enter " << size+1 << " to draw a card." << endl;

//Outputs the hand for the user to select.

for(int i=0;i<size;i++){

if(phTemp[i].getNum() < 0){

cout << i+1 << ") " << phTemp[i].getColor() << " " << phTemp[i].getStrAction() << endl;

}

else{

cout << i+1 << ") " << phTemp[i].getColor() << " " << phTemp[i].getNum() << endl;

}

}

do{

while(!(cin >> choice)){

cout << "Please choose a valid card!" << endl;

cin.clear();

cin.ignore();

}

}while(choice > size+1);

//If the players choice was to draw a card.

if(choice == size+1){

return 109;

}

//If the player entered an invalid card.

if(choice < 1){

return -1;

}

//Returns the choice the player made if it is valid.

else{

return choice;

}

}

bool GameState::getWon(){

return won;

}

bool GameState::checkCard(Deck& d,Player p, Player p2){

setSkip(false);

//Controls the card that the user picks.

int choice = cardChoice(p,p2);

if(choice < 0){

choice = cardChoice(p,p2);

}

bool draw = false;

if(choice == 109){

//If the user drew a card.

draw = addNewCard(d,false,p,p2);

return draw;

}

Card\* phTemp;

int tempSize = 0;

//If player 1's turn

if(turn % 2 == 1){

phTemp = p1Hand; //Temp stores the player 1's hand.

tempSize = p1Size; //Temp stores the player 2's hand size.

}

//If Player 2's turn.

if(turn % 2 == 0){

phTemp = p2Hand; //Temp stores the player 2's hand.

tempSize=p2Size; //Temp stores the player 2's hand size.

}

bool isSkip = false; //Is true if the card placed is an skip or a reverse.

//Checks if the card placed is valid.

if(phTemp[choice-1].getColor() == dpPile[0].getColor() || phTemp[choice-1].getNum() == dpPile[0].getNum()

|| phTemp[choice-1].getWild() == true){

//Adds the card to the discard pile.

setDP(phTemp[choice-1]);

//Checks if the card is a skip card.

isSkip = checkAction(d,0,p,p2);

//Removes the card from the players hand.

removeCard(tempSize,choice,isSkip,p,p2);

return true;

}

else{

cout << "Please choose a card of the same number, color or a wild" << endl;

return false;

}

}

void GameState::removeCard(int &size, int choice, bool a,Player p, Player p2){

Card\* phTemp;

//If player one's turn.

if(turn % 2 == 1){

phTemp = p1Hand; //Temp stores the player 1's hand.

}

//If player two's turn.

if(turn % 2 == 0){

phTemp = p2Hand; //Temp stores the player 2's hand

}

//Swaps the card chosen with the last card and then the card is deleted.

swap(phTemp[choice-1], phTemp[size-1]);

if(turn % 2 == 1){

p1Size--;

p1Hand = phTemp;

}

if(turn % 2 == 0){

p2Size--;

p2Hand = phTemp;

}

if(turn%2 == 0){

cout << p2.getName() << " your turn is over." << endl << endl << endl;

cout << p2.getName() << " placed: "; outputDP(); cout<<endl;

}

else{

cout << p.getName() << " your turn is over." << endl << endl << endl;

cout << p.getName() << " placed: "; outputDP(); cout<<endl;

}

//Increments to the next players turn.

if(a == true){

turn++;

}

turn++;

checkWin(p,p2);

}

int GameState::getSize(Card\* ph){

int size = 0;

if(ph == p1Hand){

size = p1Size;

}

if(ph == p2Hand){

size = p2Size;

}

return size;

}

bool GameState::addNewCard(Deck& d, bool draw, Player p, Player p2){

if(turn % 2 == 1){

//Allocates a new card array with one extra space.

Card\* newHand = new Card[p1Size+1];

//Copies contents of the old hand to the new.

for(int i = 0; i < p1Size; i++){

newHand[i] = p1Hand[i];

}

newHand[p1Size] = d.draw();

p1Size++;

//Deallocated the memory of the old hand.

delete [] p1Hand;

//Sets the new hand equal to the old hand.

p1Hand = newHand;

if(draw == false){

cout << p.getName() << " your turn is over." << endl << endl << endl;

turn++;

}

}

else if(turn % 2 == 0){

Card\* newHand = new Card[p2Size+1];

for(int i = 0; i < p2Size; i++){

newHand[i] = p2Hand[i];

}

newHand[p2Size] = d.draw();

p2Size++;

delete [] p2Hand;

p2Hand = newHand;

if(draw == false){

cout << p2.getName() << " your turn is over." << endl << endl << endl;

turn++;

}

}

return true;

}

bool GameState::checkAction(Deck& d,int ai,Player p, Player p2){

Player dummy;

if(ai==0){

checkWild();

}

else{

checkAIWild();

}

bool isSkip = checkSkip(p,p2);

if(isSkip == true){

return true;

}

if(ai==0){

checkDrawAct(d,dummy,dummy);

}

else{

checkDrawAI(d);

}

return false;

}

void GameState::checkWin(Player p, Player p2){

if(p1Size == 0){

won = true;

cout << p.getName() << " wins!" << endl;

storeStats(p);

checkStats(p);

}

if(p2Size == 0){

won = true;

cout << p2.getName() << " wins!" << endl;

storeStats(p2);

checkStats(p);

}

}

void GameState::checkWild(){

//If it is a wild card the user is prompted to enter a color.

if(dpPile[0].getWild() == true){

int pColor;

string pColors;

cout << "You placed a wild! Choose a color!" << endl;

cout << "1) Red " << endl << "2) Blue " << endl << "3) Green " << endl

<< "4) Yellow" << endl;

cin >> pColor;

while(pColor < 0 && pColor > 4){

cout << "Please choose a valid number." << endl;

cin >> pColor;

}

if(pColor==1){pColors = "Red";}

else if(pColor==2){pColors = "Blue";}

else if(pColor==3){pColors = "Green";}

else{pColors == "Yellow";}

dpPile[0].setColor(pColors);

}

}

bool GameState::checkSkip(Player p, Player p2){

//If the card is a skip then isSkip is set to true allowing for the turn to be skipped.

if(dpPile[0].getIntAction()== SKIP || dpPile[0].getIntAction() == REVERSE){

if(turn % 2 == 0){

cout << p.getName() << "'s turn has been skipped!" << endl;

}

else{

cout << p2.getName() << "'s turn has been skipped!" << endl;

}

setSkip(true);

return true;

}

return false;

}

void GameState::checkDrawAct(Deck& d, Player p, Player p2){

bool draw = false;

//Adds 2 new cards to the hand if true.

if(dpPile[0].getIntAction() == DRAW2){

for(int i = 0; i<2;i++){

draw = true;

turn++;

addNewCard(d,draw,p,p2);

turn--;

}

}

//Adds 4 new cards to the hand if true.

if(dpPile[0].getIntAction() == DRAW4){

for(int i = 0; i<4;i++){

draw = true;

turn++;

addNewCard(d,draw,p,p2);

turn--;

}

}

draw = false;

}

int GameState::getTurn(){

return turn;

}

void GameState::setTurn(int t){

turn = t;

}

void GameState::setSkip(bool s){

skipped = s;

}

bool GameState::getSkip(){

return skipped;

}

void GameState::AI(Deck& d){

Player dummy;

//Controls all aspects of the AI's turn.

setSkip(false);

int choice;

bool draw = false;

bool isSkip = false;

choice = checkAI();

if(choice < 0){

AIDraw(d);

draw = true;

}

if(choice >= 0){

setDP(p2Hand[choice]);

cout << "The AI placed "; outputDP(); cout<<endl;

cout << "The AI now has " << p2Size << " cards" << endl << endl;

isSkip = checkAction(d,1,dummy,dummy);

AIRemove(choice);

}

}

int GameState::checkAI(){

int choice = 0;

bool found = false;

//Runs through AI's hand and chooses the first card able to be placed.

for(int i = 0; i < p2Size;i++){

if(p2Hand[i].getColor() == dpPile[0].getColor() || p2Hand[i].getNum() == dpPile[0].getNum()

|| p2Hand[i].getWild() == true){

choice = i;

found = true;

if(found == true){

break;

}

}

}

if(found == false){

cout << "The computer drew a card!" << endl;

return -1;

}

return choice;

}

void GameState::AIDraw(Deck& d){

//Draws a card for the AI.

Card\* newHand = new Card[p2Size+1];

for(int i = 0; i < p2Size; i++){

newHand[i] = p2Hand[i];

}

newHand[p2Size] = d.draw();

p2Size++;

delete [] p2Hand;

p2Hand = newHand;

}

void GameState::AIRemove(int choice){

//Removed a card from the AI's hand.

swap(p2Hand[choice], p2Hand[p2Size-1]);

p2Size--;

}

void GameState::checkAIWild(){

//Runs through the AI's hand and the wild card is chosen by the most

//frequently occurring color and sets the wild to that.

int red = 0;

int blue = 0;

int yellow = 0;

int green = 0;

if(dpPile[0].getWild() == true){

for(int i=0;i<p2Size;i++){

if(p2Hand[i].getColor() == "Red"){

red++;

}

else if(p2Hand[i].getColor() == "Blue"){

blue++;

}

else if(p2Hand[i].getColor() == "Yellow"){

yellow++;

}

else if(p2Hand[i].getColor() == "Green"){

green++;

}

}

if(red>blue&&red>yellow&&red>green){

dpPile[0].setColor("Red");

}

else if(blue>red&&blue>yellow&&blue>green){

dpPile[0].setColor("Blue");

}

else if(yellow>red&&yellow>blue&&yellow>green){

dpPile[0].setColor("Yellow");

}

else{

dpPile[0].setColor("Green");

}

}

}

void GameState::checkDrawAI(Deck& d){

bool draw = false;

//If the AI places a draw 2 it adds 2 cards to p1's hand.

if(dpPile[0].getIntAction() == DRAW2){

for(int i = 0; i < 2; i++){

Card\* newHand = new Card[p1Size+1];

for(int i = 0; i < p1Size; i++){

newHand[i] = p1Hand[i];

}

newHand[p1Size] = d.draw();

p1Size++;

delete [] p1Hand;

p1Hand = newHand;

}

}

//If the AI places a draw 4 it adds 4 cards to p1's hand.

if(dpPile[0].getIntAction() == DRAW4){

for(int i = 0; i < 4; i++){

Card\* newHand = new Card[p1Size+1];

for(int i = 0; i < p1Size; i++){

newHand[i] = p1Hand[i];

}

newHand[p1Size] = d.draw();

p1Size++;

delete [] p1Hand;

p1Hand = newHand;

}

}

}

int GameState::menu(){

int choice;

cout << "WELCOME TO UNO!" << endl;

cout << "Which game would you like to play?" << endl;

cout << "1) Player vs. Player" << endl;

cout << "2) Player vs. AI" << endl;

cin >> choice;

if(choice > 2 || choice < 0){

cout << "Please choose a valid game!" << endl;

cin >> choice;

}

return choice;

}

void GameState::setUp(Player &p, Player &p2){

cout << "Welcome Player 1." << endl;

p.setName();

p.setAge();

cout << endl<<endl;

cout << "Welcome Player 2." << endl;

p2.setName();

p2.setAge();

if(p.getAge() < p2.getAge()){

cout << p;

turn = 1;

}

else{

cout << p2;

turn = 2;

}

}

void GameState::storeStats(Player p){

fstream file("stats.bin",ios::binary | ios::in | ios::out | ios::trunc);

if(!file.is\_open()){

cout << "Error opening file" << endl;

}

else{

file.write((char\*)&p, sizeof(Player));

file.seekg(0);

}

file.close();

}

void GameState::statsOut(Player winner){

fstream file("stats.bin",ios::binary | ios::in | ios::out);

if(!file.is\_open()){

cout << "Error opening file" << endl;

}

else{

file.read((char\*)&winner, sizeof(Player));

}

cout << winner.getName() << " won in " << turn << " turns!" << endl;

file.close();

}

void GameState::checkStats(Player p){

char answer;

cout << "Would you like to see see the game stats? [y/n]" << endl;

cin >> answer;

if(!cin){

cout << "Invalid answer!" << endl;

checkStats(p);

}

else if(tolower(answer) == 'y'){

statsOut(p);

}

}

Deck class:

#include "Card.h"

#include <string>

class Deck{

private:

Card \*gameDeck;

void createDeck(std::string);

void addWilds();

int size;

public:

Deck();

virtual ~Deck();

void setUp();

void setUpHand();

void shuffleDeck();

void outputDeck();

virtual Card draw();

};

Deck.cpp:

#include <cstdlib>

#include <iostream>

#include "Deck.h"

#include "Card.h"

using namespace std;

enum Action { blank, DRAW2, DRAW4, WILD, SKIP, REVERSE };

enum Color { RED, GREEN, BLUE, YELLOW };

Deck::Deck(){

size = 109;

gameDeck = new Card[size];

for(int i = 0;i<105;i++){

gameDeck[i].setColor("NA");

gameDeck[i].setIntAction(-3);

gameDeck[i].setWild(false);

gameDeck[i].setStrAction("N/A");

gameDeck[i].setNum(-5);

}

}

Deck::~Deck(){

delete [] gameDeck;

}

void Deck::setUp(){

createDeck("Red");

createDeck("Green");

createDeck("Yellow");

createDeck("Blue");

addWilds();

}

void Deck::outputDeck(){

cout << "Default deck " << endl;

for(int i=0;i<108;i++){

if(gameDeck[i].getNum() < 0){

cout << i << "Card Value: " << gameDeck[i].getStrAction();

}

else{

cout << i << "Card Value: " << gameDeck[i].getNum();

}

cout << " Color: " << gameDeck[i].getColor();

cout << endl;

}

}

void Deck::createDeck(string color){

int cNum = 1;

int end = 0;

int i = 0;

if(color == "Red"){

i = 0;

end = 25;

}

if(color == "Green"){

i = 25;

end = 50;

}

if(color == "Yellow"){

i = 50;

end = 75;

}

if(color == "Blue"){

i = 75;

end = 100;

}

for(i;i<end;i++){

gameDeck[i].setColor(color);

gameDeck[i].setNum(cNum);

gameDeck[i].setWild(false);

gameDeck[i].setIntAction(0);

cNum++; //Increases from 1 - 9 then resets if cNum == 10;

if(cNum == 10){

cNum = 1;

}

if(i==end-7){

gameDeck[i].setStrAction("Reverse");

gameDeck[i].setIntAction(REVERSE);

gameDeck[i].setColor(color);

gameDeck[i+1].setStrAction("Reverse");

gameDeck[i+1].setIntAction(REVERSE);

gameDeck[i+1].setColor(color);

gameDeck[i+2].setStrAction("Skip");

gameDeck[i+2].setIntAction(SKIP);

gameDeck[i+2].setColor(color);

gameDeck[i+3].setStrAction("Skip");

gameDeck[i+3].setIntAction(SKIP);

gameDeck[i+3].setColor(color);

gameDeck[i+4].setStrAction("Draw2");

gameDeck[i+4].setIntAction(DRAW2);

gameDeck[i+4].setColor(color);

gameDeck[i+5].setStrAction("Draw2");

gameDeck[i+5].setIntAction(DRAW2);

gameDeck[i+5].setColor(color);

gameDeck[i+6].setNum(0);

gameDeck[i+6].setColor(color);

break;

}

}

}

void Deck::addWilds(){

for(int i = 100;i<105;i++){

gameDeck[i].setWild(true);

gameDeck[i].setColor("Wild");

gameDeck[i].setStrAction("Wild");

gameDeck[i].setIntAction(WILD);

}

for(int i=104;i<109;i++){

gameDeck[i].setWild(true);

gameDeck[i].setColor("Wild");

gameDeck[i].setStrAction("Draw4");

gameDeck[i].setIntAction(DRAW4);

}

}

Card Deck::draw(){

Card\* newDeck = new Card[size-1];

for(int i = 1; i<size-1;i++){

newDeck[i-1] = gameDeck[i];

}

Card ret = gameDeck[0];

//Deletes the old deck.

delete [] gameDeck;

gameDeck = newDeck;

return ret;

}

void Deck::shuffleDeck(){

int shuffle = 0;

for(int i=0;i<size;i++){

shuffle = rand() % 108 + 0;

swap(gameDeck[i],gameDeck[shuffle]);

}

}

Card class:

class Card{

private:

std::string color;

int number;

int action;

bool wild;

std::string strAction;

public:

Card();

std::string getColor(){return color;}

std::string getStrAction(){return strAction;}

int getIntAction(){return action;}

int getNum(){return number;}

bool getWild(){return wild;}

void setColor(std::string c) {color = c;}

void setStrAction(std::string sA){strAction = sA;}

void setIntAction(int iA){action = iA;}

void setNum(int n){number = n;}

void setWild(bool w){wild = w;}

//Card& operator= (const Card& c);

};

Card.cpp

Card::Card(){

color = "NC";

number = -3;

action = -5;

wild = false;

strAction = "NSA";

}

Person.h

class Person{

protected:

std::string name;

int age;

public:

//Declared virtual in order to be overridden by the Player class.

virtual std::string getName() = 0;

virtual void setName() = 0;

virtual void setAge() = 0;

virtual int getAge() = 0;

};

Player.h

class Player : protected Person{

private:

public:

Player(){

name = " ";

age = 0;

}

Player(std::string, int);

virtual std::string getName();

virtual void setName();

virtual void setAge();

virtual int getAge();

void exception(int);

friend std::ostream &operator<<(std::ostream& out, Player& p);

friend std::istream &operator>>(std::istream& in, Player& p);

};

Player.cpp

void Player::exception(int ex){

if(ex == 3){

cout << "Invalid age, ERROR MESSAGE: " << ex << endl;

}

cout << "Enter an age";

cin.clear();

cin.ignore();

setAge();

}

Player::Player(string n, int a){

name = n;

age = a;

}

string Player::getName(){

return name;

}

void Player::setName(){

string n;

cout << "Enter your name: ";

cin >> n;

name = n;

cout << "\nWelcome " << name << "!" << endl;

}

void Player::setAge(){

int a;

cout << "Enter your age: ";

cin >> a;

try{

if(!cin){

throw 3;

}

else{

age = a;

}

}catch(int ex){

exception(ex);

cin.clear();

}

}

int Player::getAge(){

return age;

}

ostream &operator<<(ostream &out, Player &p){

out << p.name << " is younger and will go first!" << endl<<endl<<endl;

return out;

}

istream &operator>>(istream &in, Player &p){

in >> p.name >> p.age;

return in;

}