**Main.cpp**

#include <cstdlib>

#include <ctime>

#include <sstream>

#include <string>

#include <fstream>

#include "SDL.h"

#include "SDL\_mixer.h"

#include "SDL\_ttf.h"

#include "resource.h"

#include "game.h"

#include "menu.h"

#include "options.h"

#include "level\_select.h"

#include "string.h"

#include "level.h"

SDL\_Surface \*screen = NULL;

SDL\_Surface \*medalSprites = NULL;

SDL\_Surface \*blockSprites = NULL;

SDL\_Surface \*message = NULL;

SDL\_Surface \*paddleSprites = NULL;

SDL\_Surface \*backgroundImg = NULL;

Mix\_Chunk \*noteA = NULL;

Mix\_Chunk \*noteB = NULL;

Mix\_Chunk \*noteC = NULL;

Mix\_Chunk \*noteD = NULL;

Mix\_Chunk \*noteE = NULL;

Mix\_Chunk \*noteF = NULL;

Mix\_Chunk \*noteG = NULL;

Mix\_Chunk \*blockKick = NULL;

Mix\_Chunk \*wall1 = NULL;

Mix\_Chunk \*wall2 = NULL;

Mix\_Chunk \*wall3 = NULL;

Mix\_Chunk \*wall4 = NULL;

Mix\_Chunk \*wall5 = NULL;

Mix\_Chunk \*vBlock1 = NULL;

Mix\_Chunk \*vBlock2 = NULL;

Mix\_Chunk \*dockL = NULL;

Mix\_Chunk \*dockR = NULL;

Mix\_Chunk \*paddleBounce = NULL;

Mix\_Chunk \*paddleLockA = NULL;

Mix\_Chunk \*paddleLockB = NULL;

Mix\_Chunk \*paddleLockC = NULL;

Mix\_Chunk \*paddleLockD = NULL;

Mix\_Chunk \*paddleLockE = NULL;

Mix\_Chunk \*paddleLockF = NULL;

Mix\_Chunk \*paddleLockG = NULL;

Mix\_Chunk \*noteBlockA = NULL;

Mix\_Chunk \*noteBlockB = NULL;

Mix\_Chunk \*noteBlockC = NULL;

Mix\_Chunk \*noteBlockD = NULL;

Mix\_Chunk \*noteBlockE = NULL;

Mix\_Chunk \*noteBlockF = NULL;

Mix\_Chunk \*noteBlockG = NULL;

Mix\_Chunk \*paddleA = NULL;

Mix\_Chunk \*paddleB = NULL;

Mix\_Chunk \*paddleC = NULL;

Mix\_Chunk \*paddleD = NULL;

Mix\_Chunk \*paddleE = NULL;

//VOICE NOTES

Mix\_Chunk \*welcome = NULL;

Mix\_Chunk \*playgame = NULL;

Mix\_Chunk \*breakout = NULL;

Mix\_Chunk \*virtuo = NULL;

Mix\_Chunk \*virtuoR = NULL;

Mix\_Chunk \*spaceForOption = NULL;

Mix\_Chunk \*spaceForEnter = NULL;

Mix\_Chunk \*downFoOption = NULL;

Mix\_Chunk \*levelselect = NULL;

Mix\_Chunk \*currentRuleIs = NULL;

Mix\_Chunk \*optionsMenu = NULL;

Mix\_Chunk \*instructions = NULL;

Mix\_Chunk \*advance13 = NULL;

Mix\_Chunk \*advance2 = NULL;

Mix\_Chunk \*HSvirtuo = NULL;

Mix\_Chunk \*HSvirtuoR = NULL;

Mix\_Chunk \*VHSis = NULL;

Mix\_Chunk \*BHSis = NULL;

Mix\_Chunk \*VRHSis = NULL;

Mix\_Chunk \*credits = NULL;

Mix\_Chunk \*selectBall = NULL;

Mix\_Chunk \*selectPaddle = NULL;

Mix\_Chunk \*selectGame = NULL;

Mix\_Chunk \*resetSpace = NULL;

Mix\_Chunk \*rightArrow = NULL;

Mix\_Chunk \*high = NULL;

Mix\_Chunk \*low = NULL;

Mix\_Chunk \*medium = NULL;

Mix\_Chunk \*returnToMain = NULL;

Mix\_Chunk \*levelNum = NULL;

Mix\_Chunk \*one = NULL;

Mix\_Chunk \*two = NULL;

Mix\_Chunk \*three = NULL;

Mix\_Chunk \*four = NULL;

Mix\_Chunk \*five = NULL;

Mix\_Chunk \*six = NULL;

Mix\_Chunk \*seven = NULL;

Mix\_Chunk \*eight = NULL;

Mix\_Chunk \*nine = NULL;

Mix\_Chunk \*ten = NULL;

Mix\_Chunk \*eleven = NULL;

Mix\_Chunk \*twelve = NULL;

Mix\_Chunk \*thirteen = NULL;

Mix\_Chunk \*fourteen = NULL;

Mix\_Chunk \*fifteen = NULL;

Mix\_Chunk \*sixteen = NULL;

Mix\_Chunk \*seventeen = NULL;

Mix\_Chunk \*eighteen = NULL;

Mix\_Chunk \*ninteen = NULL;

Mix\_Chunk \*twenty = NULL;

Mix\_Chunk \*highSC = NULL;

TTF\_Font \*font = NULL;

int score = 10;

bool medal20 = false;

bool medalHS = false;

bool medal3L = false;

bool storymode = true;

int main(int argc, char\* args[])

{

srand((unsigned)time(0));

int bestScore[41];

std::ifstream file;

file.open("data/bestscores.txt", std::ios::in | std::ios::binary);

if (!file)

{

for (int x = 0; x < 41; x++) { bestScore[x] = 0; }

}

else

{

for (int x = 0; x < 41; x++) { file >> bestScore[x]; }

}

file.close();

if (!init())

{

SDL\_Quit();

return 6;

}

int begin\_lvl = 0;

bool storymode = true;

bool levelAssign = false;

int paddleSpeed = 1;

int ballSpeed = 1;

int gameType = 1;

bool quit = false;

Uint32 time = SDL\_GetTicks();

int gameState = 0;

medalSprites = loadImage("data/medals.png");

blockSprites = loadImage("data/BlockSprites.png");

backgroundImg = loadImage("data/GameBack.png");

welcome = Mix\_LoadWAV("data/audio/1Welcome.wav"); if (welcome == NULL){ return error("Error loading WAV file."); }

playgame = Mix\_LoadWAV("data/audio/2PlayGame.wav"); if (playgame == NULL){ return error("Error loading WAV file."); }

breakout = Mix\_LoadWAV("data/audio/BreakOut.wav"); if (breakout == NULL){ return error("Error loading WAV file."); }

virtuo = Mix\_LoadWAV("data/audio/Virtuo.wav"); if (virtuo == NULL){ return error("Error loading WAV file."); }

virtuoR = Mix\_LoadWAV("data/audio/VirtuoR.wav"); if (virtuoR == NULL){ return error("Error loading WAV file."); }

spaceForOption = Mix\_LoadWAV("data/audio/UseSpace4Option.wav"); if (spaceForOption == NULL){ return error("Error loading WAV file."); }

spaceForEnter = Mix\_LoadWAV("data/audio/UseSpace4Enter.wav"); if (spaceForEnter == NULL){ return error("Error loading WAV file."); }

downFoOption = Mix\_LoadWAV("data/audio/Down4Options.wav"); if (downFoOption == NULL){ return error("Error loading WAV file."); }

levelselect = Mix\_LoadWAV("data/audio/3LevelSelect.wav"); if (levelselect == NULL){ return error("Error loading WAV file."); }

currentRuleIs = Mix\_LoadWAV("data/audio/CurrentRulesetIs..wav"); if (currentRuleIs == NULL){ return error("Error loading WAV file."); }

optionsMenu = Mix\_LoadWAV("data/audio/4OptionsMenu.wav"); if (optionsMenu == NULL){ return error("Error loading WAV file."); }

instructions = Mix\_LoadWAV("data/audio/5Instructions.wav"); if (instructions == NULL){ return error("Error loading WAV file."); }

advance13 = Mix\_LoadWAV("data/audio/Advance13.wav"); if (advance13 == NULL){ return error("Error loading WAV file."); }

advance2 = Mix\_LoadWAV("data/audio/Advance2.wav"); if (advance2 == NULL){ return error("Error loading WAV file."); }

HSvirtuo = Mix\_LoadWAV("data/audio/HSpossibleV.wav"); if (HSvirtuo == NULL){ return error("Error loading WAV file."); }

HSvirtuoR = Mix\_LoadWAV("data/audio/HSpossibleVR.wav"); if (HSvirtuoR == NULL){ return error("Error loading WAV file."); }

VHSis = Mix\_LoadWAV("data/audio/VHSis.wav"); if (VHSis == NULL){ return error("Error loading WAV file."); }

BHSis = Mix\_LoadWAV("data/audio/BHSis.wav"); if (BHSis == NULL){ return error("Error loading WAV file."); }

VRHSis = Mix\_LoadWAV("data/audio/VRHSis.wav"); if (VRHSis == NULL){ return error("Error loading WAV file."); }

credits = Mix\_LoadWAV("data/audio/6Credits.wav"); if (credits == NULL){ return error("Error loading WAV file."); }

selectBall = Mix\_LoadWAV("data/audio/selectBS.wav"); if (selectBall == NULL){ return error("Error loading WAV file."); }

selectPaddle = Mix\_LoadWAV("data/audio/selectps.wav"); if (selectPaddle == NULL){ return error("Error loading WAV file."); }

selectGame = Mix\_LoadWAV("data/audio/selectgt.wav"); if (selectGame == NULL){ return error("Error loading WAV file."); }

resetSpace = Mix\_LoadWAV("data/audio/spacereseths.wav"); if (resetSpace == NULL){ return error("Error loading WAV file."); }

rightArrow = Mix\_LoadWAV("data/audio/rightArroechangevals.wav"); if (rightArrow == NULL){ return error("Error loading WAV file."); }

high = Mix\_LoadWAV("data/audio/High.wav"); if (high == NULL){ return error("Error loading WAV file."); }

low = Mix\_LoadWAV("data/audio/Low.wav"); if (low == NULL){ return error("Error loading WAV file."); }

medium = Mix\_LoadWAV("data/audio/Medium.wav"); if (medium == NULL){ return error("Error loading WAV file."); }

returnToMain = Mix\_LoadWAV("data/audio/mainreturn.wav"); if (returnToMain == NULL){ return error("Error loading WAV file."); }

levelNum = Mix\_LoadWAV("data/audio/level..wav"); if (levelNum == NULL){ return error("Error loading WAV file."); }

one = Mix\_LoadWAV("data/audio/one.wav"); if (one == NULL){ return error("Error loading WAV file."); }

two = Mix\_LoadWAV("data/audio/two.wav"); if (two == NULL){ return error("Error loading WAV file."); }

three = Mix\_LoadWAV("data/audio/three.wav"); if (three == NULL){ return error("Error loading WAV file."); }

four = Mix\_LoadWAV("data/audio/four.wav"); if (four == NULL){ return error("Error loading WAV file."); }

five = Mix\_LoadWAV("data/audio/five.wav"); if (five == NULL){ return error("Error loading WAV file."); }

six = Mix\_LoadWAV("data/audio/six.wav"); if (six == NULL){ return error("Error loading WAV file."); }

seven = Mix\_LoadWAV("data/audio/seven.wav"); if (seven == NULL){ return error("Error loading WAV file."); }

eight = Mix\_LoadWAV("data/audio/eight.wav"); if (eight == NULL){ return error("Error loading WAV file."); }

nine = Mix\_LoadWAV("data/audio/nine.wav"); if (nine == NULL){ return error("Error loading WAV file."); }

ten = Mix\_LoadWAV("data/audio/ten.wav"); if (ten == NULL){ return error("Error loading WAV file."); }

eleven = Mix\_LoadWAV("data/audio/eleven.wav"); if (eleven == NULL){ return error("Error loading WAV file."); }

twelve = Mix\_LoadWAV("data/audio/twelve.wav"); if (twelve == NULL){ return error("Error loading WAV file."); }

thirteen = Mix\_LoadWAV("data/audio/thirteen.wav"); if (thirteen == NULL){ return error("Error loading WAV file."); }

fourteen = Mix\_LoadWAV("data/audio/fourteen.wav"); if (fourteen == NULL){ return error("Error loading WAV file."); }

fifteen = Mix\_LoadWAV("data/audio/fifteen.wav"); if (fifteen == NULL){ return error("Error loading WAV file."); }

sixteen = Mix\_LoadWAV("data/audio/sisteen.wav"); if (sixteen == NULL){ return error("Error loading WAV file."); }

seventeen = Mix\_LoadWAV("data/audio/sevteen.wav"); if (seventeen == NULL){ return error("Error loading WAV file."); }

eighteen = Mix\_LoadWAV("data/audio/eighteen.wav"); if (eighteen == NULL){ return error("Error loading WAV file."); }

ninteen = Mix\_LoadWAV("data/audio/ninteen.wav"); if (ninteen == NULL){ return error("Error loading WAV file."); }

twenty = Mix\_LoadWAV("data/audio/twenty.wav"); if (twenty == NULL){ return error("Error loading WAV file."); }

highSC = Mix\_LoadWAV("data/audio/highsc.wav"); if (highSC == NULL){ return error("Error loading WAV file."); }

//Note Loadings

noteA = Mix\_LoadWAV("data/audio/StringNoteA.wav"); if (noteA == NULL){ return error("Error loading WAV file. [note\_A]"); }

noteB = Mix\_LoadWAV("data/audio/StringNoteB.wav"); if (noteB == NULL){ return error("Error loading WAV file. [note\_B]"); }

noteC = Mix\_LoadWAV("data/audio/StringNoteC.wav"); if (noteC == NULL){ return error("Error loading WAV file. [note\_C]"); }

noteD = Mix\_LoadWAV("data/audio/StringNoteD.wav"); if (noteD == NULL){ return error("Error loading WAV file. [note\_D]"); }

noteE = Mix\_LoadWAV("data/audio/StringNoteE.wav"); if (noteE == NULL){ return error("Error loading WAV file. [note\_E]"); }

noteF = Mix\_LoadWAV("data/audio/StringNoteF.wav"); if (noteF == NULL){ return error("Error loading WAV file. [note\_F]"); }

noteG = Mix\_LoadWAV("data/audio/StringNoteG.wav"); if (noteG == NULL){ return error("Error loading WAV file. [note\_G]"); }

wall1 = Mix\_LoadWAV("data/audio/WallBipE.wav"); if (wall1 == NULL){ return error("Error loading WAV file. [wall\_1]"); }

wall2 = Mix\_LoadWAV("data/audio/WallBipFL.wav"); if (wall2 == NULL){ return error("Error loading WAV file. [wall\_2]"); }

wall3 = Mix\_LoadWAV("data/audio/WallBipFR.wav"); if (wall3 == NULL){ return error("Error loading WAV file. [wall\_3]"); }

wall4 = Mix\_LoadWAV("data/audio/WallBipCL.wav"); if (wall4 == NULL){ return error("Error loading WAV file. [wall\_4]"); }

wall5 = Mix\_LoadWAV("data/audio/WallBipCR.wav"); if (wall5 == NULL){ return error("Error loading WAV file. [wall\_5]"); }

blockKick = Mix\_LoadWAV("data/audio/BlockKick.wav"); if (blockKick == NULL){ return error("Error loading WAV file. [block\_Kick]"); }

vBlock1 = Mix\_LoadWAV("data/audio/VirtuoBang1.wav"); if (vBlock1 == NULL){ return error("Error loading WAV file. [vBlock\_1]"); }

vBlock2 = Mix\_LoadWAV("data/audio/VirtuoBang2.wav"); if (vBlock2 == NULL){ return error("Error loading WAV file. [vBlock\_2]"); }

dockL = Mix\_LoadWAV("data/audio/LDock.wav"); if (dockL == NULL){ return error("Error loading WAV file. [dock\_L]"); }

dockR = Mix\_LoadWAV("data/audio/RDock.wav"); if (dockR == NULL){ return error("Error loading WAV file. [dock\_R]"); }

paddleBounce = Mix\_LoadWAV("data/audio/Paddle\_Bounce.wav"); if (paddleBounce == NULL){ return error("Error loading WAV file. [Paddle\_Bounce]"); }

paddleLockA = Mix\_LoadWAV("data/audio/PaddleA.wav"); if (paddleLockA == NULL){ return error("Error loading WAV file. [paddleA]"); }

paddleLockB = Mix\_LoadWAV("data/audio/PaddleB.wav"); if (paddleLockB == NULL){ return error("Error loading WAV file. [paddleB]"); }

paddleLockC = Mix\_LoadWAV("data/audio/PaddleC.wav"); if (paddleLockC == NULL){ return error("Error loading WAV file. [paddleC]"); }

paddleLockD = Mix\_LoadWAV("data/audio/PaddleD.wav"); if (paddleLockD == NULL){ return error("Error loading WAV file. [paddleD]"); }

paddleLockE = Mix\_LoadWAV("data/audio/PaddleE.wav"); if (paddleLockE == NULL){ return error("Error loading WAV file. [paddleE]"); }

paddleLockF = Mix\_LoadWAV("data/audio/PaddleF.wav"); if (paddleLockF == NULL){ return error("Error loading WAV file. [paddleF]"); }

paddleLockG = Mix\_LoadWAV("data/audio/PaddleG.wav"); if (paddleLockG == NULL){ return error("Error loading WAV file. [paddleG]"); }

noteBlockA = Mix\_LoadWAV("data/audio/NoteblockA.wav"); if (noteBlockA == NULL){ return error("Error loading WAV file. [noteBlock\_A]"); }

noteBlockB = Mix\_LoadWAV("data/audio/NoteblockB.wav"); if (noteBlockB == NULL){ return error("Error loading WAV file. [noteBlock\_B]"); }

noteBlockC = Mix\_LoadWAV("data/audio/NoteblockC.wav"); if (noteBlockC == NULL){ return error("Error loading WAV file. [noteBlock\_C]"); }

noteBlockD = Mix\_LoadWAV("data/audio/NoteblockD.wav"); if (noteBlockD == NULL){ return error("Error loading WAV file. [noteBlock\_D]"); }

noteBlockE = Mix\_LoadWAV("data/audio/NoteblockE.wav"); if (noteBlockE == NULL){ return error("Error loading WAV file. [noteBlock\_E]"); }

noteBlockF = Mix\_LoadWAV("data/audio/NoteblockF.wav"); if (noteBlockF == NULL){ return error("Error loading WAV file. [noteBlock\_F]"); }

noteBlockG = Mix\_LoadWAV("data/audio/NoteblockG.wav"); if (noteBlockG == NULL){ return error("Error loading WAV file. [noteBlock\_G]"); }

menu mainMenu;

game thisGame(bestScore);

level\_select seLev;

options optionMenu;

while (!quit)

{

time = SDL\_GetTicks();

SDL\_ShowCursor(SDL\_DISABLE);

if (gameState == 0)//Menu

{

gameState = mainMenu.show(gameType, storymode);

if (gameType == 1)

{

score = 10;

}

else

{

score = 0;

}

}

else if (gameState == 1)//Play

{

gameState = thisGame.gameLoop(begin\_lvl, storymode, paddleSpeed, ballSpeed, gameType, bestScore, medalHS, medal3L, medal20, levelAssign);

}

else if (gameState == 2)//Level Select Menu

{

gameState = seLev.show(begin\_lvl, storymode, bestScore);

if (gameType == 1)

{

score = 10;

}

else

{

score = 0;

}

}

else if (gameState == 3)//Options Menu

{

gameState = optionMenu.show(paddleSpeed, ballSpeed, gameType, bestScore);

}

else if (gameState == 4)//Instructions

{

gameState = 0;

}

else if (gameState == 5)//Credits

{

gameState = 0;

}

else if (gameState == 6)//Quit

{

quit = true;

}

if ((SDL\_GetTicks() - time) < 1000 / 60)

{

SDL\_Delay((1000 / 60) - (SDL\_GetTicks() - time));

}

}

close();

Mix\_CloseAudio();

TTF\_Quit();

SDL\_Quit();

return 0;

}

**Resource.cpp**

#include "SDL.h"

#include "SDL\_image.h"

#include "SDL\_mixer.h"

#include "SDL\_ttf.h"

#include <string>

#include <iostream>

#include <fstream>

#include "resource.h"

SDL\_Surface \*loadImage(std::string filename)

{

SDL\_Surface\* LoadedImg = NULL;

SDL\_Surface\* OptImg = NULL;

LoadedImg = IMG\_Load(filename.c\_str());

if (LoadedImg != NULL)

{

OptImg = SDL\_DisplayFormat(LoadedImg);

SDL\_FreeSurface(LoadedImg);

}

else

{

error("Could Not Open " + filename);

}

return OptImg;

}

void blit(int x, int y, SDL\_Surface\* source, SDL\_Surface\* destination, SDL\_Rect\* clip)

{

SDL\_Rect rect;

rect.x = x;

rect.y = y;

SDL\_BlitSurface(source, clip, destination, &rect);

}

void printText(int x, int y, std::string words, int red, int green, int blue)

{

SDL\_Color color = {red, green, blue};

message = TTF\_RenderText\_Solid(font, words.c\_str(), color);

blit(x, y, message, screen);

SDL\_FreeSurface(message);

}

bool init()

{

if (SDL\_Init(SDL\_INIT\_EVERYTHING) == -1)

{

return false;

}

screen = SDL\_SetVideoMode(520, 640, 32, SDL\_SWSURFACE);

if (screen == NULL) { return false; }

if (Mix\_OpenAudio(44100, MIX\_DEFAULT\_FORMAT, 2, 4096) == -1) { return false; }

if( TTF\_Init() == -1 ) { return false; }

SDL\_Surface\* icon = loadImage("data/icon.png");

SDL\_WM\_SetIcon(icon, NULL);

SDL\_WM\_SetCaption("Virtuo : Breakout Remake for the Visually Impaired", NULL);

SDL\_FreeSurface(icon);

return true;

}

void close()

{

if (welcome != NULL) Mix\_FreeChunk(welcome);

if (playgame != NULL) Mix\_FreeChunk(playgame);

if (breakout != NULL) Mix\_FreeChunk(breakout);

if (virtuo != NULL) Mix\_FreeChunk(virtuo);

if (virtuoR != NULL) Mix\_FreeChunk(virtuoR);

if (spaceForOption != NULL) Mix\_FreeChunk(spaceForOption);

if (spaceForEnter != NULL) Mix\_FreeChunk(spaceForEnter);

if (downFoOption != NULL) Mix\_FreeChunk(downFoOption);

if (levelselect != NULL) Mix\_FreeChunk(levelselect);

if (currentRuleIs != NULL) Mix\_FreeChunk(currentRuleIs);

if (optionsMenu != NULL) Mix\_FreeChunk(optionsMenu);

if (instructions != NULL) Mix\_FreeChunk(instructions);

if (advance13 != NULL) Mix\_FreeChunk(advance13);

if (advance2 != NULL) Mix\_FreeChunk(advance2);

if (HSvirtuo != NULL) Mix\_FreeChunk(HSvirtuo);

if (HSvirtuoR != NULL) Mix\_FreeChunk(HSvirtuoR);

if (VHSis != NULL) Mix\_FreeChunk(VHSis);

if (BHSis != NULL) Mix\_FreeChunk(BHSis);

if (VRHSis != NULL) Mix\_FreeChunk(VRHSis);

if (credits != NULL) Mix\_FreeChunk(credits);

if (selectBall != NULL) Mix\_FreeChunk(selectBall);

if (selectPaddle != NULL) Mix\_FreeChunk(selectPaddle);

if (selectGame != NULL) Mix\_FreeChunk(selectGame);

if (resetSpace != NULL) Mix\_FreeChunk(resetSpace);

if (rightArrow != NULL) Mix\_FreeChunk(rightArrow);

if (high != NULL) Mix\_FreeChunk(high);

if (low != NULL) Mix\_FreeChunk(low);

if (medium != NULL) Mix\_FreeChunk(medium);

if (returnToMain != NULL) Mix\_FreeChunk(returnToMain);

if (levelNum != NULL) Mix\_FreeChunk(levelNum);

if (one != NULL) Mix\_FreeChunk(one);

if (two != NULL) Mix\_FreeChunk(two);

if (three != NULL) Mix\_FreeChunk(three);

if (four != NULL) Mix\_FreeChunk(four);

if (five != NULL) Mix\_FreeChunk(five);

if (six != NULL) Mix\_FreeChunk(six);

if (seven != NULL) Mix\_FreeChunk(seven);

if (eight != NULL) Mix\_FreeChunk(eight);

if (nine != NULL) Mix\_FreeChunk(nine);

if (ten != NULL) Mix\_FreeChunk(ten);

if (eleven != NULL) Mix\_FreeChunk(eleven);

if (twelve != NULL) Mix\_FreeChunk(twelve);

if (thirteen != NULL) Mix\_FreeChunk(thirteen);

if (fourteen != NULL) Mix\_FreeChunk(fourteen);

if (fifteen != NULL) Mix\_FreeChunk(fifteen);

if (sixteen != NULL) Mix\_FreeChunk(sixteen);

if (seventeen != NULL) Mix\_FreeChunk(seventeen);

if (eighteen != NULL) Mix\_FreeChunk(eighteen);

if (ninteen != NULL) Mix\_FreeChunk(ninteen);

if (twenty != NULL) Mix\_FreeChunk(twenty);

if (highSC != NULL) Mix\_FreeChunk(highSC);

if ( noteA != NULL) Mix\_FreeChunk(noteA);

if ( noteB != NULL) Mix\_FreeChunk(noteB);

if ( noteC != NULL) Mix\_FreeChunk(noteC);

if ( noteD != NULL) Mix\_FreeChunk(noteD);

if ( noteE != NULL) Mix\_FreeChunk(noteE);

if ( noteF != NULL) Mix\_FreeChunk(noteF);

if ( noteG != NULL) Mix\_FreeChunk(noteG);

if (blockKick != NULL) Mix\_FreeChunk(blockKick);

if (wall1 != NULL) Mix\_FreeChunk(wall1);

if (wall2 != NULL) Mix\_FreeChunk(wall2);

if (wall3 != NULL) Mix\_FreeChunk(wall3);

if (wall4 != NULL) Mix\_FreeChunk(wall4);

if (wall5 != NULL) Mix\_FreeChunk(wall5);

if ( vBlock1 != NULL) Mix\_FreeChunk(vBlock1);

if ( vBlock2 != NULL) Mix\_FreeChunk(vBlock2);

if ( dockL != NULL) Mix\_FreeChunk(dockL);

if ( dockR != NULL) Mix\_FreeChunk(dockR);

if (paddleBounce != NULL) Mix\_FreeChunk(paddleBounce);

if (paddleLockA != NULL) Mix\_FreeChunk(paddleLockA);

if (paddleLockB != NULL) Mix\_FreeChunk(paddleLockB);

if (paddleLockC != NULL) Mix\_FreeChunk(paddleLockC);

if (paddleLockD != NULL) Mix\_FreeChunk(paddleLockD);

if (paddleLockE != NULL) Mix\_FreeChunk(paddleLockE);

if (paddleLockF != NULL) Mix\_FreeChunk(paddleLockF);

if (paddleLockG != NULL) Mix\_FreeChunk(paddleLockG);

if ( noteBlockA != NULL) Mix\_FreeChunk(noteBlockA);

if ( noteBlockB != NULL) Mix\_FreeChunk(noteBlockB);

if ( noteBlockC != NULL) Mix\_FreeChunk(noteBlockC);

if ( noteBlockD != NULL) Mix\_FreeChunk(noteBlockD);

if ( noteBlockE != NULL) Mix\_FreeChunk(noteBlockE);

if ( noteBlockF != NULL) Mix\_FreeChunk(noteBlockD);

if ( noteBlockB != NULL) Mix\_FreeChunk(noteBlockE);

if (medalSprites != NULL) SDL\_FreeSurface(medalSprites);

if (blockSprites != NULL) SDL\_FreeSurface(blockSprites);

Mix\_CloseAudio();

TTF\_Quit();

SDL\_Quit();

}

int error(std::string error)

{

std::ofstream file;

file.open("error.txt", std::ios::out | std::ios::trunc);

file << error;

file << "Game Error.";

file.close();

close();

return 6;

}

**Resource.h**

#ifndef RESOURCE\_H

#define RESOURCE\_H

#include "SDL.h"

#include "SDL\_image.h"

#include "SDL\_mixer.h"

#include "SDL\_ttf.h"

#include <string>

extern SDL\_Surface \*medalSprites;

extern SDL\_Surface \*blockSprites;

extern SDL\_Surface \*screen;

extern SDL\_Surface \*backgroundImg;

extern SDL\_Surface \*message;

//Voice clips

extern Mix\_Chunk \*welcome;

extern Mix\_Chunk \*playgame;

extern Mix\_Chunk \*breakout;

extern Mix\_Chunk \*virtuo;

extern Mix\_Chunk \*virtuoR;

extern Mix\_Chunk \*spaceForOption;

extern Mix\_Chunk \*spaceForEnter;

extern Mix\_Chunk \*downFoOption;

extern Mix\_Chunk \*levelselect;

extern Mix\_Chunk \*currentRuleIs;

extern Mix\_Chunk \*optionsMenu;

extern Mix\_Chunk \*instructions;

extern Mix\_Chunk \*advance13;

extern Mix\_Chunk \*advance2;

extern Mix\_Chunk \*HSvirtuo;

extern Mix\_Chunk \*HSvirtuoR;

extern Mix\_Chunk \*VHSis;

extern Mix\_Chunk \*BHSis;

extern Mix\_Chunk \*VRHSis;

extern Mix\_Chunk \*credits;

extern Mix\_Chunk \*selectBall;

extern Mix\_Chunk \*selectPaddle;

extern Mix\_Chunk \*selectGame;

extern Mix\_Chunk \*resetSpace;

extern Mix\_Chunk \*rightArrow;

extern Mix\_Chunk \*high;

extern Mix\_Chunk \*low;

extern Mix\_Chunk \*medium;

extern Mix\_Chunk \*returnToMain;

extern Mix\_Chunk \*levelNum;

extern Mix\_Chunk \*one;

extern Mix\_Chunk \*two;

extern Mix\_Chunk \*three;

extern Mix\_Chunk \*four;

extern Mix\_Chunk \*five;

extern Mix\_Chunk \*six;

extern Mix\_Chunk \*seven;

extern Mix\_Chunk \*eight;

extern Mix\_Chunk \*nine;

extern Mix\_Chunk \*ten;

extern Mix\_Chunk \*eleven;

extern Mix\_Chunk \*twelve;

extern Mix\_Chunk \*thirteen;

extern Mix\_Chunk \*fourteen;

extern Mix\_Chunk \*fifteen;

extern Mix\_Chunk \*sixteen;

extern Mix\_Chunk \*seventeen;

extern Mix\_Chunk \*eighteen;

extern Mix\_Chunk \*ninteen;

extern Mix\_Chunk \*twenty;

extern Mix\_Chunk \*highSC;

extern Mix\_Chunk \*noteA;

extern Mix\_Chunk \*noteB;

extern Mix\_Chunk \*noteC;

extern Mix\_Chunk \*noteD;

extern Mix\_Chunk \*noteE;

extern Mix\_Chunk \*noteF;

extern Mix\_Chunk \*noteG;

extern Mix\_Chunk \*wall1;

extern Mix\_Chunk \*wall2;

extern Mix\_Chunk \*wall3;

extern Mix\_Chunk \*wall4;

extern Mix\_Chunk \*wall5;

extern Mix\_Chunk \*blockKick;

extern Mix\_Chunk \*vBlock1;

extern Mix\_Chunk \*vBlock2;

extern Mix\_Chunk \*dockL;

extern Mix\_Chunk \*dockR;

extern Mix\_Chunk \*paddleBounce;

extern Mix\_Chunk \*paddleLockA;

extern Mix\_Chunk \*paddleLockB;

extern Mix\_Chunk \*paddleLockC;

extern Mix\_Chunk \*paddleLockD;

extern Mix\_Chunk \*paddleLockE;

extern Mix\_Chunk \*paddleLockF;

extern Mix\_Chunk \*paddleLockG;

extern Mix\_Chunk \*noteBlockA;

extern Mix\_Chunk \*noteBlockB;

extern Mix\_Chunk \*noteBlockC;

extern Mix\_Chunk \*noteBlockD;

extern Mix\_Chunk \*noteBlockE;

extern Mix\_Chunk \*noteBlockF;

extern Mix\_Chunk \*noteBlockG;

extern TTF\_Font \*font;

extern int score;

SDL\_Surface \*loadImage(std::string filename);

void blit(int x, int y, SDL\_Surface\* source, SDL\_Surface\* destination, SDL\_Rect\* clip = NULL);

void printText(int x, int y, std::string words, int r, int g, int b);

bool init();

void close();

int error(std::string error);

#endif

**Menu.cpp**

#include "menu.h"

#include <string>

#include "SDL.h"

#include "SDL\_ttf.h"

#include "resource.h"

menu::menu()

{

back = loadImage("data/menuBack.png");

bgclip.x = 0;

bgclip.y = 0;

bgclip.w = 520;

bgclip.h = 640;

bgrect.x = 0;

bgrect.y = 0;

option[0].x = 521;

option[0].y = 0;

option[0].w = 387;

option[0].h = 68;

option[1].x = 521;

option[1].y = 77;

option[1].w = 387;

option[1].h = 68;

option[2].x = 521;

option[2].y = 154;

option[2].w = 387;

option[2].h = 68;

option[3].x = 521;

option[3].y = 231;

option[3].w = 387;

option[3].h = 68;

option[4].x = 521;

option[4].y = 308;

option[4].w = 387;

option[4].h = 68;

rect[0].x = 0;

rect[0].y = 233;

rect[1].x = 0;

rect[1].y = 310;

rect[2].x = 0;

rect[2].y = 387;

rect[3].x = 0;

rect[3].y = 464;

rect[4].x = 0;

rect[4].y = 541;

}

menu::~menu()

{

SDL\_FreeSurface(back);

}

int menu::show(int gameType, bool &story)

{

story = true;

if (playIntro == 0)

{

Mix\_PlayChannel(1, welcome, 0);

playIntro = 1;

}

if (!Mix\_Playing(1) && playIntro == 1)

{

Mix\_PlayChannel(1, downFoOption, 0);

playIntro = 2;

}

if (!Mix\_Playing(1) && playIntro == 2)

{

Mix\_PlayChannel(1, spaceForEnter, 0);

playIntro = 3;

}

if ((currentOpt == 0) && (!Mix\_Playing(1)))

{

if (playPlay == 1)

{

Mix\_PlayChannel(1, currentRuleIs, 0);

playPlay = 2;

}

else if (playPlay == 2)

{

if (gameType == 1)

{

Mix\_PlayChannel(1, virtuo, 0);

playPlay = 0;

}

else if (gameType == 2)

{

Mix\_PlayChannel(1, breakout, 0);

playPlay = 0;

}

else

{

Mix\_PlayChannel(1, virtuoR, 0);

playPlay = 0;

}

}

}

if ((currentOpt == 3) && (!Mix\_Playing(1)))

{

if ((playInst == 1) && (gameType == 2))

{

Mix\_PlayChannel(1, advance2, 0);

playInst = 2;

}

else if ((playInst == 1) && ((gameType == 1) || (gameType == 3)))

{

Mix\_PlayChannel(1, advance13, 0);

playInst = 2;

}

if ((playInst == 2) && (gameType == 1))

{

Mix\_PlayChannel(1, HSvirtuo, 0);

playInst = 0;

}

else if ((playInst == 2) && (gameType == 2))

{

Mix\_PlayChannel(1, HSvirtuoR, 0);

playInst = 0;

}

}

while (SDL\_PollEvent(&event))

{

if (event.type == SDL\_KEYDOWN)

{

SDLKey keypressed = event.key.keysym.sym;

switch (keypressed)

{

case SDLK\_SPACE:

{

if (currentOpt == 4)

{

Mix\_PlayChannel(-1, instructions, 0);

if (gameType == 1 || gameType == 3)

{

Mix\_PlayChannel(-1, advance13, 0);

}

else

{

Mix\_PlayChannel(-1, advance2, 0);

}

}

else if (currentOpt == 5)

{

Mix\_PlayChannel(-1, credits, 0);

}

return (currentOpt + 1);

}

case SDLK\_DOWN:

{

if (currentOpt == 0)

{

currentOpt = 1;

Mix\_PlayChannel(1, levelselect, 0);

}

else if (currentOpt == 1)

{

currentOpt = 2;

Mix\_PlayChannel(1, optionsMenu, 0);

}

else if (currentOpt == 2)

{

currentOpt = 3;

playInst = 1;

Mix\_PlayChannel(1, instructions, 0);

}

else if (currentOpt == 3)

{

currentOpt = 4;

Mix\_PlayChannel(1, credits, 0);

}

else if (currentOpt == 4)

{

currentOpt = 0;

playPlay = 1;

Mix\_PlayChannel(1, playgame, 0);

}

}

}

}

if (event.type == SDL\_QUIT)

{

return 6;

}

else if (event.type == SDL\_KEYDOWN)

{

if (event.key.keysym.sym == SDLK\_ESCAPE)

{

return 6;

}

}

}

SDL\_BlitSurface(back, &bgclip, screen, &bgrect);

SDL\_BlitSurface(back, &option[currentOpt], screen, &rect[currentOpt]);

SDL\_Flip(screen);

return 0;

}

**Menu.h**

#ifndef MENU\_H

#define MENU\_H

#include <string>

#include <iostream>

#include "SDL.h"

class menu

{

public:

menu();

~menu();

int show(int gameType, bool &story);

private:

SDL\_Rect option[5];

SDL\_Rect rect[5];

SDL\_Rect bgclip;

SDL\_Rect bgrect;

SDL\_Event event;

int currentOpt = 0;

int playIntro = 0;

int playInst = 0;

int playPlay = 0;

SDL\_Surface \*back;

};

#endif

**Options.cpp**

#include "options.h"

#include <sstream>

#include <string>

#include <iostream>

#include <fstream>

#include <string>

#include "SDL.h"

#include "SDL\_ttf.h"

#include "resource.h"

using namespace std;

options::options()

{

//load image

back = loadImage("data/OptBack.png");

//clip for background

bgclip.x = 0;

bgclip.y = 0;

bgclip.w = 520;

bgclip.h = 640;

bgrect.x = 0;

bgrect.y = 0;

//clips for option select

option[0].x = 0;

option[0].y = 641;

option[0].w = 149;

option[0].h = 34;

option[1].x = 150;

option[1].y = 641;

option[1].w = 149;

option[1].h = 34;

option[2].x = 300;

option[2].y = 641;

option[2].w = 149;

option[2].h = 34;

option[3].x = 0;

option[3].y = 676;

option[3].w = 149;

option[3].h = 34;

option[4].x = 150;

option[4].y = 676;

option[4].w = 149;

option[4].h = 34;

option[5].x = 300;

option[5].y = 676;

option[5].w = 149;

option[5].h = 34;

option[6].x = 0;

option[6].y = 711;

option[6].w = 149;

option[6].h = 34;

option[7].x = 150;

option[7].y = 711;

option[7].w = 149;

option[7].h = 34;

option[8].x = 300;

option[8].y = 711;

option[8].w = 149;

option[8].h = 34;

option[9].x = 0;

option[9].y = 746;

option[9].w = 149;

option[9].h = 34;

option[10].x = 150;

option[10].y = 746;

option[10].w = 149;

option[10].h = 34;

rect[0].x = 32;

rect[0].y = 166;

rect[1].x = 185;

rect[1].y = 166;

rect[2].x = 338;

rect[2].y = 166;

rect[3].x = 32;

rect[3].y = 275;

rect[4].x = 185;

rect[4].y = 275;

rect[5].x = 338;

rect[5].y = 275;

rect[6].x = 32;

rect[6].y = 384;

rect[7].x = 185;

rect[7].y = 384;

rect[8].x = 338;

rect[8].y = 384;

rect[9].x = 32;

rect[9].y = 461;

rect[10].x = 304;

rect[10].y = 579;

}

options::~options()

{

//clean up

SDL\_FreeSurface(back);

}

int options::show(int &paddleSpeed, int &ballSpeed, int &gameType, int(&bestscore)[41])

{

if (playIntro == 0)

{

Mix\_PlayChannel(1, selectBall, 0);

playIntro = 1;

}

if (!Mix\_Playing(1) && playIntro == 1)

{

Mix\_PlayChannel(1, downFoOption, 0);

playIntro = 2;

}

if (!Mix\_Playing(1) && playIntro == 2)

{

Mix\_PlayChannel(1, rightArrow, 0);

playIntro = 3;

}

while(SDL\_PollEvent(&event))

{

if (event.type == SDL\_KEYDOWN)

{

SDLKey keypressed = event.key.keysym.sym;

switch (keypressed)

{

case SDLK\_SPACE:

{

if (currOpt == 10)

{

return 0;

}

if (currOpt == 9)

{

for (int x = 0; x < 41; x++)

{

bestscore[x] = 0;

}

std::ofstream file("data/bestscores.txt");

if (file.is\_open())

{

for (int count = 0; count < 41; count++)

{

file << bestscore[count] << " ";

}

file.close();

}

}

}

case SDLK\_DOWN:

{

if (currOpt < 3)

{

currOpt = 3;

Mix\_PlayChannel(1, selectPaddle, 0);

}

else if (currOpt < 6)

{

currOpt = 6;

Mix\_PlayChannel(1, selectGame, 0);

}

else if (currOpt < 9)

{

currOpt = 9;

Mix\_PlayChannel(1, resetSpace, 0);

}

else if (currOpt == 9)

{

currOpt++;

Mix\_PlayChannel(1, returnToMain, 0);

}

else if (currOpt == 10)

{

currOpt = 0;

Mix\_PlayChannel(-1, selectBall, 0);

}

break;

}

case SDLK\_RIGHT:

{

if (currOpt < 3)

{

if (currOpt == 0)

{

Mix\_PlayChannel(1, medium, 0);

}

else if (currOpt == 1)

{

Mix\_PlayChannel(1, high, 0);

}

else if (currOpt == 2)

{

Mix\_PlayChannel(1, low, 0);

}

currOpt++;

currOpt = currOpt % 3;

paddleSpeed = currOpt + 1;

}

else if (currOpt < 6)

{

if (currOpt == 3)

{

Mix\_PlayChannel(1, medium, 0);

}

else if (currOpt == 4)

{

Mix\_PlayChannel(1, high, 0);

}

else if (currOpt == 5)

{

Mix\_PlayChannel(1, low, 0);

}

currOpt++;

currOpt = currOpt % 3;

currOpt = currOpt + 3;

ballSpeed = (currOpt % 3) + 1;

}

else if (currOpt < 9)

{

if (currOpt == 6)

{

Mix\_PlayChannel(1, breakout, 0);

}

else if (currOpt == 7)

{

Mix\_PlayChannel(1, virtuoR, 0);

}

else if (currOpt == 8)

{

Mix\_PlayChannel(1, virtuo, 0);

}

currOpt++;

currOpt = currOpt % 3;

currOpt = currOpt + 6;

gameType = (currOpt % 3) + 1;

if (gameType != 1)

{

score = 0;

}

}

}

if (event.type == SDL\_QUIT)

{

return 0; //return STATE\_EXIT

}

else if (event.type == SDL\_KEYDOWN)

{

if (event.key.keysym.sym == SDLK\_ESCAPE)

{

return 0; //return STATE\_EXIT

}

}

}

}

}

//update screen

SDL\_BlitSurface(back, &bgclip, screen, &bgrect);

SDL\_BlitSurface(back, &option[currOpt], screen, &rect[currOpt]);

SDL\_Flip(screen);

return 3;

}

**Options.h**

#ifndef OPTIONS\_H

#define OPTIONS\_H

#include <string>

#include <iostream>

#include "SDL.h"

class options

{

public:

options();

~options();

int show(int &paddleSpeed, int &ballSpeed, int &gameType, int (&bestscore)[41]);

private:

SDL\_Rect option[11];

SDL\_Rect rect[11];

SDL\_Rect bgclip;

SDL\_Rect bgrect;

SDL\_Event event;

int currOpt;

int playIntro;

SDL\_Surface \*back;

};

#endif

**Block.cpp**

#include "SDL.h"

#include "resource.h"

#include "block.h"

#include "standard\_block.h"

#include "heavy\_block.h"

#include "note\_block.h"

#include "titan\_block.h"

#include "virtuo\_block.h"

block::block()

{

activated = true;

inplay = false;

alive = false;

}

block::block(int x, int y, int gameType)

{

gameForm = gameType;

rect.x = x; //block's coordinates

rect.y = y;

}

void block::update(int bX, int bY)

{

}

void block::draw()

{

if (alive)

{

SDL\_BlitSurface(blockSprites, &clip, screen, &rect);

}

}

int block::myX()

{

if (alive)

return rect.x;

else

return 0;

}

int block::myY()

{

if (alive)

return rect.y;

else

return 0;

}

bool block::activeNote()

{

if (activated)

{

return true;

}

else

{

return false;

}

}

bool block::breakoutValid()

{

if (inplay)

{

return false;

}

else

{

return true;

}

}

bool block::gone()

{

if (alive)

{

return false;

}

else

{

return true;

}

}

void block::establish()

{

activated = true;

inplay = false;

alive = false;

}

void block::clearBlock()

{

alive = false;

}

**Block.h**

#ifndef BLOCK\_H

#define BLOCK\_H

#include "SDL.h"

class block

{

public:

block();

block(int x, int y, int gameType);

virtual void update(int bX, int bY);

void draw();

int myX();

int myY();

bool activeNote();

bool breakoutValid();

bool gone();

virtual void establish();

void clearBlock();

protected:

int gameForm;

bool activated;

bool inplay;

bool damaged = false;

bool colDet = false;

bool alive;

SDL\_Rect rect;

SDL\_Rect clip;

};

#endif

**Virtuo\_block.cpp**

#include "SDL.h"

#include "resource.h"

#include "virtuo\_block.h"

#include "block.h"

using namespace std;

virtuo\_block::virtuo\_block()

{

alive = false;

}

virtuo\_block::virtuo\_block(int x, int y, int gameType)

{

gameForm = gameType;

alive = true;

inplay = true;

activated = true;

if (gameForm == 1)

{

clip.x = 0;

clip.y = 0;

clip.w = 36;

clip.h = 20;

}

else if (gameForm == 2)

{

clip.x = 148;

clip.y = 84;

clip.w = 36;

clip.h = 20;

}

else

{

clip.x = 148;

clip.y = 42;

clip.w = 36;

clip.h = 20;

}

rect.x = x; //block's coordinates

rect.y = y;

}

void virtuo\_block::update(int bX, int bY)

{

if (alive)

{

if(((bX + 7 >= rect.x) && (bX <= rect.x + 36)) && ((bY + 7 >= rect.y) && (bY <= rect.y + 20))) //ball collision

{

if (!damaged)

{

damaged = true;

//sprite change

if (gameForm == 1)

{

clip.x = 37;

clip.y = 0;

clip.w = 36;

clip.h = 20;

}

else if (gameForm == 2)

{

alive = false;

inplay = false;

score = score + 3;

}

else

{

Mix\_PlayChannel(-1, vBlock2, 0);

alive = false;

inplay = false;

score++;

}

}

else if ((damaged) && (gameForm == 1))

{

Mix\_PlayChannel(-1, vBlock1, 0);

alive = false;

inplay = false;

score--;

}

}

}

}

void virtuo\_block::establish()

{

alive = true;

inplay = true;

activated = true;

}

**Virtuo\_block.h**

#ifndef VIRTUO\_BLOCK\_H

#define VIRTUO\_BLOCK\_H

#include "SDL.h"

#include "block.h"

#include "standard\_block.h"

#include "heavy\_block.h"

#include "note\_block.h"

#include "titan\_block.h"

#include "virtuo\_block.h"

class virtuo\_block : public block

{

public:

virtuo\_block();

virtuo\_block(int x, int y, int gameType);

void update(int bX, int bY);

void establish();

};

#endif

**Titan\_block.cpp**

#include "SDL.h"

#include "resource.h"

#include "block.h"

#include "standard\_block.h"

#include "heavy\_block.h"

#include "note\_block.h"

#include "titan\_block.h"

#include "virtuo\_block.h"

titan\_block::titan\_block()

{

alive = false;

inplay = false;

activated = true;

}

titan\_block::titan\_block(int x, int y, int gameType)

{

alive = true;

inplay = false;

activated = true;

clip.x = 148;

clip.y = 0;

clip.w = 36;

clip.h = 20;

rect.x = x; //block's coordinates

rect.y = y;

}

void titan\_block::update(int bX, int bY)

{

if (alive)

{

if (((bX + 7 >= rect.x) && (bX <= rect.x + 36)) && ((bY + 7 >= rect.y) && (bY <= rect.y + 20))) //ball collision

{

//play music

}

}

}

void titan\_block::establish()

{

alive = true;

inplay = false;

activated = true;

}

**Titan\_block.h**

#ifndef TITAN\_BLOCK\_H

#define TITAN\_BLOCK\_H

#include "SDL.h"

#include "block.h"

class titan\_block : public block

{

public:

titan\_block();

titan\_block(int x, int y, int gameType);

void update(int bX, int bY);

void establish();

};

#endif

**Note\_block.cpp**

#include "SDL.h"

#include "resource.h"

#include "block.h"

#include "standard\_block.h"

#include "heavy\_block.h"

#include "note\_block.h"

#include "titan\_block.h"

#include "virtuo\_block.h"

note\_block::note\_block()

{

alive = false;

activated = true;

inplay = false;

}

note\_block::note\_block(int x, int y, int gameType, int note)

{

gameForm = gameType;

activated = false;

inplay = false;

alive = true;

myNote = note;

if (myNote == 0)// myNote A

{

clip.x = 0;

clip.y = 21;

clip.w = 36;

clip.h = 20;

}

else if (myNote == 1)// myNote B

{

clip.x = 0;

clip.y = 42;

clip.w = 36;

clip.h = 20;

}

else if (myNote == 2)// myNote C

{

clip.x = 0;

clip.y = 63;

clip.w = 36;

clip.h = 20;

}

else if (myNote == 3)// myNote D

{

clip.x = 0;

clip.y = 84;

clip.w = 36;

clip.h = 20;

}

else if (myNote == 4)// myNote E

{

clip.x = 0;

clip.y = 105;

clip.w = 36;

clip.h = 20;

}

else if (myNote == 5)// myNote F

{

clip.x = 0;

clip.y = 126;

clip.w = 36;

clip.h = 20;

}

else if (myNote == 6)// myNote G

{

clip.x = 0;

clip.y = 147;

clip.w = 36;

clip.h = 20;

}

rect.x = x; //block's coordinates

rect.y = y;

}

void note\_block::update(int bX, int bY)

{

if (alive)

{

if (((bX + 7 >= rect.x) && (bX <= rect.x + 36)) && ((bY + 7 >= rect.y) && (bY <= rect.y + 20))) //ball collision

{

if (damaged == false)

{

damaged = true;

activated = true;

if (gameForm == 2)

{

score = score + 5;

}

if (myNote == 0)// note A

{

clip.x = 37;

clip.y = 21;

clip.w = 36;

clip.h = 20;

Mix\_PlayChannel(-1, noteBlockA, 0);

}

else if (myNote == 1)// note B

{

clip.x = 37;

clip.y = 42;

clip.w = 36;

clip.h = 20;

Mix\_PlayChannel(-1, noteBlockB, 0);

}

else if (myNote == 2)// myNote C

{

clip.x = 37;

clip.y = 63;

clip.w = 36;

clip.h = 20;

Mix\_PlayChannel(-1, noteBlockC, 0);

}

else if (myNote == 3)// myNote D

{

clip.x = 37;

clip.y = 84;

clip.w = 36;

clip.h = 20;

Mix\_PlayChannel(-1, noteBlockD, 0);

}

else if (myNote == 4)// myNote E

{

clip.x = 37;

clip.y = 105;

clip.w = 36;

clip.h = 20;

Mix\_PlayChannel(-1, noteBlockE, 0);

}

else if (myNote == 5)// myNote F

{

clip.x = 37;

clip.y = 126;

clip.w = 36;

clip.h = 20;

Mix\_PlayChannel(-1, noteBlockF, 0);

}

else if (myNote == 6)// myNote G

{

clip.x = 37;

clip.y = 147;

clip.w = 36;

clip.h = 20;

Mix\_PlayChannel(-1, noteBlockG, 0);

}

}

}

}

}

void note\_block::establish()

{

activated = false;

inplay = false;

alive = true;

}

**Note\_block.h**

#ifndef NOTE\_BLOCK\_H

#define NOTE\_BLOCK\_H

#include "SDL.h"

#include "block.h"

class note\_block : public block

{

public:

note\_block();

note\_block(int x, int y, int gameType, int note);

void update(int bX, int bY);

void draw();

int myX();

int myY();

bool gone();

void makeAlive();

void establish();

bool active;

int myNote;

};

#endif

**Heavy\_block.cpp**

#include "SDL.h"

#include "resource.h"

#include "heavy\_block.h"

#include "block.h"

heavy\_block::heavy\_block()

{

alive = false;

inplay = false;

activated = true;

}

heavy\_block::heavy\_block(int x, int y, int gameType)

{

gameForm = gameType;

alive = true;

activated = true;

inplay = true;

clip.x = 111;

clip.y = 0;

clip.w = 36;

clip.h = 20;

rect.x = x; //block's coordinates

rect.y = y;

}

void heavy\_block::update(int bX, int bY)

{

if (alive)

{

if (((bX + 7 >= rect.x) && (bX <= rect.x + 36)) && ((bY + 7 >= rect.y) && (bY <= rect.y + 20))) //ball collision

{

if (!damaged)

{

damaged = true;

//sprite change

clip.x = 148;

clip.y = 21;

clip.w = 36;

clip.h = 20;

}

else if (damaged)

{

if (gameForm == 2)

{

score = score + 2;

}

inplay = false;

alive = false;

}

}

}

}

void heavy\_block::establish()

{

alive = true;

activated = true;

inplay = true;

}

**Heavy\_block.h**

#ifndef HEAVY\_BLOCK\_H

#define HEAVY\_BLOCK\_H

#include "SDL.h"

#include "block.h"

#include "standard\_block.h"

#include "heavy\_block.h"

#include "note\_block.h"

#include "titan\_block.h"

#include "virtuo\_block.h"

class heavy\_block : public block

{

public:

heavy\_block();

heavy\_block(int x, int y, int gameType);

void update(int bX, int bY);

void establish();

};

#endif

**Standard\_block.cpp**

#include "SDL.h"

#include "resource.h"

#include "block.h"

#include "standard\_block.h"

#include "heavy\_block.h"

#include "note\_block.h"

#include "titan\_block.h"

#include "virtuo\_block.h"

using namespace std;

standard\_block::standard\_block()

{

alive = false;

inplay = false;

activated = true;

}

standard\_block::standard\_block(int x, int y, int gameType)

{

gameForm = gameType;

alive = true;

inplay = true;

activated = true;

clip.x = 74;

clip.y = 0;

clip.w = 36;

clip.h = 20;

rect.x = x; //block's coordinates

rect.y = y;

}

void standard\_block::update(int bX, int bY)

{

if (alive)

{

if (((bX + 7 >= rect.x) && (bX <= rect.x + 36)) && ((bY + 7 >= rect.y) && (bY <= rect.y + 20))) //ball collision

{

if (gameForm == 2)

{

score++;

}

alive = false;

inplay = false;

}

}

}

void standard\_block::establish()

{

alive = true;

inplay = true;

activated = true;

}

**Standard\_block.h**

#ifndef STANDARD\_BLOCK\_H

#define STANDARD\_BLOCK\_H

#include "SDL.h"

#include "block.h"

class standard\_block : public block

{

public:

standard\_block();

standard\_block(int x, int y, int gameType);

void update(int bX, int bY);

void establish();

};

#endif

**Level\_select.cpp**

#include <sstream>

#include <fstream>

#include <string>

#include "level\_select.h"

#include "game.h"

#include "SDL.h"

#include "SDL\_ttf.h"

#include "resource.h"

using namespace std;

level\_select::level\_select()

{

//load image

back = loadImage("data/LvlBack.png");

//clip for background

bgclip.x = 0;

bgclip.y = 0;

bgclip.w = 520;

bgclip.h = 640;

bgrect.x = 0;

bgrect.y = 0;

//clips for play select

levelclip[0].x = 0;

levelclip[0].y = 641;

levelclip[0].w = 64;

levelclip[0].h = 33;

levelclip[1].x = 65;

levelclip[1].y = 641;

levelclip[1].w = 64;

levelclip[1].h = 33;

//create array for urect possitions [73(+96)x382(+50)]

levelrect[0].x = 73;

levelrect[0].y = 296;

levelrect[1].x = 73;

levelrect[1].y = 346;

levelrect[2].x = 73;

levelrect[2].y = 396;

levelrect[3].x = 73;

levelrect[3].y = 446;

levelrect[4].x = 175;

levelrect[4].y = 296;

levelrect[5].x = 175;

levelrect[5].y = 346;

levelrect[6].x = 175;

levelrect[6].y = 396;

levelrect[7].x = 175;

levelrect[7].y = 446;

levelrect[8].x = 276;

levelrect[8].y = 296;

levelrect[9].x = 276;

levelrect[9].y = 346;

levelrect[10].x = 276;

levelrect[10].y = 396;

levelrect[11].x = 276;

levelrect[11].y = 446;

levelrect[12].x = 379;

levelrect[12].y = 296;

levelrect[13].x = 379;

levelrect[13].y = 346;

levelrect[14].x = 379;

levelrect[14].y = 396;

levelrect[15].x = 379;

levelrect[15].y = 446;

levelrect[16].x = 379;

levelrect[16].y = 496;

levelrect[17].x = 276;

levelrect[17].y = 496;

levelrect[18].x = 175;

levelrect[18].y = 496;

levelrect[19].x = 73;

levelrect[19].y = 496;

levelrect[20].x = 361;

levelrect[20].y = 581;

}

level\_select::~level\_select()

{

//clean up

SDL\_FreeSurface(back);

}

int level\_select::show(int &begin\_lvl, bool &storymode, int bestscore[41])

{

//storymode = false;

if (playIntro == 0)

{

Mix\_PlayChannel(1, downFoOption, 0);

playIntro = 1;

}

if (!Mix\_Playing(1) && playIntro == 1)

{

Mix\_PlayChannel(1, spaceForEnter, 0);

playIntro = 2;

}

while (SDL\_PollEvent(&event))

{

if (event.type == SDL\_KEYDOWN)

{

SDLKey keypressed = event.key.keysym.sym;

switch (keypressed)

{

case SDLK\_SPACE:

{

if (currOpt == 20)

{

playIntro = 0;

return 0;

}

else

{

begin\_lvl = currOpt;

storymode = false;

playIntro = 0;

return 1;

}

}

case SDLK\_DOWN:

{

if (currOpt == 19)

{

currOpt++;

Mix\_PlayChannel(1, returnToMain, 0);

}

else if (currOpt == 20)

{

currOpt = 0;

}

else

{

currOpt = currOpt++;

}

if (currOpt != 20)

{

levclip = 0;

}

else

{

levclip = 1;

}

playLevel = 0;

}

}

}

if (event.type == SDL\_QUIT)

{

return 0;

}

else if (event.type == SDL\_KEYDOWN)

{

if (event.key.keysym.sym == SDLK\_ESCAPE)

{

return 0;

}

}

}

//update screen

SDL\_BlitSurface(back, &bgclip, screen, &bgrect);

//update these inside of an array

SDL\_BlitSurface(back, &levelclip[levclip], screen, &levelrect[currOpt]);

std::stringstream out;

std::string levelOut;

out << (currOpt + 1);

levelOut = out.str();

std::stringstream out2;

std::string virtuoOut;

out2 << bestscore[currOpt];

virtuoOut = out2.str();

std::stringstream out3;

std::string virtuoXOut;

out3 << bestscore[currOpt+20];

virtuoXOut = out3.str();

std::stringstream out4;

std::string breakoutOut;

out4 << bestscore[40];

breakoutOut = out4.str();

if (currOpt != 20)

{

font = TTF\_OpenFont("data/forte.ttf", 36);

printText(140, 118, levelOut, 255, 255, 0);

font = TTF\_OpenFont("data/forte.ttf", 20);

printText(260, 178, virtuoOut, 255, 255, 255);

printText(260, 212, breakoutOut, 255, 255, 255);

printText(260, 246, virtuoXOut, 255, 255, 255);

}

if (!Mix\_Playing(1) && (playLevel == 0) && (currOpt!=20) && (playIntro == 2))

{

Mix\_PlayChannel(1, levelNum, 0);

playLevel = 1;

}

if (!Mix\_Playing(1) && playLevel == 1)

{

if (currOpt == 0)

{

Mix\_PlayChannel(1, one, 0);

playLevel = 2;

}

else if (currOpt == 1)

{

Mix\_PlayChannel(1, two, 0);

playLevel = 2;

}

else if (currOpt == 2)

{

Mix\_PlayChannel(1, three, 0);

playLevel = 2;

}

else if (currOpt == 3)

{

Mix\_PlayChannel(1, four, 0);

playLevel = 2;

}

else if (currOpt == 4)

{

Mix\_PlayChannel(1, five, 0);

playLevel = 2;

}

else if (currOpt == 5)

{

Mix\_PlayChannel(1, six, 0);

playLevel = 2;

}

else if (currOpt == 6)

{

Mix\_PlayChannel(1, seven, 0);

playLevel = 2;

}

else if (currOpt == 7)

{

Mix\_PlayChannel(1, eight, 0);

playLevel = 2;

}

else if (currOpt == 8)

{

Mix\_PlayChannel(1, nine, 0);

playLevel = 2;

}

else if (currOpt == 9)

{

Mix\_PlayChannel(1, ten, 0);

playLevel = 2;

}

else if (currOpt == 10)

{

Mix\_PlayChannel(1, eleven, 0);

playLevel = 2;

}

else if (currOpt == 11)

{

Mix\_PlayChannel(1, twelve, 0);

playLevel = 2;

}

else if (currOpt == 12)

{

Mix\_PlayChannel(1, thirteen, 0);

playLevel = 2;

}

else if (currOpt == 13)

{

Mix\_PlayChannel(1, fourteen, 0);

playLevel = 2;

}

else if (currOpt == 14)

{

Mix\_PlayChannel(1, fifteen, 0);

playLevel = 2;

}

else if (currOpt == 15)

{

Mix\_PlayChannel(1, sixteen, 0);

playLevel = 2;

}

else if (currOpt == 16)

{

Mix\_PlayChannel(1, seventeen, 0);

playLevel = 2;

}

else if (currOpt == 17)

{

Mix\_PlayChannel(1, eighteen, 0);

playLevel = 2;

}

else if (currOpt == 18)

{

Mix\_PlayChannel(1, ninteen, 0);

playLevel = 2;

}

else if (currOpt == 19)

{

Mix\_PlayChannel(1, twenty, 0);

playLevel = 2;

}

}

if (!Mix\_Playing(1) && playLevel == 2)

{

if ((bestscore[currOpt]) != 0)

{

Mix\_PlayChannel(1, VHSis, 0);

playLevel = 3;

}

else

{

playLevel = 4;

}

}

if (!Mix\_Playing(1) && playLevel == 3)

{

if (bestscore[currOpt] == 1)

{

Mix\_PlayChannel(1, one, 0); playLevel = 4;

}

else if (bestscore[currOpt] == 2)

{

Mix\_PlayChannel(1, two, 0); playLevel = 4;

}

else if (bestscore[currOpt] == 3)

{

Mix\_PlayChannel(1, three, 0); playLevel = 4;

}

else if (bestscore[currOpt] == 4)

{

Mix\_PlayChannel(1, four, 0); playLevel = 4;

}

else if (bestscore[currOpt] == 5)

{

Mix\_PlayChannel(1, five, 0); playLevel = 4;

}

else if (bestscore[currOpt] == 6)

{

Mix\_PlayChannel(1, six, 0); playLevel = 4;

}

else if (bestscore[currOpt] == 7)

{

Mix\_PlayChannel(1, seven, 0); playLevel = 4;

}

else if (bestscore[currOpt] == 8)

{

Mix\_PlayChannel(1, eight, 0); playLevel = 4;

}

else if (bestscore[currOpt] == 9)

{

Mix\_PlayChannel(1, nine, 0); playLevel = 4;

}

else if (bestscore[currOpt] == 10)

{

Mix\_PlayChannel(1, ten, 0); playLevel = 4;

}

else if (bestscore[currOpt] == 11)

{

Mix\_PlayChannel(1, eleven, 0); playLevel = 4;

}

else if (bestscore[currOpt] == 12)

{

Mix\_PlayChannel(1, twelve, 0); playLevel = 4;

}

else if (bestscore[currOpt] == 13)

{

Mix\_PlayChannel(1, thirteen, 0); playLevel = 4;

}

else if (bestscore[currOpt] == 14)

{

Mix\_PlayChannel(1, fourteen, 0); playLevel = 4;

}

else if (bestscore[currOpt] == 15)

{

Mix\_PlayChannel(1, fifteen, 0); playLevel = 4;

}

else if (bestscore[currOpt] == 16)

{

Mix\_PlayChannel(1, sixteen, 0); playLevel = 4;

}

else if (bestscore[currOpt] == 17)

{

Mix\_PlayChannel(1, seventeen, 0); playLevel = 4;

}

else if (bestscore[currOpt] == 18)

{

Mix\_PlayChannel(1, eighteen, 0); playLevel = 4;

}

else if (bestscore[currOpt] == 19)

{

Mix\_PlayChannel(1, ninteen, 0); playLevel = 4;

}

else if (bestscore[currOpt] == 20)

{

Mix\_PlayChannel(1, twenty, 0);

playLevel = 4;

}

}

if (!Mix\_Playing(1) && (playLevel == 4))

{

if ((bestscore[currOpt + 20]) != 0)

{

Mix\_PlayChannel(1, VRHSis, 0);

playLevel = 5;

}

else

{

playLevel = 6;

}

}

if (!Mix\_Playing(1) && (playLevel == 5))

{

if (bestscore[currOpt + 20] == 1)

{

Mix\_PlayChannel(1, one, 0); playLevel = 6;

}

else if (bestscore[currOpt + 20] == 2)

{

Mix\_PlayChannel(1, two, 0); playLevel = 6;

}

else if (bestscore[currOpt + 20] == 3)

{

Mix\_PlayChannel(1, three, 0); playLevel = 6;

}

else if (bestscore[currOpt + 20] == 4)

{

Mix\_PlayChannel(1, four, 0); playLevel = 6;

}

else if (bestscore[currOpt + 20] == 5)

{

Mix\_PlayChannel(1, five, 0); playLevel = 6;

}

else if (bestscore[currOpt + 20] == 6)

{

Mix\_PlayChannel(1, six, 0); playLevel = 6;

}

else if (bestscore[currOpt + 20] == 7)

{

Mix\_PlayChannel(1, seven, 0); playLevel = 6;

}

else if (bestscore[currOpt + 20] == 8)

{

Mix\_PlayChannel(1, eight, 0); playLevel = 6;

}

else if (bestscore[currOpt + 20] == 9)

{

Mix\_PlayChannel(1, nine, 0); playLevel = 6;

}

else if (bestscore[currOpt + 20] == 10)

{

Mix\_PlayChannel(1, ten, 0); playLevel = 6;

}

else if (bestscore[currOpt + 20] == 11)

{

Mix\_PlayChannel(1, eleven, 0); playLevel = 6;

}

else if (bestscore[currOpt + 20] == 12)

{

Mix\_PlayChannel(1, twelve, 0); playLevel = 6;

}

else if (bestscore[currOpt + 20] == 13)

{

Mix\_PlayChannel(1, thirteen, 0); playLevel = 6;

}

else if (bestscore[currOpt + 20] == 14)

{

Mix\_PlayChannel(1, fourteen, 0); playLevel = 6;

}

else if (bestscore[currOpt + 20] == 15)

{

Mix\_PlayChannel(1, fifteen, 0); playLevel = 6;

}

else if (bestscore[currOpt + 20] == 16)

{

Mix\_PlayChannel(1, sixteen, 0); playLevel = 6;

}

else if (bestscore[currOpt + 20] == 17)

{

Mix\_PlayChannel(1, seventeen, 0); playLevel = 6;

}

else if (bestscore[currOpt + 20] == 18)

{

Mix\_PlayChannel(1, eighteen, 0); playLevel = 6;

}

else if (bestscore[currOpt + 20] == 19)

{

Mix\_PlayChannel(1, ninteen, 0); playLevel = 6;

}

else if (bestscore[currOpt + 20] == 20)

{

Mix\_PlayChannel(1, twenty, 0); playLevel = 6;

}

playLevel = 6;

}

SDL\_Flip(screen);

return 2;

}

**Level\_select.h**

#ifndef LEVEL\_SELECT\_H

#define LEVEL\_SELECT\_H

#include <string>

#include <iostream>

#include "SDL.h"

class level\_select

{

public:

level\_select();

~level\_select();

int show(int &begin\_lvl, bool &storymode, int bestscore[41]);

private:

SDL\_Rect levelclip[2];

SDL\_Rect levelrect[21];

SDL\_Rect bgclip;

SDL\_Rect bgrect;

SDL\_Event event;

int currOpt;

int levclip;

int returned;

int playIntro = 0;

int playLevel = 0;

SDL\_Surface \*back;

};

#endif

**Ball.cpp**

#include <cstdlib>

#include <ctime>

#include "SDL.h"

#include "SDL\_mixer.h"

#include "resource.h"

#include "math.h"

#include "ball.h"

ball::ball()

{

clip.x = 115; //clip for the ball sprite in tiles.png

clip.y = 105;

clip.w = 7;

clip.h = 7;

rect.x = 290; //ball's coordinates

rect.y = 430;

xVel = rand() % 8 - 4; //random beginning velocity between -4 and 4

if (xVel == 0) //don't want the ball going straight up to start

xVel = 1;

moving = false;

moved = false;

yVel = -1;

}

ball::ball(int ballSpeed)

{

clip.x = 48; //clip for the ball sprite in tiles.png

clip.y = 37;

clip.w = 7;

clip.h = 7;

rect.x = 290; //ball's coordinates

rect.y = 430;

xVel = rand()%8-4; //random beginning velocity between -4 and 4

if (xVel == 0) //don't want the ball going straight up to start

xVel = 1;

moving = false;

moved = false;

yVel = (ballSpeed)\*(-1);

}

void ball::update(int pX, int bX, int bY)

{

if (moving)

{

if (!moved)

{

rect.x += xVel;

rect.y += yVel;

moved = true;

//Wall Collisions

if (rect.y <= 35)

{

Mix\_PlayChannel(-1, wall1, 0);

yVel = (yVel)\*(-1);

}

if (rect.x <= 5)

{

Mix\_PlayChannel(-1, wall2, 0);

xVel = (xVel)\*(-1);

rect.x = 6;

}

else if (rect.x + 7 >= 515)

{

Mix\_PlayChannel(-1, wall3, 0);

xVel = (xVel)\*(-1);

rect.x = 507;

}

if ((rect.y + 7 >= 430) && (rect.x <= 80))

{

Mix\_PlayChannel(-1, wall4, 0);

xVel = (xVel)\*(-1);

rect.x = 81;

}

if ((rect.y + 7 >= 430) && (rect.x + 7 >= 439))

{

Mix\_PlayChannel(-1, wall5, 0);

xVel = (xVel)\*(-1);

rect.x = 431;

}

if ((rect.y + 7 >= 409) && (rect.y + 7 < 429))

{

if ((rect.x >= 59) && (rect.x <= 80))

{

Mix\_PlayChannel(-1, wall2, 0);

xVel = ((sqrt(yVel\*yVel)) + 1);

yVel = -((sqrt(yVel\*yVel)) + 1);

}

else if ((rect.x + 7 >= 439) && (rect.x + 7 <= 468))

{

Mix\_PlayChannel(-1, wall3, 0);

xVel = -((sqrt(yVel\*yVel)) + 1);

yVel = -((sqrt(yVel\*yVel)) + 1);

}

else if (rect.x < 59)

{

Mix\_PlayChannel(-1, wall2, 0);

yVel = -yVel;

}

else if (rect.x + 7 > 468)

{

Mix\_PlayChannel(-1, wall3, 0);

yVel = -yVel;

}

}

//Paddle Collisions

if ((rect.x > pX - 8 && rect.x < pX + 84) && ((rect.y + 7 > 610 && rect.y + 7 < 620)))

{

yVel = -yVel;

Mix\_PlayChannel(-1, paddleBounce, 0);

if (rect.x < pX + 2) { xVel = -3.5; }

else if (rect.x < pX + 12) { xVel = -3.0; }

else if (rect.x < pX + 22) { xVel = -2.2; }

else if (rect.x < pX + 32) { xVel = -1.4; }

else if (rect.x < pX + 44) { xVel = 0; }

else if (rect.x < pX + 54) { xVel = 1.4; }

else if (rect.x < pX + 64) { xVel = 2.2; }

else if (rect.x < pX + 74) { xVel = 3.0; }

else if (rect.x < pX + 84) { xVel = 3.5; }

}

}

//String Collision

if (((rect.y + 3 >= 434) && (rect.y + 3 <= 436)) || ((rect.y + 3 > 469) && (rect.y + 3 < 471)) || ((rect.y + 3 > 504) && (rect.y + 3 < 506)) ||

((rect.y + 3 > 539) && (rect.y + 3 < 541)) || ((rect.y + 3 > 574) && (rect.y + 3 < 576)))

{

if (rect.x + 3 <= 132) { Mix\_PlayChannel(-1, noteA, 0); }

else if (rect.x + 3 <= 183) { Mix\_PlayChannel(-1, noteB, 0); }

else if (rect.x + 3 <= 234) { Mix\_PlayChannel(-1, noteC, 0); }

else if (rect.x + 3 <= 285) { Mix\_PlayChannel(-1, noteD, 0); }

else if (rect.x + 3 <= 336) { Mix\_PlayChannel(-1, noteE, 0); }

else if (rect.x + 3 <= 387) { Mix\_PlayChannel(-1, noteF, 0); }

else

{ Mix\_PlayChannel(-1, noteG, 0); }

}

//BLOCK COLLISION

if ((rect.x + 7 >= bX && rect.x < bX + 36) && (rect.y + 7 > bY && rect.y + 7 < bY + 3)) //Block Collision TOPBOTTOM

{

rect.y = bY - 8;

yVel = -yVel;

Mix\_PlayChannel(-1, blockKick, 0);

}

if ((rect.x + 7 >= bX && rect.x < bX + 36) && (rect.y >= bY + 17 && rect.y < bY + 20))

{

rect.y = bY + 21;

yVel = -yVel;

Mix\_PlayChannel(-1, blockKick, 0);

}

if ((rect.y + 7 >= bY && rect.y <= bY + 20) && ((rect.x + 7 >= bX && rect.x + 7 <= bX + 3) || (rect.x >= bX + 33 && rect.x <= bX + 36)))//Block Collision LEFTRIGHT

{

xVel = -xVel;

Mix\_PlayChannel(-1, blockKick, 0);

}

}

else

{

rect.y = 603;

rect.x = pX+31;

}

}

void ball::draw()

{

SDL\_BlitSurface(blockSprites, &clip, screen, &rect);

moved = false;

}

void ball::move()

{

moving = true;

}

void ball::reset(int ballSpeed)

{

moving = false;

xVel = rand()%8-4;

if (xVel == 0)

{

xVel = 1;

}

yVel = (ballSpeed)\*(-1);

}

int ball::myX()

{

return rect.x;

}

int ball::myY()

{

return rect.y;

}

**Ball.h**

#ifndef BALL\_H

#define BALL\_H

#include "SDL.h"

class ball

{

public:

ball();

ball(int ballSpeed);

void update(int pX, int bX, int bY);

void draw();

void move();

void reset(int ballSpeed);

int myX();

int myY();

private:

SDL\_Rect rect;

SDL\_Rect clip;

bool moving;

bool moved;

double xVel;

double yVel;

};

#endif

**Paddle.cpp**

#include "SDL.h"

#include "resource.h"

#include "options.h"

#include "paddle.h"

paddle::paddle()

{

clip.w = 83;

clip.h = 10;

rect.x = 256; //paddle's coordinates

rect.y = 610;

}

void paddle::update(int paddleSpeed)

{

Uint8 \*keystates = SDL\_GetKeyState(NULL);

if (keystates[SDLK\_LEFT])

{

if (rect.x > 81)

{

rect.x -= (paddleSpeed + 2);

}

else

{

Mix\_PlayChannel(1, dockL, 0);

}

}

if (keystates[SDLK\_RIGHT])

{

if (rect.x + 83 <= 438)

{

rect.x += (paddleSpeed + 2);

}

else

{

Mix\_PlayChannel(1, dockR, 0);

}

}

if (!Mix\_Playing(2))

{

if (rect.x + 41 <= 132)

{

Mix\_PlayChannel(-1, paddleLockA, 0);

}

else if (rect.x + 41 <= 183)

{

Mix\_PlayChannel(-1, paddleLockB, 0);

}

else if (rect.x + 41 <= 234)

{

Mix\_PlayChannel(-1, paddleLockC, 0);

}

else if (rect.x + 41 <= 285)

{

Mix\_PlayChannel(-1, paddleLockD, 0);

}

else if (rect.x + 41 <= 336)

{

Mix\_PlayChannel(-1, paddleLockE, 0);

}

else if (rect.x + 41 <= 387)

{

Mix\_PlayChannel(-1, paddleLockF, 0);

}

else

{

Mix\_PlayChannel(-1, paddleLockG, 0);

}

}

}

void paddle::draw(int gameType)

{

if (gameType == 2)

{

clip.x = 74;

clip.y = 146;

}

else if (gameType == 3)

{

clip.x = 74;

clip.y = 136;

}

else

{

clip.x = 74;

clip.y = 156;

}

SDL\_BlitSurface(blockSprites, &clip, screen, &rect);

}

int paddle::myX()

{

return rect.x;

}

**Paddle.h**

#ifndef PADDLE\_H

#define PADDLE\_H

#include "SDL.h"

class paddle

{

public:

paddle();

void update(int paddleSpeed);

void draw(int gameType);

int myX();

private:

SDL\_Rect rect;

SDL\_Rect clip;

};

#endif

**Level.cpp**

#include <string>

#include "SDL.h"

#include "SDL\_ttf.h"

#include "resource.h"

#include "paddle.h"

#include "ball.h"

#include "block.h"

#include "virtuo\_block.h"

#include "note\_block.h"

#include "titan\_block.h"

#include "heavy\_block.h"

#include "standard\_block.h"

#include "level.h"

level::level(int levelArray[14][14], int gameType, int ballSpeed)

{

//background image

bgClip.x = 0;

bgClip.y = 0;

bgClip.w = 520;

bgClip.h = 640;

//background position

bgRect.x = 0;

bgRect.y = 0;

for (int x = 0; x <= 13; x++) //make blocks

{

for (int y = 0; y <= 13; y++)

{

if (levelArray[y][x] == 0)

{

blocks.push\_back(new block()); //Empty

}

else if (levelArray[y][x] == 1)

{

blocks.push\_back(new standard\_block(8 + x \* 36, 50 + y \* 20, gameType)); //Standard

}

else if (levelArray[y][x] == 2)

{

blocks.push\_back(new heavy\_block(8 + x \* 36, 50 + y \* 20, gameType)); //Heavy

}

else if (levelArray[y][x] == 3)

{

blocks.push\_back(new titan\_block(8 + x \* 36, 50 + y \* 20, gameType)); //Titan

}

else if (levelArray[y][x] == 4)

{

blocks.push\_back(new note\_block(8 + x \* 36, 50 + y \* 20, gameType, 0)); //NoteA

}

else if (levelArray[y][x] == 5)

{

blocks.push\_back(new note\_block(8 + x \* 36, 50 + y \* 20, gameType, 1)); //NoteB

}

else if (levelArray[y][x] == 6)

{

blocks.push\_back(new note\_block(8 + x \* 36, 50 + y \* 20, gameType, 2)); //NoteC

}

else if (levelArray[y][x] == 7)

{

blocks.push\_back(new note\_block(8 + x \* 36, 50 + y \* 20, gameType, 3)); //NoteD

}

else if (levelArray[y][x] == 8)

{

blocks.push\_back(new note\_block(8 + x \* 36, 50 + y \* 20, gameType, 4)); //NoteE

}

else if (levelArray[y][x] == 9)

{

blocks.push\_back(new note\_block(8 + x \* 36, 50 + y \* 20, gameType, 5)); //NoteF

}

else if (levelArray[y][x] == 10)

{

blocks.push\_back(new note\_block(8 + x \* 36, 50 + y \* 20, gameType, 6)); //NoteG

}

else if (levelArray[y][x] == 11)

{

blocks.push\_back(new virtuo\_block(8 + x \* 36, 50 + y \* 20, gameType)); //Virtuo

}

}

}

ball ball1(ballSpeed);

}

void level::draw(int paddleSpeed, int ballSpeed, int gameType)

{

paddle1.update(paddleSpeed);

for (unsigned int x = 0; x < blocks.size(); x++)

{

ball1.update(paddle1.myX(), (\*blocks[x]).myX(), (\*blocks[x]).myY()); //update ea. ball

(\*blocks[x]).update(ball1.myX(), ball1.myY()); //and ea. block

}

SDL\_BlitSurface(backgroundImg, &bgClip, screen, &bgRect);

paddle1.draw(gameType);

ball1.draw();

for (unsigned int x = 0; x < blocks.size(); x++)

{

(\*blocks[x]).draw();

}

}

void level::update(SDL\_Event event)

{

Uint8 \*keystates = SDL\_GetKeyState(NULL);

if (keystates[SDLK\_SPACE])

{

ball1.move();

}

}

bool level::levelComplete(int gameType)

{

if (gameType == 1 || gameType == 3)

{

for (unsigned int x = 0; x < blocks.size(); x++)

{

if (!(\*blocks[x]).activeNote())

{

return false;

}

}

}

else if (gameType == 2)

{

for (unsigned int x = 0; x < blocks.size(); x++)

{

if (!(\*blocks[x]).breakoutValid())

{

return false;

}

}

}

return true;

}

bool level::dead(int ballSpeed)

{

if (ball1.myY() < 640) //if ball is not below screen

{

return false;

}

ball1.reset(ballSpeed);

return true;

}

void level::clearScreen()

{

for (unsigned int x = 0; x < blocks.size(); x++)

(\*blocks[x]).clearBlock();

}

void level::reset(int ballSpeed)

{

for (unsigned int x = 0; x < blocks.size(); x++)

{

(\*blocks[x]).establish();

}

ball1.reset(ballSpeed);

}

**Level.h**

#ifndef LEVEL\_H

#define LEVEL\_H

#include <vector>

#include <string>

#include "SDL.h"

#include "paddle.h"

#include "ball.h"

#include "block.h"

#include "standard\_block.h"

#include "heavy\_block.h"

#include "note\_block.h"

#include "titan\_block.h"

#include "virtuo\_block.h"

class level

{

public:

level(int levelArray[14][14], int gameType, int ballSpeed);

void update(SDL\_Event event);

void draw(int paddleSpeed, int ballSpeed, int gameType);

bool levelComplete(int gameType);

bool dead(int ballSpeed);

void reset(int ballSpeed);

void clearScreen();

private:

paddle paddle1;

ball ball1;

std::vector<block\*> blocks;

SDL\_Rect bgClip;

SDL\_Rect bgRect;

int backCol;

};

#endif

**Game.cpp**

#include <sstream>

#include <string>

#include <fstream>

#include "SDL.h"

#include "SDL\_ttf.h"

#include "game.h"

#include "resource.h"

#include "level.h"

#include "menu.h"

#include "level\_select.h"

#include "options.h"

#include "block.h"

#include "standard\_block.h"

#include "heavy\_block.h"

#include "note\_block.h"

#include "titan\_block.h"

#include "virtuo\_block.h"

game::game(int(&bestScore)[41])

{

//Set starting number of Lives

lives = 3;

//Clip for Medals

medalClip[0].x = 0;

medalClip[0].y = 0;

medalClip[0].h = 137;

medalClip[0].w = 135;

medalClip[1].h = 158;

medalClip[1].w = 157;

medalClip[1].y = 139;

medalClip[2].x = 137;

medalClip[2].y = 0;

medalClip[2].h = 137;

medalClip[2].w = 135;

//Rects for Medals

medalRect[0].x = 23;

medalRect[0].y = 225;

medalRect[1].x = 181;

medalRect[1].y = 177;

medalRect[2].x = 360;

medalRect[2].y = 225;

//Clip for Life Sprites

livesClip.x = 74;

livesClip.y = 105;

livesClip.w = 40;

livesClip.h = 28;

rect[0].x = 10;

rect[0].y = 5;

rect[1].x = 40;

rect[1].y = 5;

rect[2].x = 70;

rect[2].y = 5;

}

game::~game()

{

}

int game::gameLoop(int &beginLevel, bool storymode, int paddleSpeed, int ballSpeed, int gameType, int(&bestScore)[41], bool &medalHS, bool &medal3L, bool &medal20, bool &levelAssign)

{

int level\_1[14][14] = {

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 2, 1, 1, 1, 1, 1, 1, 2, 0, 0, 0 },

{ 0, 0, 11, 1, 1, 1, 1, 1, 1, 1, 1, 11, 0, 0 },

{ 0, 0, 11, 1, 1, 1, 1, 1, 1, 1, 1, 11, 0, 0 },

{ 0, 0, 11, 11, 0, 0, 0, 0, 0, 0, 11, 11, 0, 0 },

{ 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0 },

{ 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 5, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 11, 11, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 3, 1, 1, 3, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 } };

int level\_2[14][14] = {

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 11, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 11, 0 },

{ 0, 11, 11, 11, 3, 1, 1, 1, 1, 3, 11, 11, 11, 0 },

{ 0, 11, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 11, 0 },

{ 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 6, 6, 0, 0, 0, 0, 0, 0 },

{ 0, 1, 3, 0, 0, 0, 0, 0, 0, 0, 0, 3, 1, 0 },

{ 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0 },

{ 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0 },

{ 0, 0, 0, 0, 0, 2, 3, 3, 2, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 3, 1, 2, 2, 1, 3, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 } };

int level\_3[14][14] = {

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 11, 3, 2, 2, 2, 2, 2, 2, 2, 2, 3, 11, 0 },

{ 0, 0, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 0, 0 },

{ 0, 0, 3, 1, 11, 1, 2, 2, 1, 11, 1, 3, 0, 0 },

{ 0, 0, 1, 0, 0, 2, 5, 7, 2, 0, 0, 1, 0, 0 },

{ 0, 0, 11, 0, 0, 2, 0, 0, 2, 0, 0, 11, 0, 0 },

{ 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0 },

{ 0, 0, 11, 0, 0, 0, 0, 0, 0, 0, 0, 11, 0, 0 },

{ 0, 11, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 11, 0 },

{ 0, 2, 3, 0, 0, 0, 0, 0, 0, 0, 0, 3, 2, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 } };

int level\_4[14][14] = {

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0 },

{ 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0 },

{ 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0 },

{ 0, 0, 0, 1, 11, 0, 0, 0, 0, 11, 1, 0, 0, 0 },

{ 0, 0, 0, 1, 3, 0, 0, 0, 0, 3, 1, 0, 0, 0 },

{ 0, 11, 3, 1, 1, 11, 11, 11, 11, 1, 1, 3, 11, 0 },

{ 0, 11, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 11, 0 },

{ 0, 0, 2, 2, 1, 9, 1, 1, 10, 1, 2, 2, 0, 0 },

{ 0, 0, 2, 2, 1, 1, 1, 1, 1, 1, 2, 2, 0, 0 },

{ 0, 0, 3, 1, 0, 0, 0, 0, 0, 0, 1, 3, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 } };

int level\_5[14][14] = {

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0 },

{ 0, 3, 2, 3, 11, 3, 11, 11, 3, 11, 3, 2, 3, 0 },

{ 0, 0, 2, 0, 0, 6, 0, 0, 8, 0, 0, 2, 0, 0 },

{ 0, 3, 2, 0, 0, 0, 0, 0, 0, 0, 0, 2, 3, 0 },

{ 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0 },

{ 0, 3, 2, 11, 1, 0, 0, 0, 0, 1, 11, 2, 3, 0 },

{ 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0 },

{ 0, 0, 0, 11, 11, 0, 0, 0, 0, 11, 11, 0, 0, 0 },

{ 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 } };

int level\_6[14][14] = {

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 11, 11, 11, 11, 0, 0, 0, 0, 0 },

{ 0, 2, 2, 3, 0, 1, 1, 1, 1, 0, 3, 2, 2, 0 },

{ 0, 0, 11, 2, 0, 0, 1, 1, 0, 0, 2, 11, 0, 0 },

{ 0, 0, 0, 2, 0, 0, 1, 5, 0, 0, 2, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 4, 2, 2, 0, 0, 0, 0, 0, 0, 2, 2, 7, 0 },

{ 0, 0, 0, 2, 11, 3, 0, 0, 3, 11, 2, 0, 0, 0 },

{ 0, 0, 0, 2, 1, 1, 1, 1, 1, 1, 2, 0, 0, 0 },

{ 0, 0, 0, 0, 11, 3, 1, 1, 3, 11, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 } };

int level\_7[14][14] = {

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 2, 9, 0, 0, 0, 0, 0, 0 },

{ 0, 11, 11, 11, 0, 0, 0, 0, 0, 0, 11, 11, 11, 0 },

{ 0, 0, 1, 1, 11, 0, 0, 0, 0, 11, 1, 1, 0, 0 },

{ 0, 0, 0, 1, 1, 11, 0, 0, 11, 1, 1, 0, 0, 0 },

{ 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 3, 0, 0, 3, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0 },

{ 0, 0, 5, 2, 2, 2, 1, 1, 2, 2, 2, 5, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 } };

int level\_8[14][14] = {

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 1, 1, 11, 11, 1, 1, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 1, 10, 1, 1, 3, 1, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 11, 0, 0, 0, 0, 0, 0, 0, 0, 11, 0, 0 },

{ 0, 0, 3, 0, 0, 3, 11, 11, 3, 0, 0, 3, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 6, 1, 1, 1, 0, 0, 1, 1, 1, 10, 0, 0 },

{ 0, 0, 1, 11, 11, 1, 0, 0, 1, 11, 11, 1, 0, 0 },

{ 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0 },

{ 0, 0, 3, 0, 0, 3, 0, 0, 3, 0, 0, 3, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 } };

int level\_9[14][14] = {

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 1, 1, 1, 10, 0, 0, 4, 1, 1, 1, 0, 0 },

{ 0, 0, 0, 1, 1, 11, 0, 0, 11, 1, 1, 0, 0, 0 },

{ 0, 0, 0, 0, 1, 3, 0, 0, 3, 1, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 1, 11, 0, 0, 11, 1, 0, 0, 0, 0 },

{ 0, 0, 0, 3, 3, 3, 0, 0, 3, 3, 3, 0, 0, 0 },

{ 0, 1, 1, 11, 11, 11, 0, 0, 11, 11, 11, 1, 1, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 3, 1, 1, 3, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 3, 7, 1, 3, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 } };

int level\_10[14][14] = {

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 2, 11, 11, 11, 11, 11, 2, 0 },

{ 0, 0, 0, 0, 0, 0, 2, 3, 2, 3, 2, 3, 2, 0 },

{ 0, 0, 2, 3, 2, 3, 2, 1, 1, 1, 1, 1, 2, 0 },

{ 0, 2, 1, 11, 1, 2, 2, 1, 1, 1, 1, 1, 2, 0 },

{ 0, 3, 1, 0, 1, 3, 2, 1, 1, 1, 1, 1, 2, 0 },

{ 0, 2, 1, 0, 1, 2, 2, 1, 1, 4, 1, 1, 2, 0 },

{ 0, 0, 0, 0, 0, 2, 2, 11, 1, 0, 1, 11, 2, 0 },

{ 0, 0, 0, 0, 0, 0, 11, 3, 1, 0, 1, 3, 11, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 2, 4, 0, 7, 2, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 3, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 } };

int level\_11[14][14] = {

{ 0, 0, 0, 3, 1, 3, 1, 1, 3, 1, 3, 0, 0, 0 },

{ 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0 },

{ 0, 0, 1, 11, 0, 0, 0, 0, 0, 0, 11, 1, 0, 0 },

{ 0, 0, 0, 0, 0, 1, 11, 11, 1, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 3, 1, 1, 3, 0, 0, 0, 0, 0 },

{ 0, 4, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 7, 0 },

{ 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0 },

{ 0, 2, 2, 2, 0, 0, 0, 0, 0, 0, 2, 2, 2, 0 },

{ 0, 0, 0, 11, 1, 5, 1, 1, 4, 1, 11, 0, 0, 0 },

{ 0, 0, 0, 0, 11, 1, 1, 1, 1, 11, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 11, 1, 1, 11, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 } };

int level\_12[14][14] = {

{ 0, 0, 0, 3, 2, 3, 2, 3, 2, 3, 2, 0, 0, 0 },

{ 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0 },

{ 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 3, 2, 3, 2 },

{ 0, 0, 0, 1, 0, 0, 6, 6, 0, 0, 0, 0, 0, 3 },

{ 0, 11, 11, 1, 0, 0, 11, 11, 0, 0, 0, 0, 0, 2 },

{ 2, 1, 1, 3, 0, 0, 11, 11, 0, 0, 3, 0, 0, 3 },

{ 3, 0, 0, 0, 0, 0, 5, 4, 0, 0, 2, 0, 0, 2 },

{ 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 3 },

{ 3, 0, 0, 3, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0 },

{ 2, 0, 0, 2, 3, 2, 11, 2, 3, 2, 3, 0, 0, 0 },

{ 3, 0, 0, 3, 11, 11, 11, 1, 0, 0, 0, 0, 0, 0 },

{ 2, 0, 0, 2, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0 },

{ 3, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 } };

int level\_13[14][14] = {

{ 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3 },

{ 2, 1, 2, 1, 11, 2, 1, 1, 2, 11, 1, 2, 1, 2 },

{ 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 3, 11, 1, 11, 11, 1, 11, 3, 0, 0, 0 },

{ 0, 0, 0, 2, 3, 9, 8, 10, 9, 3, 2, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 2, 0, 0, 0, 1, 2, 2, 1, 0, 0, 0, 2, 0 },

{ 0, 3, 11, 11, 0, 1, 0, 0, 1, 0, 11, 11, 3, 0 },

{ 0, 0, 1, 2, 0, 1, 0, 0, 1, 0, 2, 1, 0, 0 },

{ 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 } };

int level\_14[14][14] = {

{ 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 2, 2, 2, 3, 3, 2, 2, 3, 0, 0, 0 },

{ 0, 0, 2, 2, 1, 1, 6, 6, 1, 1, 2, 2, 0, 0 },

{ 0, 0, 1, 2, 0, 0, 0, 0, 0, 0, 2, 1, 0, 0 },

{ 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0 },

{ 3, 1, 8, 0, 0, 1, 2, 2, 1, 0, 0, 1, 1, 3 },

{ 2, 0, 0, 0, 0, 2, 11, 11, 2, 0, 0, 0, 0, 2 },

{ 3, 0, 0, 0, 3, 2, 11, 11, 2, 3, 0, 0, 0, 3 },

{ 2, 1, 1, 1, 2, 11, 11, 11, 11, 2, 1, 1, 1, 2 },

{ 3, 0, 0, 0, 3, 11, 0, 0, 11, 3, 0, 0, 0, 8 },

{ 0, 0, 0, 0, 2, 0, 0, 0, 0, 2, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 3, 0, 0, 0, 0, 3, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 } };

int level\_15[14][14] = {

{ 0, 0, 0, 11, 1, 11, 1, 1, 11, 1, 11, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 1, 1, 1, 0, 11, 1, 1, 11, 0, 1, 1, 1, 0 },

{ 0, 1, 5, 1, 0, 0, 0, 0, 0, 0, 1, 4, 1, 0 },

{ 0, 1, 1, 1, 0, 0, 11, 11, 0, 0, 1, 1, 1, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 1, 1, 1, 11, 6, 7, 11, 1, 1, 1, 0, 0 },

{ 0, 0, 0, 1, 1, 2, 2, 2, 2, 1, 1, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 1, 3, 0, 0, 0, 0, 0, 0, 0, 0, 3, 1, 0 },

{ 0, 3, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 3, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 } };

int level\_16[14][14] = {

{ 0, 0, 0, 0, 1, 2, 0, 0, 2, 1, 0, 0, 0, 0 },

{ 0, 0, 1, 1, 9, 2, 0, 0, 2, 8, 1, 1, 0, 0 },

{ 0, 2, 2, 2, 2, 2, 0, 0, 2, 2, 2, 2, 2, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 11, 0, 0, 0, 0, 11, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 11, 0, 0, 0, 0, 11, 10, 0, 0, 0 },

{ 0, 0, 11, 11, 11, 0, 0, 0, 0, 11, 11, 11, 0, 0 },

{ 0, 1, 2, 1, 0, 0, 0, 0, 0, 0, 1, 2, 1, 0 },

{ 0, 3, 2, 7, 0, 0, 0, 0, 0, 0, 7, 2, 3, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 } };

int level\_17[14][14] = {

{ 0, 0, 0, 0, 0, 3, 0, 0, 3, 0, 0, 0, 0, 0 },

{ 0, 2, 0, 2, 0, 2, 2, 2, 2, 0, 2, 0, 2, 0 },

{ 0, 3, 11, 3, 11, 3, 6, 2, 3, 11, 3, 11, 3, 0 },

{ 0, 2, 2, 2, 2, 3, 0, 0, 3, 2, 2, 2, 2, 0 },

{ 0, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 0 },

{ 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0 },

{ 0, 3, 2, 2, 0, 0, 0, 0, 0, 0, 2, 2, 3, 0 },

{ 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0 },

{ 0, 3, 1, 1, 1, 11, 0, 0, 11, 1, 1, 1, 3, 0 },

{ 0, 2, 2, 2, 4, 0, 0, 0, 0, 4, 2, 2, 2, 0 },

{ 0, 3, 4, 0, 0, 0, 0, 0, 0, 0, 0, 7, 3, 0 },

{ 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0 },

{ 0, 11, 11, 0, 0, 0, 0, 0, 0, 0, 0, 11, 11, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 } };

int level\_18[14][14] = {

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 3, 3, 3, 3, 3, 3, 0, 0, 0, 0 },

{ 0, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0 },

{ 0, 0, 9, 1, 1, 1, 1, 1, 1, 1, 1, 3, 0, 0 },

{ 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0 },

{ 0, 2, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 9, 0 },

{ 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 3, 1, 3, 3, 1, 3, 0, 0, 0, 0 },

{ 0, 0, 1, 2, 2, 2, 2, 2, 2, 2, 2, 8, 0, 0 },

{ 0, 0, 0, 0, 11, 11, 11, 11, 11, 11, 0, 0, 0, 0 },

{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 },

{ 10, 11, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 11, 4 },

{ 0, 0, 0, 0, 1, 3, 11, 11, 3, 1, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 } };

int level\_19[14][14] = {

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 3, 2, 2, 2, 2, 3, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 2, 1, 0, 1, 0, 2, 0, 0, 0, 0 },

{ 0, 0, 3, 3, 2, 0, 1, 11, 1, 2, 3, 2, 0, 0 },

{ 0, 3, 3, 5, 2, 1, 0, 1, 0, 2, 6, 2, 3, 0 },

{ 0, 2, 1, 0, 1, 7, 1, 0, 1, 0, 1, 11, 2, 0 },

{ 0, 2, 11, 1, 0, 1, 11, 1, 0, 1, 0, 1, 2, 0 },

{ 2, 2, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 2, 2 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 11, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 11, 0 },

{ 0, 11, 1, 4, 1, 0, 0, 0, 0, 1, 8, 1, 11, 0 },

{ 0, 11, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 11, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 } };

int level\_20[14][14] = {

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 2, 2, 2, 0, 0, 0, 0, 0, 0, 2, 2, 2, 0 },

{ 0, 2, 1, 1, 7, 0, 0, 0, 0, 9, 1, 1, 2, 0 },

{ 0, 11, 1, 9, 2, 11, 0, 0, 11, 2, 10, 1, 11, 0 },

{ 0, 11, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 11, 0 },

{ 0, 11, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 11, 0 },

{ 0, 0, 1, 0, 11, 0, 0, 0, 0, 11, 0, 1, 0, 0 },

{ 0, 0, 2, 2, 2, 2, 0, 0, 2, 2, 2, 2, 0, 0 },

{ 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0 },

{ 0, 3, 8, 0, 0, 0, 0, 0, 0, 0, 0, 10, 3, 0 },

{ 0, 0, 3, 1, 3, 2, 1, 1, 2, 3, 1, 3, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 2, 1, 1, 1, 0, 0, 1, 1, 1, 2, 0, 0 },

{ 0, 0, 0, 0, 0, 3, 0, 0, 3, 0, 0, 0, 0, 0 } };

if (levelAssign == false)

{

levels.push\_back(level(level\_1, gameType, ballSpeed));

levels.push\_back(level(level\_2, gameType, ballSpeed));

levels.push\_back(level(level\_3, gameType, ballSpeed));

levels.push\_back(level(level\_4, gameType, ballSpeed));

levels.push\_back(level(level\_5, gameType, ballSpeed));

levels.push\_back(level(level\_6, gameType, ballSpeed));

levels.push\_back(level(level\_7, gameType, ballSpeed));

levels.push\_back(level(level\_8, gameType, ballSpeed));

levels.push\_back(level(level\_9, gameType, ballSpeed));

levels.push\_back(level(level\_10, gameType, ballSpeed));

levels.push\_back(level(level\_11, gameType, ballSpeed));

levels.push\_back(level(level\_12, gameType, ballSpeed));

levels.push\_back(level(level\_13, gameType, ballSpeed));

levels.push\_back(level(level\_14, gameType, ballSpeed));

levels.push\_back(level(level\_15, gameType, ballSpeed));

levels.push\_back(level(level\_16, gameType, ballSpeed));

levels.push\_back(level(level\_17, gameType, ballSpeed));

levels.push\_back(level(level\_18, gameType, ballSpeed));

levels.push\_back(level(level\_19, gameType, ballSpeed));

levels.push\_back(level(level\_20, gameType, ballSpeed));

levelAssign = true;

}

//Clip for PaddleSpeed Icon

if (paddleSpeed == 1)

{

paddleSIcon.x = 74;

paddleSIcon.y = 21;

paddleSIcon.w = 24;

paddleSIcon.h = 25;

}

else if (paddleSpeed == 2)

{

paddleSIcon.x = 98;

paddleSIcon.y = 21;

paddleSIcon.w = 24;

paddleSIcon.h = 25;

}

else

{

paddleSIcon.x = 122;

paddleSIcon.y = 21;

paddleSIcon.w = 24;

paddleSIcon.h = 25;

}

//Clip for BallSpeed Icon

if (ballSpeed == 1)

{

ballSIcon.x = 74;

ballSIcon.y = 49;

ballSIcon.w = 24;

ballSIcon.h = 25;

}

else if (ballSpeed == 2)

{

ballSIcon.x = 98;

ballSIcon.y = 49;

ballSIcon.w = 24;

ballSIcon.h = 25;

}

else

{

ballSIcon.x = 122;

ballSIcon.y = 49;

ballSIcon.w = 24;

ballSIcon.h = 25;

}

//Clip for game type icon

if (gameType == 1)

{

gameTIcon.x = 74;

gameTIcon.y = 77;

gameTIcon.w = 24;

gameTIcon.h = 24;

}

else if (gameType == 2)

{

gameTIcon.x = 98;

gameTIcon.y = 77;

gameTIcon.w = 24;

gameTIcon.h = 24;

}

else if (gameType == 3)

{

gameTIcon.x = 122;

gameTIcon.y = 77;

gameTIcon.w = 24;

gameTIcon.h = 24;

}

while(SDL\_PollEvent(&event))

{

if (event.type == SDL\_QUIT)

{

return 0; //return STATE\_EXIT

}

else if (event.type == SDL\_KEYDOWN)

{

if (event.key.keysym.sym == SDLK\_ESCAPE)

{

beginLevel = 0;

return 0; //return STATE\_MENU

}

}

levels[beginLevel].update(event);

}

levels[beginLevel].draw(paddleSpeed, ballSpeed, gameType);

for (int x = 0; x < lives; x++)

{

if (x <= 3)

{

blit(392 + (x \* 43), 2, blockSprites, screen, &livesClip);

}

}

SDL\_BlitSurface(blockSprites, &gameTIcon, screen, &rect[0]);

SDL\_BlitSurface(blockSprites, &ballSIcon, screen, &rect[1]);

SDL\_BlitSurface(blockSprites, &paddleSIcon, screen, &rect[2]);

//convert score to a string and output it

font = TTF\_OpenFont("data/forte.ttf", 20);

std::stringstream out;

std::string SScore;

out << score;

SScore = out.str();

printText(320, 5, SScore, 255, 255, 0);

if (levels[beginLevel].levelComplete(gameType) && storymode) //if you complete a level in story mode

{

if (lives == 3) { medal3L = true; }

if (score >= 10) { medal20 = true; }

setHighScore(gameType, beginLevel, bestScore, medalHS);

gameOver(gameType, beginLevel, bestScore, medalHS, medal3L, medal20);

lives = 3;

if (gameType == 1) { score = 10; }

else if (gameType == 3) { score = 0; }

levels[beginLevel].reset(ballSpeed);

if (beginLevel < levels.size() - 1)

{ beginLevel++; }

else

{

if (gameType == 2)

{

score = 0;

}

return 0;

}

}

else if (levels[beginLevel].levelComplete(gameType) && !storymode) // when you complete a level that is not story mode

{

if (lives == 3) { medal3L = true; }

if (score >= 10){ medal20 = true; }

setHighScore(gameType, beginLevel, bestScore, medalHS);

gameOver(gameType, beginLevel, bestScore, medalHS, medal3L, medal20);

levels[beginLevel].reset(ballSpeed);

return 2;

}

else if (levels[beginLevel].dead(ballSpeed)) // if the ball went below screen ( DIED )

{

lives--;

if (lives <= 0)

{

if (gameType == 2)

{

setHighScore(gameType, beginLevel, bestScore, medalHS);

score = 0;

}

gameOver(gameType, beginLevel, bestScore, medalHS, medal3L, medal20);

//reset variables

lives = 3;

levels[beginLevel].reset(ballSpeed);

return 0;

}

}

SDL\_Flip(screen);

return 1;

}

void game::gameOver(int gameType, int beginLevel, int (&bestScore)[41], bool &medalHS, bool &medal3L, bool &medal20)

{

//Make background

SDL\_Surface \*back = loadImage("data/GOBack.png");

Mix\_PlayChannel(-1, virtuo, 0);

//Display everything

blit(0, 0, back, screen);

if (gameType == 1)

{

medalClip[1].x = 0;

if(medalHS) { SDL\_BlitSurface(medalSprites, &medalClip[1], screen, &medalRect[1]); }

if(medal3L) { SDL\_BlitSurface(medalSprites, &medalClip[2], screen, &medalRect[2]); }

if(medal20) { SDL\_BlitSurface(medalSprites, &medalClip[0], screen, &medalRect[0]); }

}

else if(gameType == 2)

{

medalClip[1].x = 158;

if (medalHS) { SDL\_BlitSurface(medalSprites, &medalClip[1], screen, &medalRect[1]); }

if (medal3L) { SDL\_BlitSurface(medalSprites, &medalClip[2], screen, &medalRect[2]); }

if (medal20) { SDL\_BlitSurface(medalSprites, &medalClip[0], screen, &medalRect[0]); }

}

else

{

medalClip[1].x = 316;

if (medalHS) { SDL\_BlitSurface(medalSprites, &medalClip[1], screen, &medalRect[1]); }

if (medal3L) { SDL\_BlitSurface(medalSprites, &medalClip[2], screen, &medalRect[2]); }

if (medal20) { SDL\_BlitSurface(medalSprites, &medalClip[0], screen, &medalRect[0]); }

}

levels[beginLevel].reset(ballSpeed);

medalHS, medal3L, medal20 = false;

SDL\_Flip(screen);

SDL\_Delay(6000);

SDL\_FreeSurface(back);

}

void game::setHighScore(int gameType, int beginLevel, int (&bestScore)[41], bool &medalHS)

{

medalHS = false;

if (gameType == 1)//Virtuo

{

if (score > bestScore[beginLevel])

{

bestScore[beginLevel] = score;

medalHS = true;

score = 10;

}

}

else if (gameType == 2)//Breakout

{

if (score > bestScore[40])

{

bestScore[40] = score;

medalHS = true;

}

}

else if (gameType == 3)//VirtuoX

{

if (score > bestScore[beginLevel+20])

{

bestScore[beginLevel+20] = score;

medalHS = true;

score = 0;

}

}

std::ofstream file("data/bestscores.txt");

if (file.is\_open())

{

for (int count = 0; count < 41; count++)

{

file << bestScore[count] << " ";

}

file.close();

}

}

**Game.h**

#ifndef GAME\_H

#define GAME\_H

#include <vector>

#include <string>

#include "SDL.h"

#include "level.h"

class game

{

public:

game(int(&bestScore)[41]);

~game();

int gameLoop(int &beginLevel, bool story, int paddleSpeed, int ballSpeed, int gameType, int (&bestScore)[41], bool &medalHS, bool &medal3L, bool &medal20, bool &levelAssign);

void setHighScore(int gameType, int beginLevel, int(&bestScore)[41], bool &medalHS);

private:

void gameOver(int gameType, int beginLevel, int(&bestScore)[41], bool &medalHS, bool &medal3L, bool &medal20);

SDL\_Event event;

std::vector<level> levels;

SDL\_Rect livesClip;

SDL\_Rect ballSIcon;

SDL\_Rect paddleSIcon;

SDL\_Rect gameTIcon;

SDL\_Rect rect[3];

SDL\_Rect medalClip[3];

SDL\_Rect medalRect[3];

int lives;

int paddleSpeed;

int ballSpeed;

};

#endif