Vidzemes Augstskola

Inženierzinātņu fakultāte

**C++ SPĒLE GLADIATORS**

C++ DOKUMENTĀCIJA

Darba autors: Oļģerts Klāvs Balabāns

Valmiera 2018

Programmas apraksts

Izveidotā programma „Gladiators” ir teksta bāzēta spēle kurā tu kā spēlētājs uzņemies zobena lietojoša gladiatora lomu. Spēles kaujas notiek izmantojot kursora atrašanās vietu, lai izdarītu kaut kādu darbību cīņas laikā ir jākustina pelīte. Ir šādas kontroles iespējas cīņas laikā :

* Kustināt peli pa labi uz augšu (sist pa labo augšējo pusi)
* Kustināt peli pa labi uz leju (sist pa labo apakšējo pusi)
* Kustināt peli pa kreisi uz augšu (sist pa kreiso augšējo pusi)
* Kustināt peli pa kreisi uz leju (sist pa kreiso apakšējo pusi)
* Izveidot apli ar peles kustību (bloks)

Ārpus cīņas ir iespējams pastiprināt savu sišanas spēku, savas dzīvības un savus refleksus (palielina laiku kurā var veikt cīņas laikā darbības).

Cīņas laikā ir divi posmi, aizkaramības posms un atkopšanās posms. Aizkaramības posms ir starp pretinieka sagatavošanos uzbrukt un uzbrukumu, šajā posmā spēlētājs var trāpīt pretiniekam. Atkopšanās posms ir brīdis, kad pretinieks un tu atkopjas no uzbrukuma, šajā laikā nevar izdarīt uzbrukumus vai blokus.

Cīņas laikā ir iespējams bloķēt un uzbrukt vienu reizi. Ja spēlētājs bloķē pirms uzbrūk, uzbrukuma opcija tiek noņemta uz to raundu, lai simulētu realitāti. Katru raundu spēlētājs var uzbrukt tikai tai vietai, kas tagad ir pavājināta.

Windows.h apraksts

Windows.h ir Windows specifiska galvenes fails C un C ++ programmēšanas valodām, kas satur deklarācijas par visām Windows API funkcijām, visiem kopējiem makro, ko izmanto Windows programmētāji, un visiem datu tipiem, ko izmanto dažādas funkcijas un apakšsistēmas. Tā definē ļoti lielu skaitu Windows specifisku funkciju, ko var izmantot C.

Piemēri:

excpt.h - Izņēmuma apstrāde

stdarg.h - mainīgo argumentu funkcijas (standarta C galvene)

windef.h - dažādi makro un veidi

winnt.h - dažādi makro un veidi (Windows NT)

basetsd.h - dažādi veidi

guiddef.h - GUID veids

ctype.h - rakstzīmju klasifikācija (standarta C galvene)

string.h - virknes un buferi (standarta C galvene)

Windows.h pielietošana programmā

Programmā tiek izmantota specifiska windows.h funkcija, ar kuras palīdzību var noteikt kursora atrašanās vietu. Šī funkcija ir izmantota, priekš cīņas kontrolēm.

POINT getCourserLocation() {

POINT p;

GetCursorPos(&p);

return p;

}

Šī funkcija atbild par punkta atrašanās vietas iegūšanu, kodā tā tiek izmantota katru spēles tikšķi, kurā notiek cīņa.

bool NE(POINT &n, POINT &o, int time) {

if ((o.x + time < n.x ) && (o.y- time > n.y)) {

return true;

}

else return false;

}

Šī funkcija atbild par to vai šajā brīdī tiek izveidota kustība NE (North East) virzienā. Funkcija salīdzina pašreizējo n (new courser) un veco (old courser) kursora atrašanās vietu. Lai Katra mazākā peles kustība NE virzienā netiktu uzkatīta par kustību, tiek vecajai vērtībai pieskaitīts vai atņēmts maniīgais time. Mainīgais time nosaka cik ātri pelīte ir jākustina NE virzienā pirms to ieskaita kā kustību.

Testa piemēri

Lai iztestētu jebkuru daļu no spēles ir jāuzspēlē spēle. Ir noteikti mainīgie kuru izmainīšana deklerācijā var palīdzēt iztestēt specifiskas funkcijas:

* palyerExp nomaina lai iztestētu Level up funkciju

Piem: nomaina playerExp uz 100 un spēlēs laikā izmanto Level up funkciju lai palielinātu spēlētāja dzīvības 5 reizes. Pēctam pārbauda dzīvību skaitu Stats logā vajadzētu būt:

Player health:15

Player damage:1

Player agility modifier:0

Player manuver modifier:0

* enemyLevel nomaina, lai iztestētu pretiniek pastiprināšanos

Piem: nomaina enemyLevel uz 10 un spēles laikā aiziet uz Stats logu un paskatās pretinieka status jābūt ir sekojoši:

Enemy attack damage:10

Time for enemy to attack:3000

Time for enemy to recove after attack:1000

Komentēts programmas teksts

// CourseProject.cpp : This file contains the 'main' function. Program execution begins and ends there.

//

#include "pch.h"

#include <iostream>

#include<windows.h>

#include <string>

#include <time.h>

#include <stdlib.h>

using namespace std;

POINT getCourserLocation() {

POINT p;

GetCursorPos(&p);

return p;

}

bool NE(POINT &n, POINT &o, int time) {

if ((o.x + time < n.x ) && (o.y- time > n.y)) {

return true;

}

else return false;

}

bool NW(POINT &n, POINT &o, int time) {

if ((o.x - time > n.x) && (o.y - time > n.y)) {

return true;

}

else return false;

}

bool SE(POINT &n, POINT &o, int time) {

if ((o.x + time < n.x) && (o.y + time < n.y)) {

return true;

}

else return false;

}

bool SW(POINT &n, POINT &o,int time) {

if ((o.x - time > n.x) && (o.y + time < n.y)) {

return true;

}

else return false;

}

int LockProgression(int lN, int cL[2][5], int lP[2], int selectedLock) { // returns how far the lock is

if ((lN == cL[selectedLock][lP[selectedLock]]) || (lN == cL[selectedLock][lP[selectedLock] + 1])) {

if (lN == cL[selectedLock][lP[selectedLock] + 1]) {

lP[selectedLock] = lP[selectedLock] + 1;

}

}

else {

lP[selectedLock] = 0;

}

return lP[selectedLock];

}

bool CheckIfAplis (int lP[2]) {

if (lP[0]>=4 || lP[1]>=4) {

return true;

}

return false;

}

void ExecuteHit(int &enemyHealth, int &playerAttackDamage, bool &playerHasHit) {

playerHasHit = true;

enemyHealth = enemyHealth - playerAttackDamage;

cout << "\*HIT\*" <<" "<<"Enemy health:" << enemyHealth << endl;

}

void WinSituation(int &playerExp, int &enemyLevel, int &gameMode) {

cout << "YOU WIN!!!" << endl;

playerExp = playerExp + enemyLevel \* 5 + 10;

cout << "You have earned " << enemyLevel \* 5 + 10 << "xp " << endl;

gameMode = 1;

cout << endl << "---------------------------------------------------------------------------------------" << endl;

}

void GetRandExsposedPlace(int &exsposedPlace, string &exsposedPlaceText) {

srand(time(NULL));

exsposedPlace = rand() % 4 + 1;

switch (exsposedPlace)

{

case 1: exsposedPlaceText = "RIGHT UPPER";

break;

case 2: exsposedPlaceText = "LEFT UPPER";

break;

case 3: exsposedPlaceText = "LEFT LOWER";

break;

case 4: exsposedPlaceText = "RIGHT LOWER";

break;

default:

break;

}

}

void ShowTutorialText() {

cout << "This is a game where you play a gladiator who fight for glory (no money) and experience." << endl;

cout << "Combat: To attack you have to move the mouse in the direction that the enemy has exsposed himselfe in!" << endl;

cout << "For example: He expsposes his LEFT UPPER side you would swipe diognally to your left upper corner on the screen." << endl;

cout << "You can block enemy attacks, to do so draw a circle with your mouse. After blocking you cannot attack so block after attacking or if you feel like you wont be able to react" << endl;

cout << "Combat is divided in two parts. Enemy recovery phase and enemy vulnerability phase. The recovery phase is how mutch time there is between attack rounds.";

cout << "The vulnerability phase is the time it takes for the enemy to attack, this period is also the time you can attack the enemy!" << endl;

cout << "After every enemy you defeat you gain experience points that you can spend in the level up screen." << endl;

cout << " Remember that after you close the game it wont save your progress." << endl;

}

int main()

{

POINT cursor, oldCursor;

GetCursorPos(&cursor);

int waitTime = 100; // time between game ticks

int checkTime = waitTime / 5; // used for making small movements on the mouse not count as movements, the bigger more the mouse has to move for it to be counted as movement

int lockNumber = 0; // Dont change, used for calculating if there is a circle

int circleLock[2][5] = { { 0,1,2,3,4 }, {0,4,3,2,1} }; // defines two number combinations that need to happen for a circle to form. first 0 is for it to be able to be reset

int lockProgress[2] = { 0,0 }; //How many numbers of the combination have been unlocked

bool controlsEnabled = false; //Enables or disbles the mouse controls

int gameMode = 0; // Dictates current gameMode 0-Menu 1-combat 2-movement 3-Inventory 4-stats

int choice = 0;

//Variables for combat

int getAttackReadyTime = 0;

int playerAttackDamage = 1;

int waitAfterAttack = 0;

int baseGetAttackReadyTime = 4000;

int baseWaitAfterAttack = 2000;

int diffGetAttackReadyTime = 0;

int diffWaitAfterAttack = 0;

int battlePhase = 0;

int accumulatedWaitTime = 0;

int exsposedPlace = 0;

string exsposedPlaceText;

bool blockIsActive = false;

int enemyAttackDamage = 0;

int playerCurrentHealth = 0;

int playerMaxHealth = 5; // Change this is you want more starting health

bool playerHasHit = false;

int enemyHealth = 0;

int enemyLevel = 0; // Change if you want starting level of enemy to be higher

int manuver = 0; // Starting manuver value increases the time it takes for enemy to recover after attack

int agility = 0; // Starting agility value increases the time it takes for enemy to get ready

//LevelUp variables

int playerExp = 0; // Change if you want more starting experience

int upgrHealthCost = 0;

int upgrDamageCost = 0;

int upgrAgilityCost = 0;

int upgrManuverCost = 0;

while (true) {

if (gameMode == 0) { //Start game screen

//Contains just a few navigation choices

cout << "1.Start game!" << endl << "2.Exit" << endl << "3.Tutorial" << endl << "Choose:";

cin >> choice;

while (!cin.good()) {

cin.clear();

cin.ignore(256, '\n');

}

switch (choice)

{

case 1:

gameMode = 1;

cout << endl << "---------------------------------------------------------------------------------------" << endl;

cout << "This is a roguelike YOU HAVE ONE LIFE!" << endl;

cout << "You start your gladiator journy as a warrior skilled with swords! " << endl;

break;

case 2:

break;

case 3: gameMode = 5;

cout << endl << "---------------------------------------------------------------------------------------" << endl;

break;

default:

break;

}

if (choice == 2) {

break;

}

}

if (gameMode == 1) { // menu

//Contains navigation for the main part of the game

cout << "Actions:" << endl << "1.Start next fight" << endl << "2.Show stats" << endl << "3.Level up" << endl << "Choose:";

cin >> choice;

while (!cin.good()) {

cin.clear();

cin.ignore(256, '\n');

}

switch (choice)

{

case 1: gameMode = 2;

cout << endl << "---------------------------------------------------------------------------------------" << endl;

controlsEnabled = true;

//Initiates enemy

battlePhase = 2;

accumulatedWaitTime = 0;

enemyLevel = enemyLevel + 1;

enemyHealth = enemyLevel\*2 +4;

enemyAttackDamage = enemyLevel;

diffGetAttackReadyTime = 100 \* enemyLevel;

diffWaitAfterAttack = 100 \* enemyLevel;

//Ends enemy initiation

playerCurrentHealth = playerMaxHealth;

cout << endl << "Your opponent ( level " << enemyLevel << ") approches you get ready!" << endl;

Sleep(1000);

break;

case 2: gameMode = 3;

cout << endl << "---------------------------------------------------------------------------------------" << endl;

break;

case 3: gameMode = 4;

cout << endl << "---------------------------------------------------------------------------------------" << endl;

break;

default:

break;

}

}

if (gameMode == 2) { // Battle

//This section contains the whole combat scene! Mouse controls enemy health and interaction.

Sleep(waitTime); // dictates how fast ticks are and how fast the game runs

accumulatedWaitTime = waitTime + accumulatedWaitTime;

switch (battlePhase)

{

case 1:

if ((baseGetAttackReadyTime - diffGetAttackReadyTime + agility) <= accumulatedWaitTime) { // happens after enemy finished getting ready

playerHasHit = true;

accumulatedWaitTime = 0;

if (blockIsActive == true) { // this if else contains the enemy attack calculation

cout << "You blocked his attack!" << endl;

lockProgress[1] = 0; lockProgress[0] = 0;// this prevents buggy behaviour with the block move

}

else {

blockIsActive = true;

lockProgress[1] = 0;lockProgress[0] = 0;// this prevents buggy behaviour with the block move

playerCurrentHealth = playerCurrentHealth - enemyAttackDamage;

cout << "He hit you for " << enemyAttackDamage << " damage!" << endl << "Health:" << playerCurrentHealth << "/" << playerMaxHealth << endl;

if (playerCurrentHealth <= 0) {

gameMode = 6;

cout << endl << "---------------------------------------------------------------------------------------" << endl;

}

}

battlePhase = 2;

}

break;

case 2:

if ((baseWaitAfterAttack - diffWaitAfterAttack + manuver) <= accumulatedWaitTime) { //happens after enemy finsihed waiting after attack also the start battle phase

accumulatedWaitTime = 0;

battlePhase = 1;

GetRandExsposedPlace(exsposedPlace, exsposedPlaceText); // gets random exsposed place for the enemy

cout << "Your enemy prepares to ATTACK you!" << endl << "He exsposes his " << exsposedPlaceText << " side!" << endl;

blockIsActive = false;

playerHasHit = false;

}

break;

default:

break;

}

if (controlsEnabled) { // Control section

oldCursor = cursor;

GetCursorPos(&cursor);

if (NE(cursor, oldCursor, checkTime)) { //North East detection mouse

//cout << "Labi augsa" << endl;

lockNumber = 1;

if ( !playerHasHit) {

playerHasHit = true;

if ((exsposedPlace == 1)) {

ExecuteHit(enemyHealth, playerAttackDamage,playerHasHit);

if (enemyHealth <= 0) {

WinSituation(playerExp, enemyLevel, gameMode); // Handles win situation

}

}

else {

cout << "\*MISS\*" << endl;

}

}

}

if (NW(cursor, oldCursor, checkTime)) {//North East detection mouse

// cout << "Kreisi augsa" << endl;

lockNumber = 4;

if (!playerHasHit) {

playerHasHit = true;

if ((exsposedPlace == 2)) {

ExecuteHit(enemyHealth, playerAttackDamage, playerHasHit);

if (enemyHealth <= 0) {

WinSituation(playerExp, enemyLevel, gameMode);// Handles win situation

}

}

else {

cout << "\*MISS\*" << endl;

}

}

}

if (SE(cursor, oldCursor, checkTime)) {//North East detection mouse

// cout << "Labi leja" << endl;

lockNumber = 2;

if (!playerHasHit) {

playerHasHit = true;

if ((exsposedPlace == 4)) {

ExecuteHit(enemyHealth, playerAttackDamage, playerHasHit);

if (enemyHealth <= 0) {

WinSituation(playerExp, enemyLevel, gameMode);// Handles win situation

}

}

else {

cout << "\*MISS\*" << endl;

}

}

}

if (SW(cursor, oldCursor, checkTime)) {//North East detection mouse

// cout << "Kreisi Leja" << endl;

lockNumber = 3;

if (!playerHasHit) {

playerHasHit = true;

if ((exsposedPlace == 3)) {

ExecuteHit(enemyHealth, playerAttackDamage, playerHasHit);

if (enemyHealth <= 0) {

WinSituation(playerExp, enemyLevel, gameMode);// Handles win situation

}

}

else {

cout << "\*MISS\*" << endl;

}

}

}

lockProgress[0] = LockProgression(lockNumber, circleLock, lockProgress, 0);

lockProgress[1] = LockProgression(lockNumber, circleLock, lockProgress, 1);

if (CheckIfAplis(lockProgress)) {

//cout << "Aplis" << endl;

lockProgress[1] = 0;

lockProgress[0] = 0;

if (!blockIsActive) {

blockIsActive = true;

cout << "\*BLOCK\*" << endl;

}

}

}

}

if (gameMode == 3) { // Stats

//This section just purely displays text nothing else happens here

cout << "PlayerInfo:" << endl;

cout << "Player health:" << playerMaxHealth << endl;

cout << "Player damage:" << playerAttackDamage << endl;

cout << "Player agility modifier:" << agility << endl;

cout << "Player manuver modifier:" << manuver << endl;

cout << "NextEnemyInfo:" << endl;

cout << "Enemy level:" << enemyLevel << endl;

cout << "Enemy attack damage:" << enemyLevel << endl;

cout << "Time for enemy to attack:" <<baseGetAttackReadyTime - 100 \* enemyLevel << endl;

cout << "Time for enemy to recove after attack:" << baseWaitAfterAttack - 100 \* enemyLevel << endl;

cout << "1.Back" << endl << "Choose:";

cin >> choice;

while (!cin.good()) {

cin.clear();

cin.ignore(256, '\n');

}

switch (choice)

{

case 1:

gameMode = 1;

cout << endl << "---------------------------------------------------------------------------------------" << endl;

break;

default:

break;

}

}

if (gameMode == 4) { // Leveling

cout << "You have " << playerExp << "xp" << endl;

upgrHealthCost = playerMaxHealth \* 1;

upgrDamageCost = playerAttackDamage \* 3 +10;

upgrAgilityCost = agility / 100 +10;

upgrManuverCost = manuver / 100 +10;

cout << "1.Upgr health cost: " <<upgrHealthCost <<"xp" << endl;

cout << "2.Upgr damage cost: " << upgrDamageCost << "xp" << endl;

cout << "3.Upgr agility cost: " << upgrAgilityCost << "xp" << endl;

cout << "4.Upgr manuver cost: " << upgrManuverCost << "xp" << endl;

cout << "5.Back" << endl << "Choose:";

cin >> choice;

while (!cin.good()) {

cin.clear();

cin.ignore(256, '\n');

}

switch (choice) // This switch handles the actual variable change for each choice

{

case 1:

if (playerExp >= upgrHealthCost) {

playerMaxHealth++;

playerExp = playerExp - upgrHealthCost;

}

break;

case 2:

if (playerExp >= upgrDamageCost) {

playerAttackDamage++;

playerExp = playerExp - upgrDamageCost;

}

break;

case 3:

if (playerExp >= upgrAgilityCost) {

agility = agility + 100;

playerExp = playerExp - upgrAgilityCost;

}

break;

case 4:

if (playerExp >= upgrManuverCost) {

manuver = manuver + 100;

playerExp = playerExp - upgrManuverCost;

}

break;

case 5: gameMode = 1;

cout << endl << "---------------------------------------------------------------------------------------" << endl;

break;

default:

break;

}

}

if (gameMode == 5) { // Tutorial screen

//This section only shows text nothing else

ShowTutorialText(); // Displays tutorial text

cout << "1.Back" << endl << "Choose:";

cin >> choice;

while (!cin.good()) {

cin.clear();

cin.ignore(256, '\n');

}

switch (choice)

{

case 1:

gameMode = 0;

cout << endl << "---------------------------------------------------------------------------------------" << endl;

break;

default:

break;

}

}

if (gameMode == 6) { // Game over screen

//This is the section that the player is sent to when he dies!

cout << "You have DIED in glory!" << endl;

cout << "Your only option now is to exit the game!" << endl;

cout << "PRESS 1 TO EXIT" << endl;

cin >> choice;

while (!cin.good()) {

cin.clear();

cin.ignore(256, '\n');

}

switch (choice)

{

default:

break;

}

if (choice == 1) {

break;

}

}

}

}