Project I

Lemming Ops: A War Game

Jeffrey Thomas

CSC-17A, 42636

Dr. Mark Lehr

*Table of Contents*

Introduction………………………………………………………3

Summary…………………………………………………………4

Description………………………...…………………………….5

Flowcharts………………………...…………………………..6-7

Pseudocode……………………………...………………….8-11

Table of Variables/Concepts…………………………………12

Full Code……………………………….…………………..13-34

References………………………………………….………….35

Acknowledgements…………………………………………....36

*Introduction*

Lemming Ops is a war game with its gameplay being similar to the board game Risk. What is different about this game is that progression is much more linear and other aspects of the game have been expanded to make the game feel more like a journey than a simple game of conquest.

The frequently changing nature of army sizes in a war game are what initially made me decide to use this game to demonstrate the concepts I’ve learned.

War is a relevant, universal topic that makes the game simple to pick up and understand. Making a simple game with a nuanced backend was my objective.

*Summary*

From start to finish, this project took approximately 25 hours to complete. This project utilizes various aspects of chapters 1-12, including: structured data, pointers, dynamic arrays, binary file operations, c-strings, etc. The project is just under 600 lines from the beginning of main to the end of the last function definition (including comments and spacing).

Line Count (w/o comments and spaces): ~500

Line Count (total): 622

Number of variables: 14

Number of constants: 1

Number of constructs: 46

Number of functions: 20 + main()

*Description*

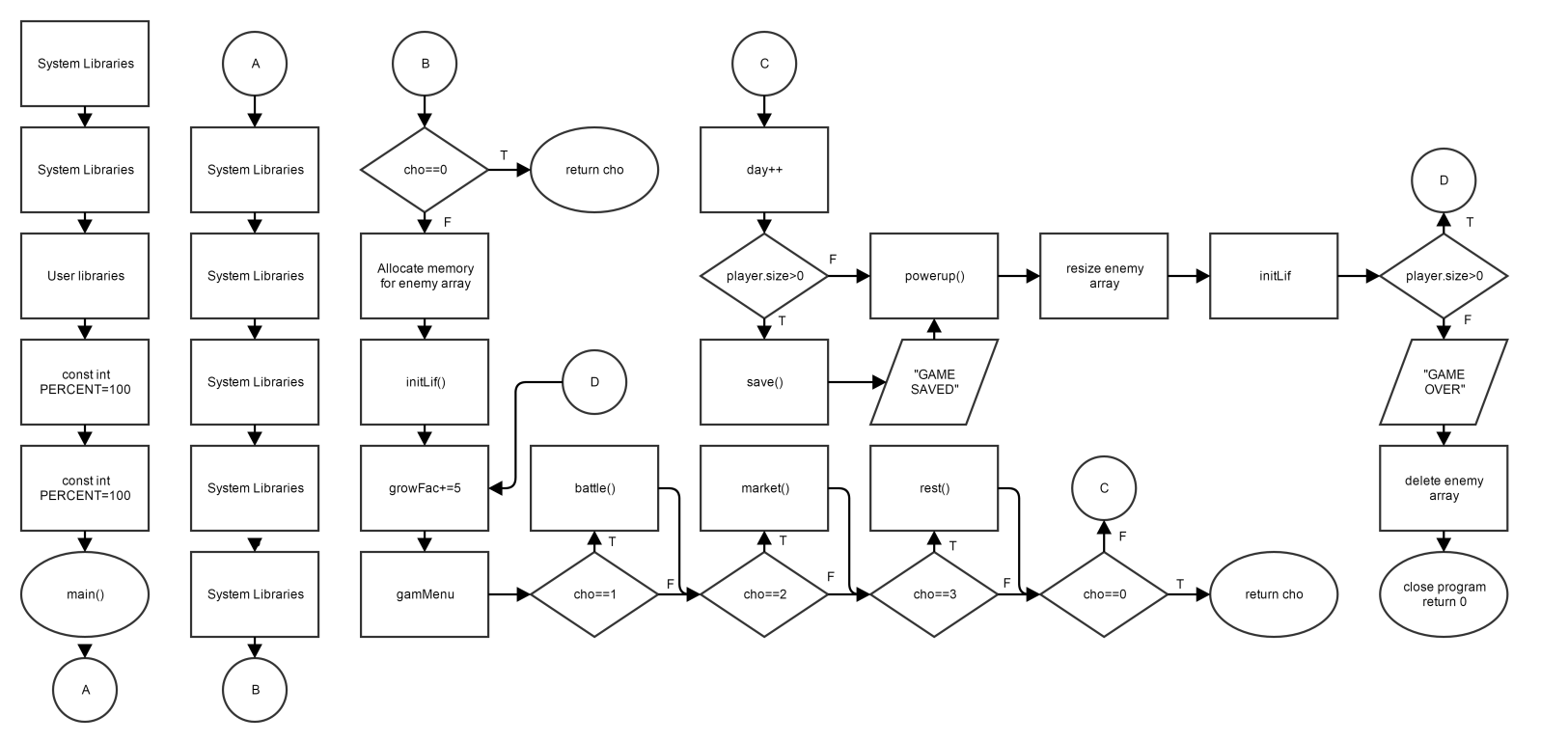
The two primary problems in making a game of this nature are as follows:

1. There is quite a bit of player data that needs to be tracked so there needs to be an orderly way of keeping track of it all without flooding the program with dozens of variables
2. There needs to be a way to frequently resize the arrays that keep track of casualties during battle

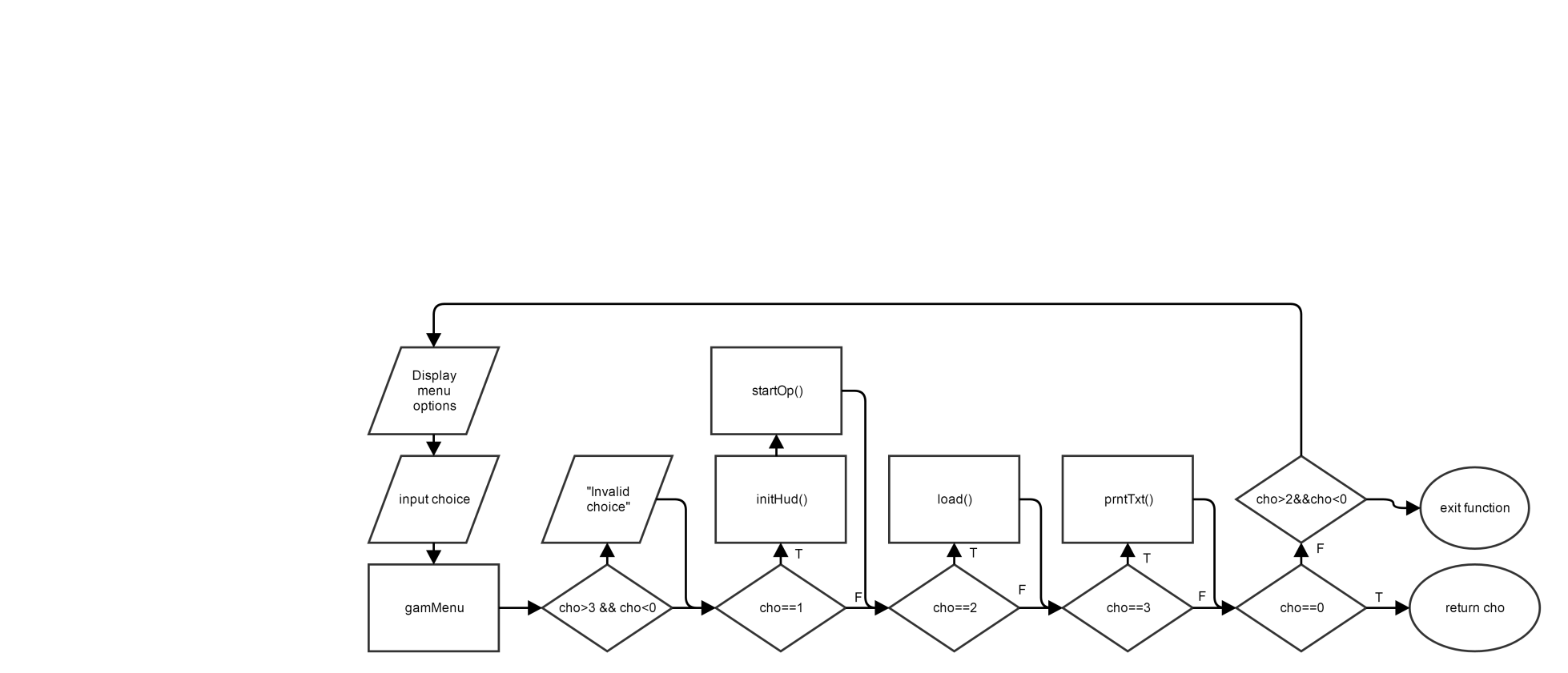
The solutions to these two problems lie in the implementation of structured data and dynamic memory.

*Flowcharts*

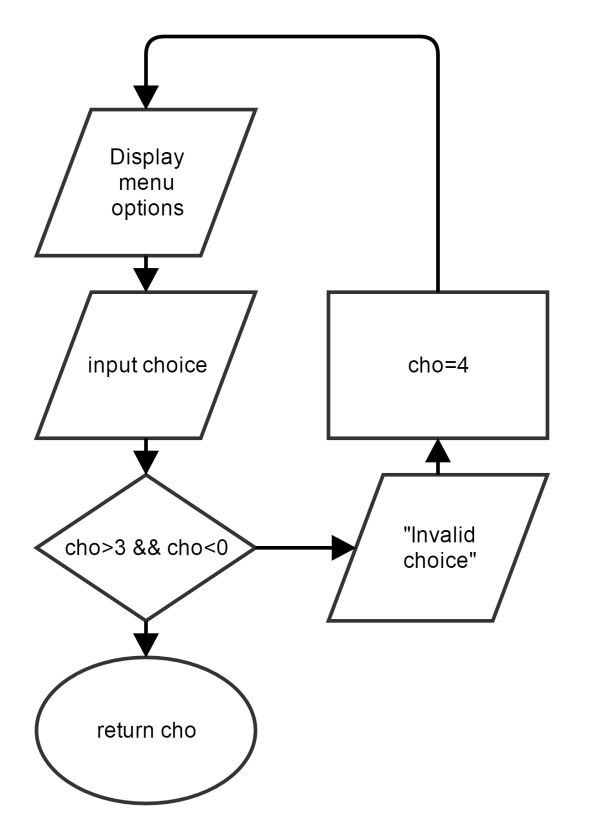
main



ttlMenu



gamMenu



*Pseudocode*

*Include system libraries*

*Include user libraries*

*Declare constants*

*Function prototypes*

*Main start*

*Initialize random seed*

*Declare primitive variables*

*Declare structure variables*

*Set default values for enemy variable*

*Output title and menu screen*

*Prompt user for input*

*Choice 1*

*Set default values for game currency*

*Output greeting message*

*Prompt user for more input*

*Set number of soldiers*

*Decrease currency according to number of soldiers*

*Set default values for soldiers*

*Declare new dynamic array*

*Display confirmation message*

*Choice 2*

*Declare file stream object*

*Load in player data*

*Load in enemy data*

*Load in HUD data*

*Load in day count*

*Close file*

*Initialize player army with new data*

*Choice 3*

*Declare string object*

*Declare file stream object*

*Open file*

*Continue until file is completely read in*

*Read in line*

*Output line*

*Close file*

*Choice 0*

*Close program*

*Continue game until player dies*

*Increase growth rate of enemy*

*Repeat until input is valid*

*Display game menu*

*Prompt user for input*

*Input choice*

*Choice 1*

*Repeat until all squads have fought*

*Display maximum possible squad*

*Input squad size*

*Set squad member counter to 0*

*Roll for first strike*

*Check if attack was successful*

*Set soldier in array to dead*

*Increment defeat*

*Display message*

*Increment counter*

*Tally up defeats*

*Reset counters*

*Display status report*

*Increase food based on number of enemy defeats*

*Increase food based on number of enemy defeats*

*Increase or decrease morale based on defeats*

*Check which soldiers are still alive*

*Resize array for living soldiers*

*Delete old array*

*If the army still has soldiers*

*Make new array for new size*

*If there are no soldiers left*

*Keep the array deleted*

*Choice 2*

*Repeat until valid choice is entered*

*Display options*

*Input choice*

*Choice 1*

*Explain selection*

*Prompt user for purchase amount*

*Input amount*

*Add purchase to inventory*

*Remove gold from inventory*

*Resize array to fit purchase*

*Choice 2*

*Explain selection*

*Prompt user for purchase amount*

*Input amount*

*Add purchase to inventory*

*Remove gold from inventory*

*Choice 0*

*Leave market*

*Choice 3*

*Set counter to 0*

*Remove food from inventory*

*Increase morale by random amount*

*Increase army size by random amount*

*Delete old array*

*Make new array for new size*

*Choice 0*

*Exit program*

*Display game over message*

*Delete enemy array*

*Exit program*

*Table of Variables/Concepts*

|  |  |
| --- | --- |
| #include | 9-13,17,18 |
| Const | 21 |
| Void | 24,etc |
| & | 24-26,etc |
| Int | 52-54,etc |
| Structured data | 57,58,61 |
| If | 71,etc |
| Return | 71,etc |
| New | 74,121,203,375,510,562,613 |
| Delete | 129,372,508,561 |
| Do-while | 80,etc |
| Switch case | 87,etc |
| Break | 91,etc |
| Math operators | 82,etc |
| Relational operators | 112,etc |
| Dot operator | 138,etc |
| For | 144,etc |
| String | 161 |
| Fstream | 163,etc |
| While | 168 |
| Else if | 266 |
| Static\_cast | 295,etc |
| Reinterpret\_cast | 573,etc |
| Logical operators | 273,etc |
| float | 136,etc |
| Char[] | 35 |
| bool | 74,etc |

A word on the number of ‘new’s vs the number of ‘delete’s.

Normally a disparity in the number of news and deletes would be an unforgivable offense. However, as a result of the nature of my game, some allocations are conditional and usually only activate if preceded by a delete. Despite looking sloppy at first glance, a more careful look will indicate that there are no leaks present in my code.

*Program*

/\*

\* File: main.cpp

\* Author: Jeffrey Thomas

\* Purpose: Project 1 CSC-17A

\*/

//System Libraries

#include <iostream> //Input/Output objects

#include <cstdlib> //Random Generator

#include <ctime> //Random Seed

#include <fstream> //File IO

using namespace std; //Namespace used in system library

//User libraries

#include "Lemming.h" //Lemming struct

#include "Hud.h" //Hud struct

//Global constants

const int PERCENT=100; //Percent conversion

//Function prototypes

void initSts(Lemming&,int,float); //Set stats for player or enemy

void initLif(Lemming&); //Set all lemmings to 'alive'

Hud initHud(Hud&); //Set default values to HUD

Lemming powerUp(Lemming&,int); //Powers up enemy lemming based on growth

void rest(Lemming&,Hud&); //Rest option

void market(Lemming&,Hud&); //Market option

int ttlMenu(Lemming&,Lemming&,Hud&,int&,int&);//Title menu

int gamMenu(Lemming,Hud,int&,int&); //Game menu

void startOp(Hud&,Lemming&); //Introduction function

void prntTxt(char[]); //Output ASCII data from file

void battle(Lemming&,Lemming,Hud&,int);//All battle phases

void divide(Lemming&,Lemming&); //Division phase of battle

void combat(Lemming&,Lemming&,Lemming,Lemming);//Combat phase of battle

Hud plunder(Hud&,Lemming,Lemming,int);//Plunder phase of battle

void regrou(Lemming&); //Regroup phase of battle

void save(Lemming,Lemming,Hud,int); //Save data to binary file

void load(Lemming&,Lemming&,Hud&,int&);//Load data from binary file

//Execution begins here

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

//Initialize random seed

srand(time(0));

//Declare Variables

int cho; //User inputted choice for various prompts

int day=1; //Initial day

int growFac=0; //Initial enemy growth factor

//Declare lemmings

Lemming player; //Player data

Lemming enemy; //Enemy data

//Declare HUD

Hud hud; //HUD data

//Initialize default enemy values

initSts(enemy,10,0.5f);

//Display title screen

prntTxt("title.txt");

//Title Menu

ttlMenu(player,enemy,hud,day,cho);

if(cho==0)return cho;

//Initial enemy memory allocation

enemy.isDead=new bool[enemy.size];

//Set all enemy lemmings to 'alive'

initLif(enemy);

//Display Menu

do{

//Increase enemy growth factor

growFac+=5;

//Display HUD and options

gamMenu(player,hud,day,cho);

//Option Branches

switch(cho){

case 1:

//Commence Battle

battle(player,enemy,hud,day);

break;

case 2:

//Enter Marketplace

market(player,hud);

break;

case 3:

//Rest for the day

rest(player,hud);

break;

case 0:

return cho;

break;

default:

//Display invalid choice message

cout<<"Invalid choice"<<endl;

}

//Increment in-game day

day++;

//Save Progress

if(player.size>0){

save(player,enemy,hud,day);

cout<<"GAME SAVED"<<endl<<endl;

}

//Increase Enemy Power

powerUp(enemy,growFac);

//Reinitialize isDead array

enemy.isDead=new bool[enemy.size];

initLif(enemy);

}while(player.size>0);

//Display 'game over' message

cout<<"GAME OVER"<<endl<<endl;

//Delete Arrays

delete []enemy.isDead;

//Exit program

return 0;

}

//Initializers

void initSts(Lemming &l,int size,float dodge){

//Set size

l.size=size;

//Set dodge rate

l.dodge=dodge;

}

void initLif(Lemming &l){

//Set all elements in array to '0'

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

l.isDead[i]=0;

}

}

Hud initHud(Hud &h){

//Default values for start of game

h.gold=100;

h.food=100;

h.morale=0.0f;

//Return newly initialized HUD

return h;

}

//Output Functions

void prntTxt(char file[]){

//String object to hold data

string line;

//File stream object

ifstream read;

//Open file

read.open(file);

//Read line into string

getline(read,line);

while(getline(read,line)){

//Output line

cout<<line<<endl;

}

//Close file

read.close();

}

//Introduction Function

void startOp(Hud &h,Lemming &l){

//Holds value for user inputted choice

int cho;

//Output Greeting

cout<<"Greetings!"<<endl<<endl;

cout<<"You have been selected to spearhead our new"<<endl

<<"military project! You will be granted "<<h.gold<<" gold as"<<endl

<<"a starting grant. Use as you'd like! But remember that"<<endl

<<"this is all you're getting from us. You'll have to get"<<endl

<<"more money on your own. In any case, good luck commander!"<<endl

<<"Make us proud."<<endl<<endl;

cout<<"~ Project LEMMING"<<endl<<endl;

//Prompt User for Initial Army Size

do{

cout<<"How many lemmings will you purchase? (5 Gold/Lemming)"<<endl;

cin>>l.size;

}while(l.size<1&&l.size>h.gold/5);

//Subtract price from total

h.gold-=(l.size\*5);

//Initialize player with declared army size

initSts(l,l.size,0.5f);

//Initial player memory allocation

l.isDead=new bool[l.size];

//Set all lemmings to 'alive'

initLif(l);

//Display Confirmation

cout<<"You will start with "<<l.size<<" lemmings."<<endl;

}

//Battle Phases

void battle(Lemming &player,Lemming enemy,Hud &h,int day){

do{

//Division Phase

cout<<"Division Phase"<<endl;

if(player.size>=enemy.size){

divide(player,enemy);

//Combat Phase

cout<<"Combat Phase"<<endl;

combat(player,enemy,player,enemy);

cout<<endl<<endl;

}

else{

//Enemy chooses largest possible squad size by default

enemy.squad=enemy.size/player.size;

//Set player squad size to 1

player.squad=1;

//Display squad size

cout<<"The enemy is fielding a larger army."<<endl;

cout<<"Enemy Squad Size: "<<enemy.squad<<endl;

//Combat Phase

cout<<"Combat Phase"<<endl;

combat(player,enemy,enemy,player);

cout<<endl<<endl;

}

//Plunder Phase

plunder(h,enemy,player,day);

//Regroup Phase

regrou(player);

regrou(enemy);

}while(player.size!=0&&enemy.size!=0);

}

void divide(Lemming &pl,Lemming &en){

do{

//Display sizes of armies

cout<<"Your Army : "<<pl.size<<" units"<<endl;

cout<<"Enemy Army: "<<en.size<<" units"<<endl;

//Notify user of maximum value

cout<<"You may have a squad of up to "

<<pl.size/en.size<<" for this battle."<<endl;

//Prompt user for input

cout<<"Input squad size: "<<endl;

cin>>pl.squad;

//Display error for exceeding maximum

if(pl.squad>pl.size/en.size){

cout<<"Refer to your maximum squad size and "

<<"input a valid number."<<endl;

}

//Display error for not meeting minumum

else if(pl.squad<1){

cout<<"Squad size must be at least 1."<<endl;

}

//Display confirmation message

else{

cout<<"You have selected a squad size of "<<pl.squad;

}

}while(pl.squad>(pl.size/en.size)||pl.squad<1);

//Set enemy squad to 1 by default

en.squad=1;

}

void combat(Lemming &pl,Lemming &en,Lemming big,Lemming sml){

//Set squad member counter to 0

int count=0;

//Set round defeat counter to 0

int pdef=0,edef=0;

//Set total defeats to 0

pl.defeat=0;en.defeat=0;

//Loop through rounds

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

//Display round number

cout<<"Squad "<<i+1<<" begin"<<endl;

do{

//Roll for First Strike (0=player FS,1=enemy FS)

if(rand()%2==0){

//Display who will attack first

cout<<"Player strikes first!"<<endl;

//Display message for missed attack

if(rand()%static\_cast<int>(en.dodge\*100)>

rand()%static\_cast<int>(pl.dodge\*100)){

cout<<"Attack from player missed!"<<endl;

}

//Display message for successful attack

else{

cout<<"Attack successful!"<<endl;

//Set enemy to dead

en.isDead[i]=1;

//Increment defeat counter

edef++;

}

}

//If enemy wins coin toss

else{

//Display missed attack message

cout<<"Enemy strikes first!"<<endl;

if(en.dodge\*100>rand()%100){

cout<<"Attack from Enemy missed!"<<endl;

}

//Display successful attack message

else{

cout<<"Attack successful!"<<endl;

//Set player to dead

pl.isDead[i\*pl.squad+count]=1;

//Increment defeat counter

pdef++;

}

}

//Increment squad counter

count++;

//Continue until one squad is dead or number of turns has been reached

}while((edef<en.squad&&pdef<pl.squad)&&count<big.squad);

cout<<endl;

//Tally up defeats

pl.defeat+=pdef;

en.defeat+=edef;

//Reset counters

count=0;

pdef=0;

edef=0;

}

cout<<endl;

cout<<"Status Report: "<<endl;

cout<<"Player Casualties: "<<pl.defeat<<endl;

cout<<"Enemy Casualties : "<<en.defeat<<endl;

cout<<endl;

cout<<"Key: 0 = alive, 1 = dead"<<endl<<endl;

cout<<"Player Army"<<endl;

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

cout<<pl.isDead[i];

if(i%50==49)cout<<endl;

}

cout<<endl;

cout<<"Enemy Army"<<endl;

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

cout<<en.isDead[i];

if(i%50==49)cout<<endl;

}

cout<<endl;

}

void regrou(Lemming &l){

//Initialize counter

int count=0;

//Check for dead lemmings

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

if(l.isDead[i]==0)count++;

}

//Set new army size

l.size=count;

//Delete old array

delete []l.isDead;

//Initialize new array with new size

if(l.size>0){

l.isDead=new bool[count];

}

//Set all new lemmings to 'alive'

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

l.isDead[i]=0;

}

}

Hud plunder(Hud &h,Lemming en,Lemming pl,int day){

for(int i=0;i<en.defeat;i++){

//Increase food stock by random amount per dead enemy

h.food+=rand()%(day\*5);

//Increase gold stock by random amount per dead enemy

h.gold+=rand()%(day\*5);

//Increase morale for each dead enemy

h.morale+=static\_cast<float>(en.defeat)/100;

//Decrease morale for each dead player

h.morale-=static\_cast<float>(pl.defeat)/100;

}

//Return new HUD values

return h;

}

//Key Game Functions

int ttlMenu(Lemming &player,Lemming &enemy,Hud &hud,int &day,int &cho){

//Display options

do{

cout<<"1. New Game"<<endl;

cout<<"2. Load Game"<<endl;

cout<<"3. Instructions"<<endl;

cout<<"0. Quit"<<endl;

//Input choice

cin>>cho;

//Tell user if input is invalid

if(cho>3||cho<0)cout<<"Invalid option!"<<endl;

//Option Branches

switch(cho){

case 1:

//Set default HUD values

initHud(hud);

//Play Introduction

startOp(hud,player);

break;

case 2:

//Load Game

load(player,enemy,hud,day);

break;

case 3:

//Display Instructions

prntTxt("instructions.dat");

cout<<endl;

break;

case 0:

//Return 0

return cho;

break;

}

}while(cho>2||cho<0);

}

int gamMenu(Lemming player,Hud hud,int &day,int &cho){

do{

//Display HUD

cout<<"Day "<<day<<endl;

cout<<"Troops: "<<player.size<<endl;

cout<<"Gold : "<<hud.gold<<endl;

cout<<"Food : "<<hud.food<<endl;

cout<<"Morale: "<<hud.morale<<endl;

cout<<endl;

//Display menu options

cout<<"1. Next Battle"<<endl;

cout<<"2. Marketplace"<<endl;

cout<<"3. Rest"<<endl;

cout<<"0. Quit Game"<<endl;

//Input choice

cin>>cho;

//User must have a minimum of 3 food per lemming to rest

if(cho==3&&(player.size\*3)>hud.food){

//Display insufficient food message

cout<<"There is not enough food for all of your troops."<<endl;

//Send player back into loop

cho=4;

}

}while(cho>3||cho<0);

//Return menu choice

return cho;

}

Lemming powerUp(Lemming &en,int gf){

//Increase size based on growth factor

en.size+=rand()%gf;

//Increase dodge rate based on growth factor

en.dodge+=static\_cast<float>(rand()%gf)/PERCENT;

//Return newly powered enemy

return en;

}

void market(Lemming &player,Hud &hud){

//Holds user inputted choices

int cho;

do{

//Display menu options

cout<<"Welcome! Make a selection from our fine wares."<<endl;

cout<<"1. Buy Soldiers"<<endl;

cout<<"2. Agility Training"<<endl;

cout<<"0. Leave Market"<<endl;

//Input choice

cin>>cho;

//Display invalid option message

if(cho>2||cho<0)cout<<"We don't sell that here."<<endl;

}while(cho>2&&cho<0);

//Menu option branches

switch(cho){

case 1:

do{

//Buy soldier prompt

cout<<"We sell our combat lemmings for 100 gold each."<<endl;

cout<<"How many would you like?"<<endl;

//Input amount to purchase

cin>>cho;

//Display insufficient funds message

if(cho\*100>hud.gold){

cout<<"You have insufficient funds."<<endl;

}

}while(cho\*100>hud.gold);

//Display 0 lemmings message

if(cho==0){

cout<<"Suit yourself"<<endl;

}

else{

//Remove money from purchase

hud.gold-=cho\*100;

//Increase army size

player.size+=cho;

//Delete old army array

delete[]player.isDead;

//Create bigger army array

player.isDead=new bool[player.size];

//Set all soldiers to 'alive'

initLif(player);

}

break;

case 2:

//Agility training explanation

cout<<"Agility training will make your lemmings better at "<<endl

<<"dealing and dodging hits in battle. We charge a fee "<<endl

<<"of 25 gold per lemming per hour for training."<<endl;

do{

//Buy training prompt

cout<<"How many hours of training would you like?"<<endl;

//Input purchase choice

cin>>cho;

//Display insufficient funds message

if(cho\*player.size\*25>hud.gold){

cout<<"You have insufficient funds."<<endl;

}

}while(cho\*player.size\*25>hud.gold);

//Display 0 hours message

if(cho==0){

cout<<"Suit yourself"<<endl;

}

else{

//Remove gold from player account

hud.gold-=cho\*player.size\*25;

//Increase dodge rate according to purchase

player.dodge+=static\_cast<float>(cho)/PERCENT;

}

break;

case 0:

//Leave market message

cout<<"Come again soon..."<<endl;

break;

}

}

void rest(Lemming &l,Hud &h){

//Set birth counter to '0'

int birth=0;

//Decrease food by 3 food per lemming

h.food-=(l.size\*3);

//Increase morale by random amount scaled to number of lemmings

h.morale+=static\_cast<float>(rand()%50+1)/PERCENT;

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

//Roll dice for birth of new soldier

if(h.morale\*PERCENT>rand()%100)birth++;

}

//Make army bigger (if applicable)

l.size+=birth;

delete []l.isDead;

l.isDead=new bool[l.size];

initLif(l);

}

//Save/Load

void save(Lemming pl,Lemming en,Hud h,int day){

//Declare file stream object

fstream file;

//Declare char pointer for data to be casted into

char \*par1;

//Cast pointer to first piece of data

par1=reinterpret\_cast<char\*>(&pl.size);

//Open file

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

//Write player data

file.write(reinterpret\_cast<char\*>(&pl.size), sizeof(pl.size));

file.write(reinterpret\_cast<char\*>(&pl.gift), sizeof(pl.gift));

file.write(reinterpret\_cast<char\*>(&pl.dodge),sizeof(pl.dodge));

//Write enemy data

file.write(reinterpret\_cast<char\*>(&en.size),sizeof(en.size));

file.write(reinterpret\_cast<char\*>(&en.dodge),sizeof(en.dodge));

//Write HUD data

file.write(reinterpret\_cast<char\*>(&h.gold),sizeof(h.gold));

file.write(reinterpret\_cast<char\*>(&h.food),sizeof(h.food));

file.write(reinterpret\_cast<char\*>(&h.morale),sizeof(h.morale));

//Write day counter

file.write(reinterpret\_cast<char\*>(&day),sizeof(day));

//Close file

file.close();

}

void load(Lemming &pl,Lemming &en,Hud &h,int &day){

//Instantiate file stream object

fstream file;

//Open file for binary output

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

//Read player data

file.read(reinterpret\_cast<char\*>(&pl.size),sizeof(pl.size));

file.read(reinterpret\_cast<char\*>(&pl.gift),sizeof(pl.gift));

file.read(reinterpret\_cast<char\*>(&pl.dodge),sizeof(pl.dodge));

//Read enemy data

file.read(reinterpret\_cast<char\*>(&en.size),sizeof(en.size));

file.read(reinterpret\_cast<char\*>(&en.dodge),sizeof(en.dodge));

//Read HUD data

file.read(reinterpret\_cast<char\*>(&h.gold),sizeof(h.gold));

file.read(reinterpret\_cast<char\*>(&h.food),sizeof(h.food));

file.read(reinterpret\_cast<char\*>(&h.morale),sizeof(h.morale));

//Read day count

file.read(reinterpret\_cast<char\*>(&day),sizeof(day));

//Close file

file.close();

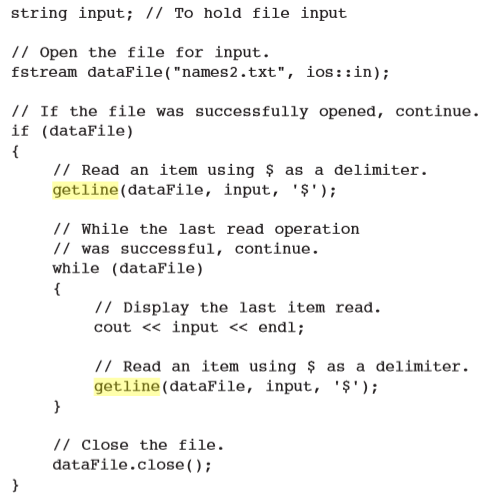
//Reinitialize player army with new data

pl.isDead=new bool[pl.size];

initLif(pl);

*References*

I had to refer to the textbook, Starting out with C++ by Tony Gaddis for help with binary file i/o. I referred to the following piece of code for reference while writing the save and load functions for my game. This piece of code is located on page 674 of the 8th ed.



*Acknowledgements*

Augies Coffee Roasters

Dr. Mark E. Lehr

All of my programmer friends who aren’t as good as me