**GWENT**

**CIS-17a 42636**

**Name: Eric Diaz**

**Date due: 4/17/17**

**Introduction**

Title: Gwent, the card game

The project is based off a mini-game in the video game “The Witcher 3.” The point of the game is to draw 10 cards, each with their own damage value. In order to win the game, you have to win 2 rounds of 3. To win a round, you have to place cards from your hand to the playing field and the player with the most damage wins. Careful about placing all your cards in the very beginning. If you have no cards in the second or third rounds, you lose.

This game is a great introduction for myself, as a student, to how structures, classes, and binary files work. So far, it has been a lot of fun programming this project.

**Summary**

Project size: about 500 lines of code.

Number of variables: About 6, where the structure contains the majority of data.

Number of different functions: Around 10 separate functions.

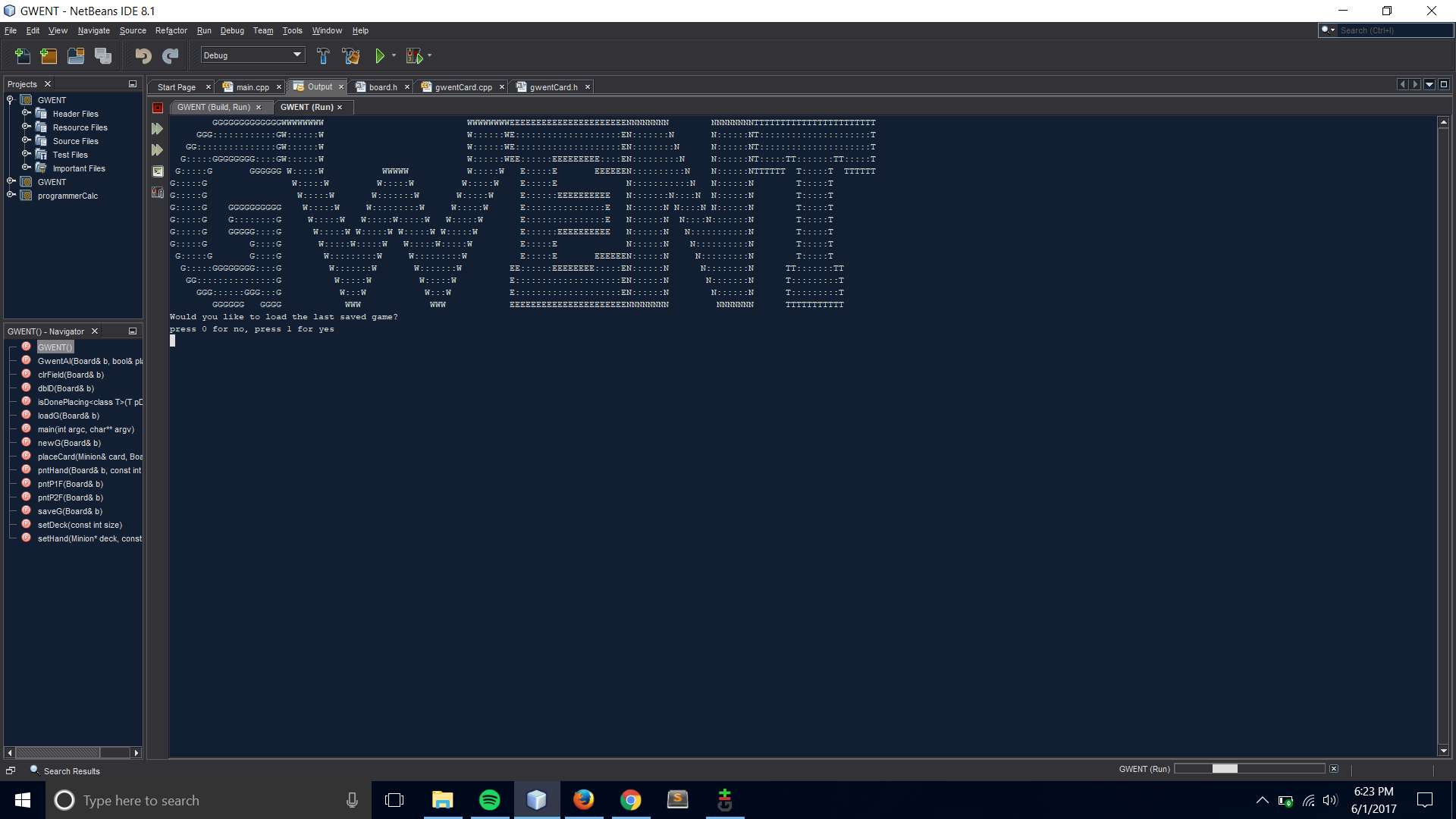
This project contains most of the concepts learned from csc 5 and csc 17a up to this point. It has dynamic arrays of structures, a 3-dimensional array of structures to contain the state of the game at separate points in time, I’ll read and/or write from binary and text files, I return pointers to class objects to structure members, load and save games from binary files.

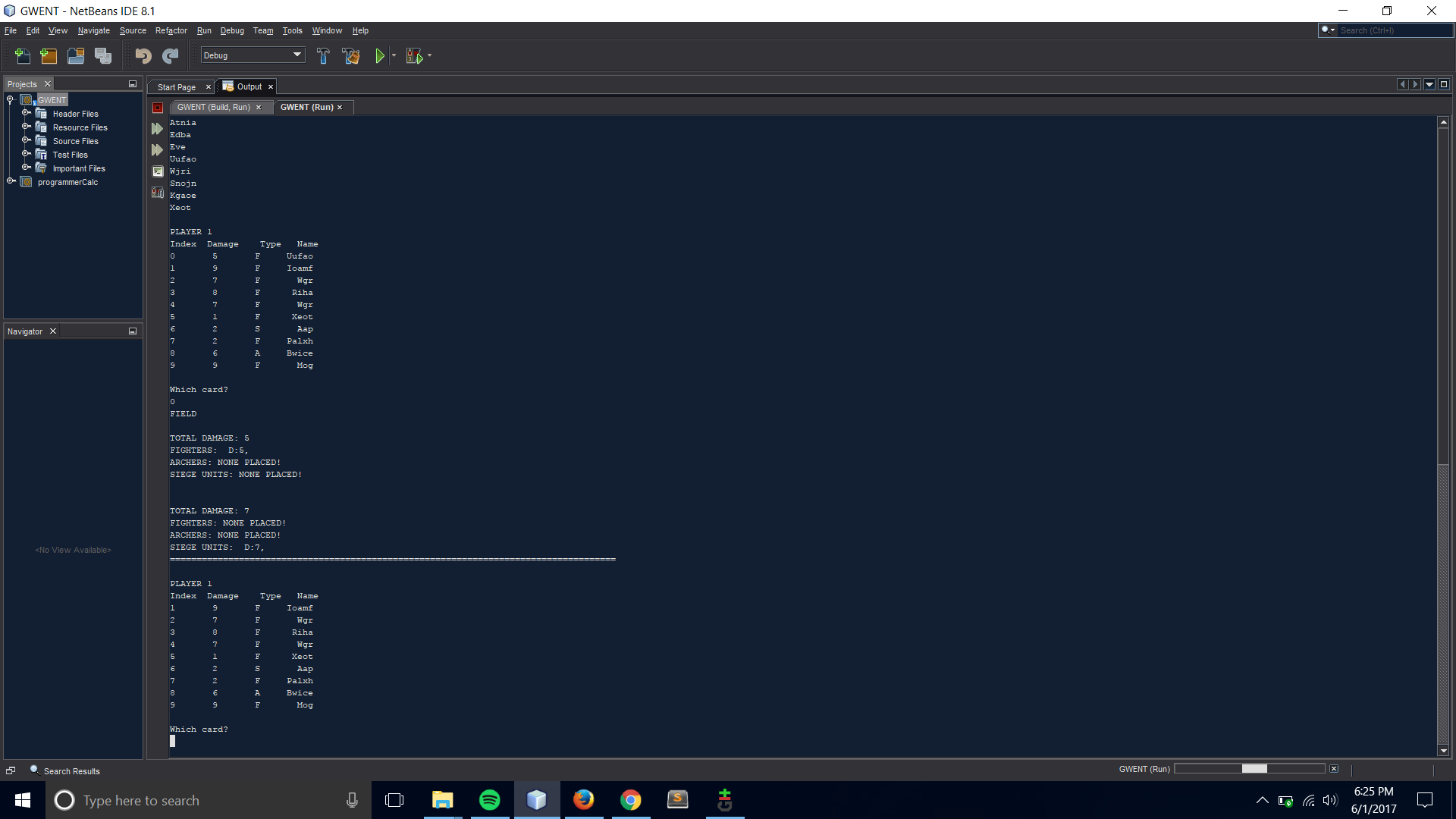
The total time it took me to actually program the game was close to 21 hours. I’ve kept track of when I started and when I stopped coding, I did this project in about three 7-hour shifts right before spring break started and I was not able to do any work on this project during spring break because I was without internet, computers, or even cell signal. Something that stuck out to me is that I know that I cannot write a paper for more than a couple hours at a time, however I can write code for more than 5 hours and still not burn myself out with the work load which is very reassuring that I am majoring in the right field.

There are a few features that I’d like to implement such as unique cards with special abilities, a cleaner output of the card game, a smarter AI that takes into account of how many cards have been placed, how much damage the player has done, choosing cards that are more deliberate than random picks and find a way to fix a couple of bugs in my code. One of the major bugs I have found is that when I load a game from a binary file, I cannot delete the dynamic pointers without having the game crash right before it’s supposed to end. The game is completely playable, it just says “run failed” right before it’s supposed to return 0. I’ll have to do more reading on how structures relate to dynamic pointers.

**Sample I/O**

The “Home Screen” of GWENT.





The sample IO screen shot shows 1 move after the beginning of the game. At this point in the game, the player placed 1 card from their hand at index 0 onto the field where the computer makes a move immediately after. The hand shows the index of the card, the damage of each card, the type of card, and the name of the card. At this point in time, the field has 2 cards placed, 1 on each side of the field. The player has 1 card dealing 5 damage placed at the FIGHTER.

lane of the field and the computer has 1 card placed at the SIEGE lane of the field dealing 7 damage. The very top of the screen shot shows the player hand at turn 0, the field is in the center, and at the very bottom, it shows the player hand at turn 1. The card at index 0 is no longer in the hand at this point and is instead played in the field.

**The Card and Minion Classes**

The Card class is the basic object that other objects were supposed to inherit from. Currently, at this point in time, the Minion class is the only thing that inherits from the Card class where I had plans to add the Weather class as another child of the Card class. The card class has basic data in it such as the name of the card, the damage, index, and if it has been used. It also has a function “virtual void setName();” that is overridden by the Minion class and would have also been overridden by the Weather class. The Minion class also has an operator (+) method that when added with another Minion, the damage of both are added which is used to calculate the total damage on the field.

**Major Variables**

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **Variable Name** | **Description** | **Location** |
| board //struct | b | Current state of the game | Main() |
| board array | clock | Stores each turn of the current round | main() |
| Card array | cClock | Stores the state of each players hand per turn | main() |
| Minion array | b.p1Hand | Stores each players hand at a single point in time | main() |

**C++ Constructs**

|  |  |  |  |
| --- | --- | --- | --- |
| Chapter | Section | Topic | Location |
| Chapter 2 | Section 2 | cout | Everywhere |
| 2 | 4 | Variables | Main |
| 2 | 5 | Identifies | Main |
| 2 | 6 | Integers | Main; cnt |
| 2 | 8 | Strings | newG(b); b.p1Name |
| 2 | 10 | bool | main(); bool again |
| 2 | 11 | sizeof | saveG(b); |
| 2 | 13 | scope | main |
| 2 | 14 | Arithmetic | For Loops |
| 2 | 15 | comments | main |
| 2 | 16 | Named constants | \*setHand(card \*, const int); |
| 3 | 1 | cin | main |
| 3 | 2 | Math Expression? | ???? |
| 3 | 3 | Mixing datatypes | saveG(&b); |
| 3 | 4 | Overflow underflow | ?????? |
| 3 | 5 | TypeCasting | saveG(&b); |
| 3 | 6 | Multiple assignment | ?????? |
| 3 | 7 | Formatting output | pntHand() |
| 3 | 9 | Math library | -------------- |

|  |  |  |  |
| --- | --- | --- | --- |
| 4 | 1 | Relational operators | main |
| 4 | 2 | if | main |
| 4 | 3 | if-else | main |
| 4 | 5 | nesting | main |
| 4 | 6 | If else if | main |
| 4 | 8 | Logical operators | main |
| 4 | 11 | Validating user input | main |
| 4 | 13 | Conditional operator | main |
| 4 | 14 | switch | gwentAI() |
| 5 | 1 | increment/decrement | main |
| 5 | 2 | while | Main |
| 5 | 5 | do-while | main |
| 5 | 6 | For loop | main |
| 5 | 11 | File input/output | saveG()/loadG(); |
| 6 | 3 | prototypes | Before main |
| 6 | 4 | Pass by value | setDeck() |
| 6 | 8 | Returning values | setDeck() |
| 6 | 9 | Boolean return values | ---------- |
| 6 | 11 | Static local variable | main |
| 6 | 12 | Default arguments | -------- |

|  |  |  |  |
| --- | --- | --- | --- |
| 6 | 13 | Reference variables | saveG(&b); |
| 6 | 14 | Overloading functions | ---------- |
| 7 | 4 | Array initialization | setHand() |
| 7 | 7 | Parallel arrays | Main ~line 96 |
| 7 | 8 | Arrays as function arguments | setHand |
| 7 | 9 | 2D arrays | I have a 3D array in main |
| 7 | 12 | Vectors | ---------------- |
| 8 | 1 | Search function | ----------------- |
| 8 | 2 | sort | -------------------------- |

|  |  |
| --- | --- |
| Topic | Location |
| Memory Allocation | setDeck() |
| Functions with structures as input and output | setDeck()//returns a class? Does that work?  saveG(b); takes struct as input |
| Arrays of structures | Found in board structure, arrays of Cards  And found in main  Card cClock[rounds][player][size]; |
| Read/Write to binary file | saveG(b);  loadG(b); |
| template | isDonePlacing(T, T); |
| Operator overload | Void dblD(Board &b); |
| Polymorphism | Classes Minion and Card |
| Inheritance | Classes Minion and Card. |

**Program**

#include <iostream>

#include <ctime>

#include <cstdlib>

#include <string>

#include <fstream>

#include "board.h"

#include "gwentCard.h"

using namespace std;

Card \*setDeck(int);

Card \*setHand(Card \*, int);

void placeCard(Card &, board &);

void pntHand(board &, int);

void pntP1F(board &);

void pntP2F(board &);

void GwentAI(board &, bool& placing);

void clrField(board &);

void GWENT();

void saveG(board &);

void loadG(board &);

void newG(board &);

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

srand(time(0));

board b = {};

board clock[10];

Card cClock[10][2][10]; //NOW WITH 3D EFFECTS

int cnt = 0;

int cho;// user input

bool again;

GWENT();//Game based off The Witcher 3

cout<<"Would you like to load the last saved game?\n";

cout<<"press 0 for no, press 1 for yes\n";

cin>>cho;

if(cho == 1){

//load game from binary files

loadG(b);

pntP1F(b);

cout << "===PLAYERFIELD ^====================AI FIELD v=============================================================================================\n";

pntP2F(b);

}

else {

newG(b); //setup a new game

}

do{

do{//GAME LOOP

do{//match loop

if (cnt < 10){ //amount of cards in hand, it's rare rounds go on this long. Also bounds check to prevent going over allocated memory.

clock[cnt] = b; //save the state of the board

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

for (int cards = 0; cards < 10; cards++){

cClock[cnt][plyrs][cards] = (plyrs == 0)? b.p1Hand[cards]:b.p2Hand[cards]; //I could add a game counter and REALLY turn back the //clock cClock[nGames][cnt][obj][card], but why stop there?

}

}

}

//Condition check at the beginning to see if players can continue to place cards.

if (b.p2Left == 0)b.p2Place = false;

if (b.p1Left == 0)b.p1Place = false;

if (b.p1Place){ //if the player can place cards

cout<<"\nPLAYER 1"<<endl;

pntHand(b, 1); //print their hand of cards

cout<<"\nWhich card?\n";

cin>>cho;

if (cho == -1) { //Player chooses to stop playing cards

b.p1Place = false;

cout<<"You cannot place cards until the next round\n";

}

else if (cho < -1) {//exit condition

cout<<"EXITING GAME\n";

saveG(b); //Save game on quit

b.gaming = false;

b.match = false;

cout << "PREDELETE\n";

delete [] b.deck; //cant figure out why run fails

cout<<"DELETE DECK\n"; //when I load the game and then

delete b.p1Hand; //try to delete the arrays.

cout<<"DELETE HAND1\n";

delete [] b.p2Hand;

cout << "DELETED\n";

return 0;

}

else if (cho == 10 && cnt >=3){ //H4X

do{

cout<<"So you found the cheat, huh. How many turns would you like to turn back the clock?\n";

cout<<"You can only turn back the clock up to 3 turns and the computer may play a different hand"<<endl;

cout<<"CNT = "<<cnt;

cin>>cho;

if (cho == -1){ //Pretend the player never activated the cheat

continue;

}

}while(cho > 3 || cho > cnt); //How far did you think I'd let you rewind the game?

b = clock[cnt-cho]; //rewind the board

for (int i = 0; i <10; i++){ //for some reason, pointers in structs didn't work just by rewinding the board.

b.p1Hand[i] = cClock[cnt-cho][0][i];

b.p2Hand[i] = cClock[cnt-cho][1][i];

}

continue; //TO THE TOP!

}

else if (cho > 10){//bounds checking

cout<<"Number is out of range!\n";

continue;

}

else if (!b.p1Hand[cho].getUsed()){//check if the card has been placed

placeCard(b.p1Hand[cho], b); //place it!

}

else{

cout<<"INVALID ENTRY!\n";

continue;

}

}

cout<<"FIELD\n";

pntP1F(b);//Print the player field

if (b.p2Left>0)

GwentAI(b, b.p2Place);//Place AI card on the field

cout<<endl<<b.p1Place<<"p1Place | p2Place "<<b.p2Place;

pntP2F(b);//print AI field, will combine into 1 function later

b.turn++;

cnt++;

cout<<"====================================================================================\n";

if (b.p1Place == false && b.p2Place == false)b.match = false; //break out once both players stop placing cards

}while(b.match); //exit condition of the round loop

b.p1Place = true; //allow players to place cards again

b.p2Place = true;

b.match = true; //start a new round

if (b.p2Dam >= b.p1Dam){ //computer wins tie

b.p2Wins++;

cout<<"The AI has won the round!"<<endl;

cout<<"You have "<<b.p1Wins<<" wins!\n";

cout<<"The AI has "<<b.p2Wins<<" wins\n";

cin>>cho;//breakpoint

}

else{

b.p1Wins++;

cout<<"You;ve win the round!"<<endl;

cout<<"You have "<<b.p1Wins<<" wins!\n";

cout<<"The AI has "<<b.p2Wins<<" wins\n";

cin>>cho;//breakpoint

}

clrField(b); //clear the field

if (b.p2Wins == 2) //Best 2 out of 3

b.gaming = false;

else if (b.p1Wins == 2)

b.gaming = false;

b.round++; //keep track of nGames

}while(b.gaming); //Main game loop

if (b.p2Wins > b.p1Wins)

cout << "\nYou LOSE!\n";

else

cout << "\nYOU WIN!!!\n";

cout<<"Press 1 to play again"<<endl;

cin>>cho;

again = cho == 1 ? true: false;

newG(b);

}while(again);

delete [] b.deck; delete [] b.p1Hand; delete [] b.p2Hand;

return 0;

}

//===================================================================================================================================

//===================================================================================================================================

void clrField(board &b){

//Reset the playing field and indexes to 0;

for (int i = 0;i < 10; i++){

b.p1Arch[i].setZero();

b.p1Fighter[i].setZero();

b.p1Siege[i].setZero();

b.p2Arch[i].setZero();

b.p2Siege[i].setZero();

b.p2Fighter[i].setZero();

}

b.p1S = 0;

b.p1F = 0;

b.p1A = 0;

b.p2A = 0;

b.p2F = 0;

b.p2S = 0;

b.p2Dam = 0;

b.p1Dam = 0;

}

void pntP2F(board &b){

cout<<"\nTOTAL DAMAGE: "<<b.p2Dam;

cout<<"\nFIGHTERS: ";

if (b.p2F!= 0){ //b.p2F is the index of the card array in the field

for (int i = 0; i < b.p2F; i++){//b.p2F = 0 means there's no fighters in the field

cout<<" D:"<<b.p2Fighter[i].getDam()<<", ";

}

}

else

cout<<"NONE PLACED!";

cout<<"\nARCHERS: ";

if (b.p2A != 0){

for (int i = 0; i < b.p2A; i++){

cout<<" D:"<<b.p2Arch[i].getDam()<<", ";

}

}

else

cout<<"NONE PLACED!";

cout<<"\nSIEGE UNITS: ";

if (b.p2S != 0) {

for (int i = 0; i < b.p2S; i++){

cout <<" D:" << b.p2Siege[i].getDam() << ", ";

}

}

else{

cout<<"NONE PLACED!\n";

}

cout<<endl;

}

void pntP1F(board &b){

cout<<"\nTOTAL DAMAGE: "<<b.p1Dam;

cout<<"\nFIGHTERS: ";

if (b.p1F!= 0){//array index on the field for the player

for (int i = 0; i < b.p1F; i++){

cout<<" D:"<<b.p1Fighter[i].getDam()<<", ";

}

}

else

cout<<"NONE PLACED!";

cout<<"\nARCHERS: ";

if (b.p1A != 0){

for (int i = 0; i < b.p1A; i++){

cout<<" D:"<<b.p1Arch[i].getDam()<<", ";

}

}

else

cout<<"NONE PLACED!";

cout<<"\nSIEGE UNITS: ";

if (b.p1S != 0) {

for (int i = 0; i < b.p1S; i++){

cout <<" D:" << b.p1Siege[i].getDam() << ", ";

}

}

else{

cout<<"NONE PLACED!\n";

}

cout<<endl;

return;

}

void pntHand(board &b, int player){

cout<<"Index Damage Type\n";

if(player == 1){//prints out the cards you have in hand

for (int i = 0; i < 10; i++){

if(!b.p1Hand[i].getUsed()){//if card has been used, don't show in hand

cout<<i;//<<" "<<b.p1Hand[i].getName();I'll make names later

cout<<" "<<b.p1Hand[i].getDam();

cout<<" "<<b.p1Hand[i].getType();

cout<<endl;

}

}

}

else {

for (int i = 0; i < 10; i++) {//This is for debugging. I'll delete it later

if (!b.p2Hand[i].getUsed()) {

cout << i;// << " " << b.p2Hand[i].getName();

cout << " " << b.p2Hand[i].getDam();

cout << " " << b.p2Hand[i].getType();

cout << endl;

}

}

}

}

void GwentAI(board &b, bool& placing){//Places cards on the field from the 'AI'

if (!placing) return;

if (b.p2Dam > b.p1Dam + 3){ //Stops placing cards once AI has 3 point lead or if it lost a round.

placing = false;

return;

}

char t; //type. i.e. Fighter/Archer/Siege

int rndPick, cnt=0;

bool boo;

do{

cnt++;

rndPick = rand()%10; //pick random card from hand. I will change this to pick a card to get closer to the p1Damage+3 threshhold

}while(b.p2Hand[rndPick].getUsed()); //Keep looping until you find something you haven't placed

t = b.p2Hand[rndPick].getType(); //Fighter etc.

switch (t) { //Places the actual card on the field

case 'S':

b.p2Siege[b.p2S] = b.p2Hand[rndPick];

b.p2Dam += b.p2Siege[b.p2S].getDam();

b.p2S++; //increases the index for the next placement.

break;

case 'A':

b.p2Arch[b.p2A] = b.p2Hand[rndPick];

b.p2Dam += b.p2Arch[b.p2A].getDam();

b.p2A++;

break;

case 'F':

b.p2Fighter[b.p2F] = b.p2Hand[rndPick];

b.p2Dam += b.p2Fighter[b.p2F].getDam();

b.p2F++;

break;

//========================For Future versions =============================

/\*case 'f':

b.fieldS[0] = card;

break;

case 'r':

b.fieldS[1] = card;

break;

case 'i':

b.fieldS[2] = card;

break;\*/

default:

break;

}

b.p2Hand[rndPick].setUsed(true);

b.p2Left -= 1; //keep track of how many cards they have on hand

}

void placeCard(Card &card, board &b){

char t;

t = card.getType();

switch (t){//literally copy pasta from AI

case 'S':

b.p1Siege[b.p1S] = card;

b.p1Dam += b.p1Siege[b.p1S].getDam();

b.p1S++;

break;

case 'A':

b.p1Arch[b.p1A] = card;

b.p1Dam += b.p1Arch[b.p1A].getDam();

b.p1A++;

break;

case 'F':

b.p1Fighter[b.p1F] = card;

b.p1Dam += b.p1Fighter[b.p1F].getDam();

b.p1F++;

//========================For Future versions =============================

break;

case 'f':

//b.fieldS[0] = card;

break;

case 'r':

//b.fieldS[1] = card;

break;

case 'i':

//b.fieldS[2] = card;

break;

default:

break; //snek

}

b.p1Left -= 1;

card.setUsed(true);

return;

}

Card \*setDeck(int size){ //create deck

Card \*deck; //pointer to Card class type

deck = new Card [size]; //Can't delete this :(

for (int i = 0; i < size; i++){ //random values for each card

deck[i].setDam(); //Will make a deck with unique cards later

deck[i].setName();

deck[i].setType();

deck[i].setUsed(false);

}

return deck;

}

Card \*setHand(Card \*deck, int size){ //create the hand

Card \*hand;

hand = new Card [10]; //can't delete this either

for (int i = 0; i < 10; i++){

hand[i] = deck[rand()%size]; //pick random card from deck

} //Will pick duplicates

return hand;

}

void GWENT(){

string t;

char newL = '\n';

ifstream g;

g.open("GWENT.txt"); //Text art by http://patorjk.com/software/taag/

while(!g.eof()){

getline(g, t);

t+=newL;

cout<<t;

}

g.close();

}

void saveG(board &b){

fstream bin, hand1, hand2;

bin.open("GWENTBIN.dat", ios::out | ios::binary);

hand1.open("hand1.dat", ios::out | ios::binary);

hand2.open("hand2.dat", ios::out | ios::binary);

bin.write(reinterpret\_cast<char \*>(&b), sizeof(b));

hand1.write(reinterpret\_cast<char \*>(&b.p1Hand[0]), sizeof(Card)\*10);

hand2.write(reinterpret\_cast<char \*> (&b.p2Hand[0]), sizeof(Card)\*10);

bin.close();

hand1.close();

hand2.close();

}

void loadG(board &b){

fstream bin, hand1, hand2;

//open files

bin.open("GWENTBIN.dat", ios::in | ios::binary);

hand1.open("hand1.dat", ios::in | ios::binary);

hand2.open("hand2.dat", ios::in | ios::binary);

b.p1Hand = new Card [10];

b.p2Hand = new Card [10];

bin.read(reinterpret\_cast<char \*>(&b), sizeof(b));

hand1.read(reinterpret\_cast<char \*> (&b.p1Hand[0]), sizeof (Card)\*10);

hand2.read(reinterpret\_cast<char \*> (&b.p2Hand[0]), sizeof (Card)\*10);

bin.close();

hand1.close();

hand2.close();

}

void newG(board &b) {

int size = 50;

b.p1Wins = 0;

b.p2Wins = 0;

b.gaming = true, b.match = true, b.p1Place = true, b.p2Place = true;

b.deck = setDeck(size); //create the array of cards. ATM Random values per card

b.p1Hand = setHand(b.deck, size); //Pick 10 cards from the deck

b.p2Hand = setHand(b.deck, size); //There's a high chance to see a duplicate card

b.p1Left = 10; //Cards left in the hand

b.p2Left = 10;

}