Ročníková práce

Hra Snake

Ročníková práce

Vedoucí práce: Vypracoval: Marian Dolinský

Mgr. František Skalka Studijní obor: IT

Číslo oboru: 18-20-M/01

Třída: IT2

Prohlašuji, že jsem ročníkovou práci vypracoval samostatně s přispěním vedoucího práce a použil jsem jen literaturu a informační zdroje uvedené v kapitole literatura.

Děkuji Mgr. Františku Skalkovi za odborné vedení a cenné rady, které mi poskytl při zpracování této ročníkové práce.

Souhlasím s půjčováním a zpřístupněním ročníkové práce.

V Brně 30. května 2016

Obsah

[Obsah 7](#_Toc452364208)

[1 Úvod 9](#_Toc452364209)

[2 O programu 11](#_Toc452364210)

[3 Zdrojový kód 13](#_Toc452364211)

[3.1 Snake.cpp 13](#_Toc452364212)

[3.2 unigfcs.h 39](#_Toc452364213)

[3.3 superior.hpp 68](#_Toc452364214)

[3.4 advio.hpp 73](#_Toc452364215)

[3.5 cnslcz.hpp 77](#_Toc452364216)

[3.6 cursor.hpp 77](#_Toc452364217)

[3.7 operators.hpp 78](#_Toc452364218)

[3.8 strext.hpp 79](#_Toc452364219)

[3.9 uibscs.hpp 80](#_Toc452364220)

[4 Vývojový diagram 85](#_Toc452364221)

[5 Závěr 87](#_Toc452364223)

# Úvod

Pro letošní ročníkovou práci jsem si vybral téma „Hra“. Původně však na mne toto téma nezbylo, a mým tématem tak byly „Tabulky“. Tehdy jsem měl již rozpracovanou hru Snake, ale i přesto jsem začal pracovat na hře Hledání min, která téma „Tabulky“ – 2d pole – splňovala. Danou hru jsem však nikdy úplně nedokončil, protože jsme si se spolužákem svá témata vyměnili, ale právě vývoj hry Hledání min mě donutil k vytvoření hlavičkového souboru „unigfcs.h“, který se dá jednoduše použít pro vytváření her a dodnes tvoří základ hry Snake. Až ke konci mě napadlo vytvořit Project Superior, ze kterého jsem však stihl udělat jen malou část, ale i tak tato hra využívá některé jeho funkce.

# O programu

Mojí ročníkovou prací je jednoduchá hra Snake ve Windows konzoli. Hra obsahuje jednoduché a přehledné menu, kde si může uživatel vybrat typ hry – hra pro jednoho hráče, nebo hru dvou hráčů, nabídku About, která zobrazí informace o programu a stránku zobrazující ovládání hry. Při začínání hry je možno nastavit obtížnost, která mění rychlost hada/ů, lze vybírat ze tří předdefinovaných rychlostí, nebo si vybrat vlastní. Hra obsahuje dva módy – Normal a Borderless. Borderless umožňuje projíždět skrz okraje okna. Dále může uživatel nastavit vzhled svého hada, jak barvu, tak znak reprezentující tělo hada. Program je plně responzivní, pokud však neprobíhá hra, kdy je změna velikosti nemožná. Ve hře jsou dva cheaty – fisa a hamster, které je možno aktivovat všude mimo probíhající hru.

# Zdrojový kód

## Snake.cpp

/\*

Name: Snake v1.7 'A better way'

Copyright: (c) 2016 Marian Dolinský

Author: Marian Dolinský

Date: 30/05/16 05:33

Description: Simple Snake game for console in Windows.

TODO:

hotseat arrows bug - sometimes rewrite

obstacles

responsive gameboard

slowdown, poison

increase speed with time in snake

settings - controls etc

save high scores

save stats

- berries eated

- win/loose rate

- specials eated

- time played

- bend counts :D

- specials eated

- games played

- difficulties played

.

.

.

!!! The game is fully responsive but when playing, board size cannot be changed and is resetted to size on game start original.

!!! Responsivity works better on Windows 10 than on older Windowses because console in W10 is resizable and can be maximized as another window apps.

!!! Don't use Dev-C++ project, it will not recompile outside-project files (unigfcs.h) each project compilation (some files must be manually deleted to recompile them).

\*/

//#define DEBUG

#define DISABLE\_BUFFERASWINDOW

const char CHEATS[][10] =

{

{ "fisa" },

{ "hamster" }

};

#define CHANGELOG\_COLUMNS 45

const char CHANGELOG\_GAME[][CHANGELOG\_COLUMNS] =

{

{ "1.7 'A better way': (05/29/2016)" },

{ "- Rewrited game engine" },

{ "- Showing score in title" },

{ "- Added new cheats" },

{ "- Added specials" },

{ "- Fixed bug with right bottom corner" },

{ "- Fully removed screen flashing" },

{ "- Some bugs were fixed" },

{ "1.6 'FISA is your hero': (12/18/2015)" },

{ "- Added a little secret" },

{ "- Changed controls" },

{ "- Fixed screen flashing" },

{ "- Improved game performance" },

{ "- Some bugs were fixed" },

{ "1.5 'Make a way!': (11/29/2015)" },

{ "- Added borderless mode" },

{ "- Better game performance" },

{ "1.4 'It grows': (11/24/2015)" },

{ "- Game is now responsive designed" },

{ "- Code were optimized" },

{ "1.3 'Let me continue': (11/22/2015)" },

{ "- Hotseat continues if one player dies" },

{ "- Added changelog" },

{ "- Some bugs were fixed" },

{ "1.2 'Do it twice': (11/17/2015)" },

{ "- Added multiplayer" },

{ "- Code is more effective" },

{ "- Some bugs were fixed" },

{ "1.1 'When meowside flies': (11/13/2015)" },

{ "- Added menus" },

{ "- Some bugs were fixed" },

{ "1.0 'First release': (11/12/2015)" },

{ "- Only singleplayer" },

{ "- Without menus" }

};

#include "unigfcs.h"

const char BERRY\_CHAR = 249;

const char NEUTRAL = -5;

const int CHEATS\_HAMSTER = 3;

//const int NAME\_LENGTH\_MAX = 16;

const int SLEEP\_EASY = 200;

const int SLEEP\_MEDIUM = 50;

const int SLEEP\_HARD = 30;

const int SNAKE\_LENGTH\_START = 3;

const int SNAKE\_LENGTH\_MAX = SNAKE\_LENGTH\_START + 100;

#define SPECIAL\_SPAWN rand() % 1000000 == 53

const int SPECIAL\_BONUS = 3;

const int LENGTH\_PRECISION = (int)floor(log10(SNAKE\_LENGTH\_MAX)) + 1;

const char CONTROLS[2][4] =

{

{ 'W', 'A', 'S', 'D'},

{ Up, Left, Down, Right }

};

typedef enum

{

Fisa,

Hamster

} CHEAT;

typedef enum

{

Bonus,

Mine/\*,

Slowdown,

Poison\*/

} SPECIAL;

typedef struct

{

char Direction;

COORD Position;

} VECTOR;

typedef struct

{

COORD Position;

VISUAL Visual;

} BERRY;

typedef struct

{

bool Collision;

int Active;

int Shown;

VECTOR Head;

VECTOR Tail;

struct BEND

{

int Active;

int ToAssign;

VECTOR Vector[SNAKE\_LENGTH\_MAX];

} Bend;

VISUAL Visual;/\*

char PlayerName[NAME\_LENGTH\_MAX];

bool Poison;

int PoisonTimer;\*/

} SNAKE;

typedef enum

{

WASD,

Arrows

} CONTROLTYPE;

typedef enum

{

KUp,

KLeft,

KDown,

KRight

} CONTROL;

char liveTitle[35] = {0};

BERRY specials[] =

{

// Bonus

{

{ }, { '+', Green }

},

// Mine

{

{ }, { 148, Red }

}/\*,

// Slowdown

{

{ }, { 233, Yellow }

},

// Poison

{

{ }, { 233, DarkYellow }

}\*/

};

int main();

//void controls(); - already declared in unigfcs.h

//bool game(DIFFICULTY difficulty, bool hotseat); - already declared in unigfcs.h

int customdifficultyselection();

//bool strrem(char \*str, int index);

bool playername(int sleep, bool hotseat, bool borders);

bool snakecreator(int sleep, bool hotseat, bool borders, const char \*playersNames[2]);

// In-game functions

void livetitle(int score);

void livetitle(const char \*playerName1, int score1, const char \*playerName2, int score2);

void setrandomcolors();

void recreateboard(bool \*gameBoard, VISUAL snake, BERRY berry, bool isSpecialShown, SPECIAL shownSpecial);

void recreateboard(bool \*gameBoard, SNAKE snakes[2], BERRY berries[2], bool isSpecialShown, SPECIAL shownSpecial);

void newberry(BERRY \*berry, bool \*gameBoard);

void playsnake(int sleep, bool borders, VISUAL visual, const char \*playerName);

void playhotseat(int sleep, bool borders, VISUAL visuals[2], const char \*playersNames[2]);

int main()

{

initialize("Snake", "v1.7 'A better way'", "beta 4", 79, 24);

mainmenu(Green);

// Program should not get there

cls();

vcenter(3);

hcenter("SOMETHING WENT WRONG :-(\n");

hcenter("Press any key to close the game . . .");

getch();

exit(EXIT\_FAILURE);

}

void controls()

{

const int CENTERING = 35;

bool render = true;

char key;

while (true)

{

if (consoleSizeChanged || render)

{

cls();

#ifdef DEBUG

vcenter(9);

#else

vcenter(8);

#endif

hcenter("CONTROLS");

fputs("\n\n", stdout);

hcenter(CENTERING, "Player 1: Arrow keys\n");

hcenter(CENTERING, "Player 2: WASD\n");

#ifdef DEBUG

hcenter(CENTERING, "Freeze game: SPACE\n");

#endif

hcenter(CENTERING, "Pause game: ESC\n");

hcenter(CENTERING, "Menus navigation: Arrow keys, ENTER, ESC");

fputs("\n\n", stdout);

hcenter("> Back ");

consoleSizeChanged = false;

render = false;

}

#ifdef DEBUG

movecursor(COORD\_ORIGIN);

DEBUG\_CHEATS;

#endif

if (kbhit())

{

key = getch();

if (key == Enter || key == Esc)

{

return;

}

}

Sleep(MENU\_SLEEP);

}

}

bool game(DIFFICULTY difficulty, bool hotseat)

{

const int CENTERING = 6;

bool renderMain = true, renderSelection = true;

char key;

int outDifficulty, selection = 0;

COORD cursorPosition;

if (difficulty == Custom)

{

outDifficulty = customdifficultyselection();

if (outDifficulty == 0)

{

return false;

}

}

else

{

outDifficulty = (difficulty == Easy ? SLEEP\_EASY : (difficulty == Medium ? SLEEP\_MEDIUM : SLEEP\_HARD));

}

while (true)

{

if (consoleSizeChanged || renderMain)

{

cls();

vcenter(5);

hcenter("CHOOSE GAME MODE");

fputs("\n\n", stdout);

cursorPosition = hcenter(CENTERING, "Normal\n");

cursorPosition.X -= 2;

hcenter(CENTERING, "Borderless\n");

putchar('\n');

hcenter(CENTERING, "Back");

consoleSizeChanged = false;

renderMain = false;

renderSelection = true;

}

#ifdef DEBUG

DEBUG\_MAIN;

#endif

if (renderSelection)

{

for (int i = (selection == 2 ? -2 : -1); i <= (selection == 1 ? 2 : 1); i++)

{

movecursor(cursorPosition.X, cursorPosition.Y + selection + (selection == 2 ? 1 : 0) + i);

putchar(' ');

}

movecursor(cursorPosition.X, cursorPosition.Y + selection + (selection == 2 ? 1 : 0));

putchar('>');

renderSelection = false;

}

if (kbhit())

{

key = getkey(getcheat(getch()));

if ((key == Enter && selection == 2) || key == Esc)

{

return false;

}

else if (key == Enter)

{

if (playername(outDifficulty, hotseat, (selection == 0 ? true : false)))

{

return true;

}

renderMain = true;

}

else if (key == Up && selection != 0)

{

selection--;

renderSelection = true;

}

else if (key == Down && selection != 2)

{

selection++;

renderSelection = true;

}

}

Sleep(MENU\_SLEEP);

}

}

int customdifficultyselection()

{

const int CENTERING = 7;

const int MIN = 10;

const int MAX = 300;

const int STEP = 5;

bool renderMain = true, renderSelection = true;

char key;

int selection = 0, wheelSelection = MIN;

COORD cursorPosition, wheelCursorPosition;

while (true)

{

if (consoleSizeChanged || renderMain)

{

cls();

vcenter(7);

hcenter("CHOOSE CUSTOM DELAY");

fputs("\n\n", stdout);

wheelCursorPosition = getcursorposition();

printf("\n\n\n");

cursorPosition = hcenter(CENTERING, "Confirm\n");

cursorPosition.X -= 2;

hcenter(CENTERING, "Back");

consoleSizeChanged = false;

renderMain = false;

renderSelection = true;

}

#ifdef DEBUG

DEBUG\_MAIN;

printf("\nwheelSelection: %3d | ", wheelSelection);

printf("wheelCursorPosition: %\*d %\*d", consoleSizeXPrecision, wheelCursorPosition.X, consoleSizeXPrecision, wheelCursorPosition.Y);

#endif

if (renderSelection)

{

wheelselection(wheelCursorPosition, wheelSelection, MIN, MAX, STEP);

movecursor(cursorPosition.X, cursorPosition.Y + (selection == 0 ? 1 : 0));

putchar(' ');

movecursor(cursorPosition.X, cursorPosition.Y + selection);

putchar('>');

renderSelection = false;

}

if (kbhit())

{

key = getkey(getcheat(getch()));

if ((key == Enter && selection == 1) || key == Esc)

{

return 0;

}

else if (key == Enter)

{

return wheelSelection;

}

else if (key == Up && selection != 0)

{

selection--;

renderSelection = true;

}

else if (key == Down && selection != 1)

{

selection++;

renderSelection = true;

}

else if (key == Left)

{

wheelSelection -= STEP;

if (wheelSelection == MIN - STEP)

{

wheelSelection = MAX;

}

renderSelection = true;

}

else if (key == Right)

{

wheelSelection += STEP;

if (wheelSelection == MAX + STEP)

{

wheelSelection = MIN;

}

renderSelection = true;

}

}

Sleep(MENU\_SLEEP);

}

}

bool playername(int sleep, bool hotseat, bool borders)

{

char playersNames[2][1] = {0};

if (snakecreator(sleep, hotseat, borders, (const char \*\*)playersNames))

{

return true;

}

return false;

// Here should be TextBox for selecting names

}

bool snakecreator(int sleep, bool hotseat, bool borders, const char \*playersNames[2])

{

const char BODIES[] = {'O', 'X', 176, 177, 219, '\0'};

const int CENTERING = 7;

const int COLOR\_CHAR = 254;

const int COLOR\_MIN = Blue;

const int COLOR\_MAX = White;

bool changing, resetVisual = true, renderMain = true, renderMainSelection = true, renderSnakeSelection = true;

char key;

int actualSnake, bodySelection, colorSelection, selection;

COORD bodyCursorPosition, colorCursorPosition, snakeCursorPosition, cursorPosition;

VISUAL snakeVisual[2];

changing = false;

actualSnake = 0;

while (true)

{

if (consoleSizeChanged || renderMain)

{

if (resetVisual)

{

bodySelection = 0;

colorSelection = White;

selection = 0;

snakeVisual[actualSnake].Char = BODIES[bodySelection];

snakeVisual[actualSnake].Color = (COLOR)colorSelection;

resetVisual = false;

}

cls();

vcenter(15);

if (!hotseat)

{

hcenter("CREATE YOUR OWN SNAKE");

}

else

{

hcenter(strlen("PLAYER X - CREATE YOUR OWN SNAKE"));

printf("PLAYER %d - CREATE YOUR OWN SNAKE", actualSnake + 1);

}

fputs("\n\n", stdout);

// Go to absolute center on X axis

colorCursorPosition = snakeCursorPosition = hcenter(0);

// Set coordinates for snake preview

snakeCursorPosition.X--;

snakeCursorPosition.Y += 2;

// Get coordinates for body selection

movecursor(0, colorCursorPosition.Y + 8);

bodyCursorPosition = hcenter(9);

// Print color selection

colorCursorPosition.X += 10;

for (int i = COLOR\_MAX; i >= COLOR\_MIN; i--)

{

setforeground((COLOR)i);

movecursor(colorCursorPosition.X, colorCursorPosition.Y + (COLOR\_MAX - i));

putchar(COLOR\_CHAR);

}

colorCursorPosition.X += 2;

putchar('\n');

// Get coordinates for main selection

cursorPosition = hcenter(CENTERING);

cursorPosition.X -= 2;

cursorPosition.Y += 3;

consoleSizeChanged = false;

renderMain = false;

renderMainSelection = true;

renderSnakeSelection = true;

}

#ifdef DEBUG

setforeground(White);

DEBUG\_MAIN;

printf("\nbodyCursor: %\*d %\*d | ", consoleSizeXPrecision, bodyCursorPosition.X, consoleSizeXPrecision, bodyCursorPosition.Y);

printf("colorCursor: %\*d %\*d | ", consoleSizeXPrecision, colorCursorPosition.X, consoleSizeXPrecision, colorCursorPosition.Y);

printf("snakeCursor: %\*d %\*d", consoleSizeXPrecision, snakeCursorPosition.X, consoleSizeXPrecision, snakeCursorPosition.Y);

printf("\nbodySelection: %2d | ", bodySelection);

printf("colorSelection: %2d | ", colorSelection);

printf("changing: %d | ", changing);

printf("snakeVisual[%d].Char: %c | ", actualSnake, snakeVisual[actualSnake].Char);

printf("snakeVisual[%d].Color: %2d", actualSnake, snakeVisual[actualSnake].Color);

#endif

if (renderSnakeSelection)

{

// Print snake preview

setforeground((COLOR)colorSelection);

for (int i = 0; i < 3; i++)

{

movecursor(snakeCursorPosition.X, snakeCursorPosition.Y + i);

putchar(BODIES[bodySelection]);

}

// Print body selection

movecursor(bodyCursorPosition);

setforeground(DarkGray);

printf("%c ", (bodySelection == 1 ? BODIES[strlen(BODIES) - 1] : (bodySelection == 0 ? BODIES[strlen(BODIES) - 2] : BODIES[bodySelection - 2])));

setforeground(Gray);

printf("%c ", (bodySelection == 0 ? BODIES[strlen(BODIES) - 1] : BODIES[bodySelection - 1]));

setforeground(White);

printf("%c ", BODIES[bodySelection]);

setforeground(Gray);

printf("%c ", (bodySelection == strlen(BODIES) - 1 ? BODIES[0] : BODIES[bodySelection + 1]));

setforeground(DarkGray);

printf("%c", (bodySelection == strlen(BODIES) - 2 ? BODIES[0] : (bodySelection == strlen(BODIES) - 1 ? BODIES[1] : BODIES[bodySelection + 2])));

setforeground(White);

}

if (renderSnakeSelection)

{

for (int i = 0; i < COLOR\_MAX; i++)

{

movecursor(colorCursorPosition.X, colorCursorPosition.Y + i);

putchar(' ');

}

if (changing)

{

movecursor(colorCursorPosition.X, colorCursorPosition.Y + (COLOR\_MAX - colorSelection));

putchar('<');

setforeground(DarkGray);

}

renderSnakeSelection = false;

}

if (renderMainSelection)

{

movecursor(0, cursorPosition.Y);

hcenter(CENTERING, "Confirm\n");

hcenter(CENTERING, "Change\n");

hcenter(CENTERING, "Back");

for (int i = -1; i <= 1; i += 2)

{

movecursor(cursorPosition.X, cursorPosition.Y + selection + i);

putchar(' ');

}

movecursor(cursorPosition.X, cursorPosition.Y + selection);

putchar('>');

renderMainSelection = false;

}

if (kbhit())

{

key = getkey(getcheat(getch()));

if (changing)

{

if (key == Enter)

{

changing = false;

renderMainSelection = true;

renderSnakeSelection = true;

// Assign snakeVisual from selection

snakeVisual[actualSnake].Char = BODIES[bodySelection];

snakeVisual[actualSnake].Color = (COLOR)colorSelection;

}

else if (key == Esc)

{

changing = false;

resetVisual = true;

renderMainSelection = true;

renderSnakeSelection = true;

// Assign selection from snakeVisual

colorSelection = (int)snakeVisual[actualSnake].Color;

for (int i = 0; i < strlen(BODIES); i++)

{

if (BODIES[i] == snakeVisual[actualSnake].Char)

{

bodySelection = i;

break;

}

}

}

else if (key == Up && colorSelection != COLOR\_MAX)

{

colorSelection++;

renderSnakeSelection = true;

}

else if (key == Down && colorSelection != COLOR\_MIN)

{

colorSelection--;

renderSnakeSelection = true;

}

else if (key == Left)

{

if (bodySelection == 0)

{

bodySelection = strlen(BODIES) - 1;

}

else

{

bodySelection--;

}

renderSnakeSelection = true;

}

else if (key == Right)

{

if (bodySelection == strlen(BODIES) - 1)

{

bodySelection = 0;

}

else

{

bodySelection++;

}

renderSnakeSelection = true;

}

}

else //if (!changing)

{

if ((key == Enter && selection == 2) || key == Esc)

{

if (actualSnake == 0)

{

return false;

}

else

{

actualSnake = 0;

renderMain = true;

// Assign selection from snakeVisual

colorSelection = (int)snakeVisual[actualSnake].Color;

for (int i = 0; i < strlen(BODIES); i++)

{

if (BODIES[i] == snakeVisual[actualSnake].Char)

{

bodySelection = i;

break;

}

}

}

}

else if (key == Enter)

{

if (selection == 1)

{

changing = true;

renderMainSelection = true;

renderSnakeSelection = true;

}

else //if (selection == 0)

{

if (!hotseat)

{

playsnake(sleep, borders, snakeVisual[actualSnake], playersNames[0]);

return true;

}

else

{

if (actualSnake == 0)

{

actualSnake = 1;

resetVisual = true;

renderMain = true;

}

else

{

playhotseat(sleep, borders, snakeVisual, playersNames);

return true;

}

}

}

}

else if (key == Up && selection != 0)

{

selection--;

renderMainSelection = true;

}

else if (key == Down && selection != 2)

{

selection++;

renderMainSelection = true;

}

}

}

Sleep(MENU\_SLEEP);

}

}

void livetitle(int score)

{

sprintf(liveTitle, "%s - score: %d", gameName, score - SNAKE\_LENGTH\_START);

SetConsoleTitle(liveTitle);

}

void livetitle(const char \*playerName1, int score1, const char \*playerName2, int score2)

{

sprintf(liveTitle, "%s - %s: %d, %s: %d", gameName, playerName1, score1 - SNAKE\_LENGTH\_START, playerName2, score2 - SNAKE\_LENGTH\_START);

SetConsoleTitle(liveTitle);

}

void setrandomcolors()

{

DWORD written;

do

{

setforeground((COLOR)(rand() % 16));

setbackground((COLOR)(rand() % 16));

} while (foregroundColor == backgroundColor);

FillConsoleOutputAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), foregroundColor + (backgroundColor << 4), (forcedConsoleSize.X + 1) \* (forcedConsoleSize.Y + 1), COORD\_ORIGIN, &written);

}

void recreateboard(bool \*gameBoard, VISUAL snake, BERRY berry, bool isSpecialShown, SPECIAL shownSpecial)

{

cls();

if (cheats[Fisa])

{

setrandomcolors();

}

// Print snake

for (int y = 0; y < forcedConsoleSize.Y + 1; y++)

{

for (int x = 0; x < forcedConsoleSize.X + 1; x++)

{

if (\*(gameBoard + (y \* (forcedConsoleSize.X + 1)) + x))

{

movecursor(x, y);

if (!cheats[Fisa])

{

draw(snake);

}

else

{

putchar(BLOCK\_NORMAL);

}

}

}

}

// Print berry

movecursor(berry.Position);

if (!cheats[Fisa])

{

draw(berry.Visual);

}

else

{

putchar(berry.Visual.Char);

}

// Print special

if (isSpecialShown)

{

movecursor(specials[shownSpecial].Position);

if (!cheats[Fisa])

{

draw(specials[shownSpecial].Visual);

}

else

{

putchar(specials[shownSpecial].Visual.Char);

}

}

}

void recreateboard(bool \*gameBoard, SNAKE snakes[2], BERRY berries[2], bool isSpecialShown, SPECIAL shownSpecial)

{

// TODO: everything here

}

void newberry(BERRY \*berry, bool \*gameBoard)

{

bool collision;

do

{

collision = false;

berry->Position.X = rand() % (forcedConsoleSize.X + 1);

berry->Position.Y = rand() % (forcedConsoleSize.Y + 1);

// Collision - new berry with snake

for (int y = -1; y <= 1; y++)

{

// Skip this cycle step when variables points out of GAMEBOARD range

if ((berry->Position.Y == 0 && y == -1) || (berry->Position.Y == forcedConsoleSize.Y && y == 1))

{

continue;

}

for (int x = -1; x <= 1; x++)

{

// Skip this cycle step when variables points out of GAMEBOARD range

if ((berry->Position.X == 0 && x == -1) || (berry->Position.X == forcedConsoleSize.X && x == 1))

{

continue;

}

if (\*(gameBoard + ((berry->Position.Y + y) \* (forcedConsoleSize.X + 1)) + (berry->Position.X + x)))

{

collision = true;

break;

}

}

if (collision)

{

break;

}

}

} while (collision);

berry->Visual.Color = (COLOR)(rand() % 8 + 8);

// Print berry

movecursor(berry->Position);

draw(berry->Visual);

}

void playsnake(int sleep, bool borders, VISUAL visual, const char \*playerName)

{

#ifndef DEBUG

timer(true);

#endif

forcedConsoleSize = consoleSize;

SuspendThread(sizecheckthreadHandle);

SuspendThread(cheatpromptthreadHandle);

cls();

bool isSpecialShown, lastFisaState;

bool gameBoard[forcedConsoleSize.Y + 1][forcedConsoleSize.X + 1];

char key;

SPECIAL shownSpecial;

BERRY berry;

SNAKE snake;

// Assign variables

isSpecialShown = false;

lastFisaState = cheats[Fisa];

memset(gameBoard, false, (forcedConsoleSize.X + 1) \* (forcedConsoleSize.Y + 1));

shownSpecial = (SPECIAL)0;

berry.Visual.Char = BERRY\_CHAR;

snake.Collision = false;

snake.Active = SNAKE\_LENGTH\_START;

snake.Shown = 0;

snake.Bend.Active = 0;

snake.Bend.ToAssign = 0;

snake.Head.Position.X = (forcedConsoleSize.X + 1) / 2;

snake.Head.Position.Y = (forcedConsoleSize.Y + 1) / 2;

snake.Tail.Position = snake.Head.Position;

snake.Head.Direction = CONTROLS[Arrows][KUp];

snake.Tail.Direction = snake.Head.Direction;

snake.Visual = visual;

//strcpy(snake.PlayerName, playerName);

gameBoard[snake.Head.Position.Y][snake.Head.Position.X] = true;

for (int i = 0; i < SNAKE\_LENGTH\_MAX; i++)

{

snake.Bend.Vector[i].Position.X = NEUTRAL;

snake.Bend.Vector[i].Position.Y = NEUTRAL;

snake.Bend.Vector[i].Direction = NEUTRAL;

}

// Create berry

newberry(&berry, gameBoard[0]);

livetitle(snake.Active);

// Main game cycle

while (true)

{

forceconsolesize();

if (consoleSizeChanged)

{

movecursor(berry.Position);

draw(berry.Visual);

consoleSizeChanged = false;

}

if (cheats[Fisa])

{

setrandomcolors();

}

// Print snake

gameBoard[snake.Head.Position.Y][snake.Head.Position.X] = true;

movecursor(snake.Head.Position);

if (!cheats[Fisa])

{

draw(snake.Visual);

}

else

{

putchar(BLOCK\_NORMAL);

}

if (snake.Shown < snake.Active)

{

snake.Shown++;

}

else

{

// Remove tail

gameBoard[snake.Tail.Position.Y][snake.Tail.Position.X] = false;

movecursor(snake.Tail.Position);

if (!cheats[Fisa])

{

setforeground(Black);

putchar(snake.Visual.Char);

setforeground(White);

}

else

{

putchar(' ');

}

// Change tail direction

if (snake.Tail.Position == snake.Bend.Vector[snake.Bend.Active].Position)

{

snake.Tail.Direction = snake.Bend.Vector[snake.Bend.Active++].Direction;

if (snake.Bend.Active == SNAKE\_LENGTH\_MAX)

{

snake.Bend.Active = 0;

}

}

// Change tail position

if (snake.Tail.Direction == CONTROLS[Arrows][KUp])

{

snake.Tail.Position.Y--;

}

else if (snake.Tail.Direction == CONTROLS[Arrows][KDown])

{

snake.Tail.Position.Y++;

}

else if (snake.Tail.Direction == CONTROLS[Arrows][KLeft])

{

snake.Tail.Position.X--;

}

else if (snake.Tail.Direction == CONTROLS[Arrows][KRight])

{

snake.Tail.Position.X++;

}

// Move tail to opposite console edges

if (!borders)

{

if (snake.Tail.Position.X < 0)

{

snake.Tail.Position.X = forcedConsoleSize.X;

}

else if (snake.Tail.Position.X > forcedConsoleSize.X)

{

snake.Tail.Position.X = 0;

}

else if (snake.Tail.Position.Y < 0)

{

snake.Tail.Position.Y = forcedConsoleSize.Y;

}

else if (snake.Tail.Position.Y > forcedConsoleSize.Y)

{

snake.Tail.Position.Y = 0;

}

}

}

movecursor(COORD\_ORIGIN);

#ifdef DEBUG

printf("head: %\*d %\*d %c | ", consoleSizeXPrecision, snake.Head.Position.X, consoleSizeXPrecision, snake.Head.Position.Y, printarrow(snake.Head.Direction));

printf("bend[%\*d]:% \*d% \*d %c | ", LENGTH\_PRECISION, snake.Bend.Active, consoleSizeXPrecision + 1, snake.Bend.Vector[snake.Bend.Active].Position.X, consoleSizeXPrecision + 1, snake.Bend.Vector[snake.Bend.Active].Position.Y,

printarrow(snake.Bend.Vector[snake.Bend.Active].Direction));

printf("active: %\*d | ", LENGTH\_PRECISION, snake.Active);

printf("berry: %\*d %\*d | ", consoleSizeXPrecision, berry.Position.X, consoleSizeXPrecision, berry.Position.Y);

printf("color: %c%c", (foregroundColor < 10 ? foregroundColor + '0' : foregroundColor - 10 + 'A'), (backgroundColor < 10 ? backgroundColor + '0' : backgroundColor - 10 + 'A'));

printf("\ntail: %\*d %\*d %c | ", consoleSizeXPrecision, snake.Tail.Position.X, consoleSizeXPrecision, snake.Tail.Position.Y, printarrow(snake.Tail.Direction));

if (snake.Bend.ToAssign > 0)

{

printf("bend[%\*d]: %\*d %\*d %c | ",

LENGTH\_PRECISION, snake.Bend.ToAssign,

consoleSizeXPrecision, snake.Bend.Vector[snake.Bend.ToAssign - 1].Position.X,

consoleSizeXPrecision, snake.Bend.Vector[snake.Bend.ToAssign - 1].Position.Y,

printarrow(snake.Bend.Vector[snake.Bend.ToAssign - 1].Direction)

);

}

else

{

printf("bend[%\*d]: %s %s - | ", LENGTH\_PRECISION, snake.Bend.ToAssign, createline('-', consoleSizeXPrecision), createline('-', consoleSizeXPrecision));

}

printf("shown: %\*d | ", LENGTH\_PRECISION, snake.Shown);

printf("sleep: %3d", sleep);

printf("\nisSpecialShown: %d | ", isSpecialShown);

printf("specials[%d]: %\*d %\*d", shownSpecial, consoleSizeXPrecision, specials[shownSpecial].Position.X, consoleSizeXPrecision, specials[shownSpecial].Position.Y);

DEBUG\_CHEATS;

#endif

Sleep(sleep);

if (kbhit())

{

key = getch();

if (key == -32 || key == 224)

{

// Get new direction for head

key = getch();

if (IF\_ARROW(key) && key != snake.Head.Direction && key != (snake.Head.Direction == Up ? Down : (snake.Head.Direction == Down ? Up : (snake.Head.Direction == Left ? Right : Left))))

{

// Assign new bend

snake.Head.Direction = snake.Bend.Vector[snake.Bend.ToAssign].Direction = key;

snake.Bend.Vector[snake.Bend.ToAssign++].Position = snake.Head.Position;

if (snake.Bend.ToAssign == SNAKE\_LENGTH\_MAX)

{

snake.Bend.ToAssign = 0;

}

}

}

else if (key == Esc)

{

ResumeThread(sizecheckthreadHandle);

ResumeThread(cheatpromptthreadHandle);

SetConsoleTitle(gameName);

setforeground(White);

setbackground(Black);

if (pausemenu(false))

{

return;

}

SuspendThread(sizecheckthreadHandle);

SuspendThread(cheatpromptthreadHandle);

livetitle(snake.Active);

if (cheats[Fisa] != lastFisaState)

{

recreateboard(gameBoard[0], snake.Visual, berry, isSpecialShown, shownSpecial);

lastFisaState = cheats[Fisa];

}

}

#ifdef DEBUG

else if (key == Space)

{

while (getch() != Space);

}

#endif

}

// Change snake position

if (snake.Head.Direction == CONTROLS[Arrows][KUp])

{

snake.Head.Position.Y--;

}

else if (snake.Head.Direction == CONTROLS[Arrows][KDown])

{

snake.Head.Position.Y++;

}

else if (snake.Head.Direction == CONTROLS[Arrows][KLeft])

{

snake.Head.Position.X--;

}

else if (snake.Head.Direction == CONTROLS[Arrows][KRight])

{

snake.Head.Position.X++;

}

if (!borders)

{

// Move head to opposite console edge

if (snake.Head.Position.X < 0)

{

snake.Head.Position.X = forcedConsoleSize.X;

}

else if (snake.Head.Position.X > forcedConsoleSize.X)

{

snake.Head.Position.X = 0;

}

else if (snake.Head.Position.Y < 0)

{

snake.Head.Position.Y = forcedConsoleSize.Y;

}

else if (snake.Head.Position.Y > forcedConsoleSize.Y)

{

snake.Head.Position.Y = 0;

}

}

// Eat berry

if (snake.Head.Position == berry.Position)

{

snake.Active += (cheats[Hamster] ? CHEATS\_HAMSTER : 1);

livetitle(snake.Active);

newberry(&berry, gameBoard[0]);

}

// Specials

if (!isSpecialShown && SPECIAL\_SPAWN)

{

// Create new special berry

shownSpecial = (SPECIAL)(rand() % (sizeof(specials) / sizeof(specials[0])));

newberry(&specials[shownSpecial], gameBoard[0]);

isSpecialShown = true;

}

else if (isSpecialShown)

{

if (snake.Head.Position == specials[shownSpecial].Position)

{

// Eat special berry

switch (shownSpecial)

{

case Bonus:

snake.Active += SPECIAL\_BONUS;

livetitle(snake.Active);

break;

case Mine:

snake.Collision = true;

break;

/\*

case Slowdown:

// TODO: slowdown

break;

case Poison:

// TODO: poison

break;\*/

}

isSpecialShown = false;

}

else if (SPECIAL\_SPAWN)

{

// Remove special berry

movecursor(specials[shownSpecial].Position);

putchar(' ');

isSpecialShown = false;

}

}

// Collision

if (!snake.Collision)

{

snake.Collision = borders && !(snake.Head.Position >= COORD\_ORIGIN && snake.Head.Position <= forcedConsoleSize) || gameBoard[snake.Head.Position.Y][snake.Head.Position.X];

}

// Game end

if (snake.Collision || snake.Active == SNAKE\_LENGTH\_MAX)

{

SetConsoleTitle(gameName);

ResumeThread(sizecheckthreadHandle);

ResumeThread(cheatpromptthreadHandle);

setforeground(White);

setbackground(Black);

showscore(snake.Active == SNAKE\_LENGTH\_MAX, snake.Active - SNAKE\_LENGTH\_START, sleep == SLEEP\_EASY ? Easy : (sleep == SLEEP\_MEDIUM ? Medium : (sleep == SLEEP\_HARD ? Hard : Custom)), borders ? "Normal" : "Borderless");

return;

}

}

}

void playhotseat(int sleep, bool borders, VISUAL visuals[2], const char \*playersNames[2])

{

#ifndef DEBUG

timer(true);

#endif

forcedConsoleSize = consoleSize;

SuspendThread(sizecheckthreadHandle);

SuspendThread(cheatpromptthreadHandle);

cls();

bool isSpecialShown, lastFisaState;

bool gotNewDirection[2];

bool gameBoard[forcedConsoleSize.Y + 1][forcedConsoleSize.X + 1];

char key;

// Represents actual snake index from snakes array

int s;

// Represents actual berry index from berries array

int b;

SPECIAL shownSpecial;

BERRY berries[2];

SNAKE snakes[2];

// Assign variables

isSpecialShown = false;

lastFisaState = cheats[Fisa];

memset(gameBoard, false, (forcedConsoleSize.X + 1) \* (forcedConsoleSize.Y + 1));

shownSpecial = (SPECIAL)0;

for (s = 0; s <= 1; s++)

{

berries[s].Visual.Char = BERRY\_CHAR;

snakes[s].Collision = false;

snakes[s].Active = SNAKE\_LENGTH\_START;

snakes[s].Shown = 0;

snakes[s].Bend.Active = 0;

snakes[s].Bend.ToAssign = 0;

snakes[s].Head.Position.X = (forcedConsoleSize.X + 1) \* (1 + (2 \* s)) / 4;

snakes[s].Head.Position.Y = (forcedConsoleSize.Y + 1) / 2;

snakes[s].Tail.Position = snakes[s].Head.Position;

snakes[s].Head.Direction = CONTROLS[s][KUp];

snakes[s].Tail.Direction = snakes[s].Head.Direction;

snakes[s].Visual = visuals[s];

//strcpy(snakes[s].PlayerName, playersNames[s]);

gameBoard[snakes[s].Head.Position.Y][snakes[s].Head.Position.X] = true;

for (int i = 0; i < SNAKE\_LENGTH\_MAX; i++)

{

snakes[s].Bend.Vector[i].Position.X = NEUTRAL;

snakes[s].Bend.Vector[i].Position.Y = NEUTRAL;

snakes[s].Bend.Vector[i].Direction = NEUTRAL;

}

}

// Create berries

newberry(&berries[0], gameBoard[0]);

do

{

newberry(&berries[1], gameBoard[0]);

} while (berries[0].Position == berries[1].Position);

livetitle("Player 1", snakes[0].Active, "Player 2", snakes[1].Active);

// Main game cycle

while (true)

{

forceconsolesize();

if (consoleSizeChanged)

{

movecursor(berries[0].Position);

draw(berries[0].Visual);

movecursor(berries[1].Position);

draw(berries[1].Visual);

consoleSizeChanged = false;

}

/\*

if (cheats[Fisa])

{

setrandomcolors();

}

\*/

for (s = 0; s <= 1; s++)

{

if (!snakes[s].Collision)

{

// Print snake

gameBoard[snakes[s].Head.Position.Y][snakes[s].Head.Position.X] = true;

movecursor(snakes[s].Head.Position);

/\*

if (!cheats[Fisa])

{\*/

draw(snakes[s].Visual);

/\*}

else

{

putchar(BLOCK\_NORMAL);

}\*/

if (snakes[s].Shown < snakes[s].Active)

{

snakes[s].Shown++;

}

else

{

// Remove tail

gameBoard[snakes[s].Tail.Position.Y][snakes[s].Tail.Position.X] = false;

movecursor(snakes[s].Tail.Position);

/\*

if (!cheats[Fisa])

{\*/

setforeground(Black);

putchar(snakes[s].Visual.Char);

setforeground(White);

/\*}

else

{

putchar(' ');

}\*/

// Change tail direction

if (snakes[s].Tail.Position == snakes[s].Bend.Vector[snakes[s].Bend.Active].Position)

{

snakes[s].Tail.Direction = snakes[s].Bend.Vector[snakes[s].Bend.Active++].Direction;

if (snakes[s].Bend.Active == SNAKE\_LENGTH\_MAX)

{

snakes[s].Bend.Active = 0;

}

}

// Change tail position

if (snakes[s].Tail.Direction == CONTROLS[s][KUp])

{

snakes[s].Tail.Position.Y--;

}

else if (snakes[s].Tail.Direction == CONTROLS[s][KDown])

{

snakes[s].Tail.Position.Y++;

}

else if (snakes[s].Tail.Direction == CONTROLS[s][KLeft])

{

snakes[s].Tail.Position.X--;

}

else if (snakes[s].Tail.Direction == CONTROLS[s][KRight])

{

snakes[s].Tail.Position.X++;

}

// Move tail to opposite console edges

if (!borders)

{

if (snakes[s].Tail.Position.X < 0)

{

snakes[s].Tail.Position.X = forcedConsoleSize.X;

}

else if (snakes[s].Tail.Position.X > forcedConsoleSize.X)

{

snakes[s].Tail.Position.X = 0;

}

else if (snakes[s].Tail.Position.Y < 0)

{

snakes[s].Tail.Position.Y = forcedConsoleSize.Y;

}

else if (snakes[s].Tail.Position.Y > forcedConsoleSize.Y)

{

snakes[s].Tail.Position.Y = 0;

}

}

}

movecursor(COORD\_ORIGIN);

}

}

Sleep(sleep);

// Get new directions for heads

// Not really efficient, should be rewrited

gotNewDirection[0] = false;

gotNewDirection[1] = false;

if (kbhit())

{

key = toupper(getch());

if (!snakes[0].Collision && IF\_WASD(key) && key != (snakes[0].Head.Direction == 'W' ? 'S' : (snakes[0].Head.Direction == 'S' ? 'W' : (snakes[0].Head.Direction == 'A' ? 'D' : 'A'))))

{

// Assign new bend

snakes[0].Head.Direction = snakes[0].Bend.Vector[snakes[0].Bend.ToAssign].Direction = key;

snakes[0].Bend.Vector[snakes[0].Bend.ToAssign++].Position = snakes[0].Head.Position;

if (snakes[0].Bend.ToAssign == SNAKE\_LENGTH\_MAX)

{

snakes[0].Bend.ToAssign = 0;

}

gotNewDirection[0] = true;

}

else if (key == -32 || key == 224)

{

key = getch();

if (!snakes[1].Collision && IF\_ARROW(key) && key != (snakes[1].Head.Direction == Up ? Down : (snakes[1].Head.Direction == Down ? Up : (snakes[1].Head.Direction == Left ? Right : Left))))

{

// Assign new bend

snakes[1].Head.Direction = snakes[1].Bend.Vector[snakes[1].Bend.ToAssign].Direction = key;

snakes[1].Bend.Vector[snakes[1].Bend.ToAssign++].Position = snakes[1].Head.Position;

if (snakes[1].Bend.ToAssign == SNAKE\_LENGTH\_MAX)

{

snakes[1].Bend.ToAssign = 0;

}

gotNewDirection[1] = true;

}

}

else if (key == Esc)

{

ResumeThread(sizecheckthreadHandle);

ResumeThread(cheatpromptthreadHandle);

SetConsoleTitle(gameName);

setforeground(White);

setbackground(Black);

if (pausemenu(true))

{

return;

}

SuspendThread(sizecheckthreadHandle);

SuspendThread(cheatpromptthreadHandle);

livetitle("Player 1", snakes[0].Active, "Player 2", snakes[1].Active);

/\*

if (cheats[Fisa] != lastFisaState)

{

recreateboard(gameBoard[0], snake.Visual, berry, isSpecialShown, shownSpecial);

lastFisaState = cheats[Fisa];

}\*/

}

#ifdef DEBUG

else if (key == Space)

{

while (getch() != Space);

}

#endif

}

if (!snakes[0].Collision && !snakes[1].Collision && kbhit())

{

key = toupper(getch());

if (!gotNewDirection[0] && IF\_WASD(key) && key != (snakes[0].Head.Direction == 'W' ? 'S' : (snakes[0].Head.Direction == 'S' ? 'W' : (snakes[0].Head.Direction == 'A' ? 'D' : 'A'))))

{

// Assign new bend

snakes[0].Head.Direction = snakes[0].Bend.Vector[snakes[0].Bend.ToAssign].Direction = key;

snakes[0].Bend.Vector[snakes[0].Bend.ToAssign++].Position = snakes[0].Head.Position;

if (snakes[0].Bend.ToAssign == SNAKE\_LENGTH\_MAX)

{

snakes[0].Bend.ToAssign = 0;

}

gotNewDirection[0] = true;

}

else if (key == -32 || key == 224)

{

key = getch();

if (!snakes[1].Collision && IF\_ARROW(key) && key != (snakes[1].Head.Direction == Up ? Down : (snakes[1].Head.Direction == Down ? Up : (snakes[1].Head.Direction == Left ? Right : Left))))

{

// Assign new bend

snakes[1].Head.Direction = snakes[1].Bend.Vector[snakes[1].Bend.ToAssign].Direction = key;

snakes[1].Bend.Vector[snakes[1].Bend.ToAssign++].Position = snakes[1].Head.Position;

if (snakes[1].Bend.ToAssign == SNAKE\_LENGTH\_MAX)

{

snakes[1].Bend.ToAssign = 0;

}

gotNewDirection[1] = true;

}

}

else if (key == Esc)

{

ResumeThread(sizecheckthreadHandle);

ResumeThread(cheatpromptthreadHandle);

SetConsoleTitle(gameName);

setforeground(White);

setbackground(Black);

if (pausemenu(true))

{

return;

}

SuspendThread(sizecheckthreadHandle);

SuspendThread(cheatpromptthreadHandle);

livetitle("Player 1", snakes[0].Active, "Player 2", snakes[1].Active);

/\*

if (cheats[Fisa] != lastFisaState)

{

recreateboard(gameBoard[0], snake.Visual, berry, isSpecialShown, shownSpecial);

lastFisaState = cheats[Fisa];

}\*/

}

#ifdef DEBUG

else if (key == Space)

{

while (getch() != Space);

}

#endif

}

for (s = 0; s <= 1; s++)

{

if (!snakes[s].Collision)

{

// Change snake position

if (snakes[s].Head.Direction == CONTROLS[s][KUp])

{

snakes[s].Head.Position.Y--;

}

else if (snakes[s].Head.Direction == CONTROLS[s][KDown])

{

snakes[s].Head.Position.Y++;

}

else if (snakes[s].Head.Direction == CONTROLS[s][KLeft])

{

snakes[s].Head.Position.X--;

}

else if (snakes[s].Head.Direction == CONTROLS[s][KRight])

{

snakes[s].Head.Position.X++;

}

if (!borders)

{

// Move head to opposite console edge

if (snakes[s].Head.Position.X < 0)

{

snakes[s].Head.Position.X = forcedConsoleSize.X;

}

else if (snakes[s].Head.Position.X > forcedConsoleSize.X)

{

snakes[s].Head.Position.X = 0;

}

else if (snakes[s].Head.Position.Y < 0)

{

snakes[s].Head.Position.Y = forcedConsoleSize.Y;

}

else if (snakes[s].Head.Position.Y > forcedConsoleSize.Y)

{

snakes[s].Head.Position.Y = 0;

}

}

// Eat berry

for (b = 0; b <= 1; b++)

{

if (snakes[s].Head.Position == berries[b].Position)

{

snakes[s].Active += (cheats[Hamster] ? CHEATS\_HAMSTER : 1);

livetitle("Player 1", snakes[0].Active, "Player 2", snakes[1].Active);

do

{

newberry(&berries[b], gameBoard[0]);

} while (berries[b].Position == berries[1 - b].Position);

}

}

// Specials

if (!isSpecialShown && SPECIAL\_SPAWN)

{

// Create new special berry

shownSpecial = (SPECIAL)(rand() % (sizeof(specials) / sizeof(specials[0])));

newberry(&specials[shownSpecial], gameBoard[0]);

isSpecialShown = true;

}

else if (isSpecialShown)

{

if (snakes[s].Head.Position == specials[shownSpecial].Position)

{

// Eat special berry

switch (shownSpecial)

{

case Bonus:

snakes[s].Active += SPECIAL\_BONUS;

livetitle("Player 1", snakes[0].Active, "Player 2", snakes[1].Active);

break;

case Mine:

snakes[s].Collision = true;

break;

/\*

case Slowdown:

// TODO: slowdown

break;

case Poison:

// TODO: poison

break;\*/

}

isSpecialShown = false;

}

else if (SPECIAL\_SPAWN)

{

// Remove special berry

movecursor(specials[shownSpecial].Position);

putchar(' ');

isSpecialShown = false;

}

}

// Collision

if (!snakes[s].Collision)

{

snakes[s].Collision = borders && !(snakes[s].Head.Position >= COORD\_ORIGIN && snakes[s].Head.Position <= forcedConsoleSize) || gameBoard[snakes[s].Head.Position.Y][snakes[s].Head.Position.X];

}

}

}

// Game end

if ((snakes[0].Collision || snakes[0].Active == SNAKE\_LENGTH\_MAX) && (snakes[1].Collision || snakes[1].Active == SNAKE\_LENGTH\_MAX))

{

bool won[2] = { snakes[0].Active == SNAKE\_LENGTH\_MAX, snakes[1].Active == SNAKE\_LENGTH\_MAX };

int scores[2] = { snakes[0].Active - SNAKE\_LENGTH\_START, snakes[1].Active - SNAKE\_LENGTH\_START};

SetConsoleTitle(gameName);

ResumeThread(sizecheckthreadHandle);

ResumeThread(cheatpromptthreadHandle);

setforeground(White);

setbackground(Black);

showscore(won, scores, sleep == SLEEP\_EASY ? Easy : (sleep == SLEEP\_MEDIUM ? Medium : (sleep == SLEEP\_HARD ? Hard : Custom)), borders ? "Normal" : "Borderless");

return;

}

}

}

## unigfcs.h

/\*

Name: unigfcs.h (Universal game functions) v1.4

Copyright: (c) 2016 Marian Dolinský

Author: Marian Dolinský

Date: 30/05/16 05:45

Description: Lots of useful functions for creating games in Windows console.

DEBUG - enables DEBUG mode

DISABLE\_HOTSEAT - disables hotseat from mainmenu

DISABLE\_DIFFICULTY - disables difficulty menu

DISABLE\_CUSTOMDIFFICULTY - disables custom difficulty option

DISABLE\_CHANGELOG - disables changelogs

DISABLE\_CHEATS - disables all cheats and functions used with cheats

DISABLE\_BUFFERASWINDOW - buffer height will be only by 1 greater than window height

\*/

#ifndef UNIGFCS\_H

#define UNIGFCS\_H

#include "Project Superior CFC\superior.hpp"

#include <time.h>

#include <math.h>

#if !defined DISABLE\_CHANGELOG && !defined CHANGELOG\_COLUMNS

#error CHANGELOG\_COLUMNS not defined

#endif

#ifdef DEBUG

#warning DEBUG enabled

#endif

#define IF\_ARROW(KEY) (KEY == Up || KEY == Down || KEY == Left || KEY == Right)

#define IF\_WASD(KEY) (KEY == 'W' || KEY == 'S' || KEY == 'A' || KEY == 'D')

// Using macros so don't have to pass so much arguments

#ifdef DEBUG

#define DEBUG\_CHEATS \

printf("\n"); \

for (int i = 0; i < sizeof(cheats) / sizeof(cheats[0]) - 1; i++) \

{ \

printf("cheats[%d]: %d | ", i, cheats[i]); \

} \

printf("cheats[%d]: %d", sizeof(cheats) / sizeof(cheats[0]) - 1, cheats[sizeof(cheats) / sizeof(cheats[0]) - 1])

#define DEBUG\_KEY(KEY) \

printf("key: %c", printarrow(KEY))

// Cannot use #ifdef etc in MACROs

#ifndef DISABLE\_CHEATS

#define DEBUG\_MAIN \

movecursor(COORD\_ORIGIN); \

printf("selection: %d | ", selection); \

printf("cursorPosition: %\*d %\*d | ", consoleSizeXPrecision, cursorPosition.X, consoleSizeXPrecision, cursorPosition.Y); \

printf("consoleSizeChanged: %d | ", consoleSizeChanged); \

DEBUG\_KEY(key); \

DEBUG\_CHEATS

#else

#define DEBUG\_MAIN \

movecursor(COORD\_ORIGIN); \

printf("selection: %d | ", selection); \

printf("cursorPosition: %\*d %\*d | ", consoleSizeXPrecision, cursorPosition.X, consoleSizeXPrecision, cursorPosition.Y); \

printf("consoleSizeChanged: %d | ", consoleSizeChanged); \

DEBUG\_KEY(key)

#endif

#endif

typedef enum

{

Easy,

Medium,

Hard,

Custom,

NoDifficulty

} DIFFICULTY;

typedef enum

{

Restart,

Menu,

Quit

} QUITACTION;

const char \*gameName, \*gameVersion, \*gameBranch;

#ifndef DISABLE\_CHANGELOG

const char CHANGELOG\_UNIGFCS[][CHANGELOG\_COLUMNS] =

{

{ "1.4: (5/30/2016)" },

{ "- Using Project Superior CFC v1.0 beta 1" },

{ "1.3: (5/29/2016)" },

{ "- Real time responsivity" },

{ "- Added custom difficulty" },

{ "- Added timer" },

{ "- Added cheats support" },

{ "- Added ability to move cursor" },

{ "- Added custom console title" },

{ "- Fully removed screen flashing" },

{ "- Lots of improvements" },

{ "- Lots of code optimalizations" },

{ "- Lots of new features" },

{ "- Some bugs were fixed" },

{ "1.2: (12/18/2015)" },

{ "- Added macros for ENTER, ESC and SPACE" },

{ "- Added unigfcs.h changelog" },

{ "- Improved responsive design" },

{ "- Some bugs were fixed" },

{ "1.1:" },

{ "- Code were optimized" },

{ "- Some bugs were fixed" },

{ "1.0: (11/24/2015)" },

{ "- First release" },

{ "- Universal game menus" },

{ "- Custom font" }

};

#endif

#ifndef DISABLE\_BUFFERASWINDOW

const int BUFFER\_ADDITION = 1;

#else

const int BUFFER\_ADDITION = 2;

#endif

const int MENU\_SLEEP = 15;

const UINT CP\_OEM = 437;

#ifndef DISABLE\_CHEATS

bool cheatActivated, showCheatPrompt, consoleSizeChangedCheatPrompt;

#endif

bool consoleSizeChanged;

bool cheats[sizeof(CHEATS) / sizeof(CHEATS[0])];

#ifdef DEBUG

int consoleSizeXPrecision;

#endif

// Console sizes are decreased by 1 so it can be compared with COORDs

COORD consoleSize, minimalConsoleSize, forcedConsoleSize;

HANDLE sizecheckthreadHandle = NULL;

#ifndef DISABLE\_CHEATS

HANDLE cheatpromptthreadHandle = NULL;

#endif

// Applicable for bool-returning functions: returns true to make caller return too

void initialize(const char \*name, const char \*version, const char \*branch, int minimalConsoleWidth, int minimalConsoleHeight);

DWORD WINAPI consolesizecheckthread(LPVOID lpvoid);

#ifndef DISABLE\_CHEATS

DWORD WINAPI cheatpromptthread(LPVOID lpvoid);

#endif

void setconsolesize(COORD size);

// Returns true if consoleSize was changed

bool assignconsolesize();

void forceconsolesize();

void copybuffer(const HANDLE \*dest, const HANDLE \*source);

void setforeground(COLOR foreground);

void setbackground(COLOR background);

char getcheat(char key);

#ifdef DEBUG

char printarrow(char arrow);

#endif

void mainmenu(COLOR titleColor);

bool difficultyselection(bool hotseat);

void wheelselection(COORD position, int selected, int min, int max, int step);

void about();

#ifndef DISABLE\_CHANGELOG

void changelog(const char CHANGELOG[][CHANGELOG\_COLUMNS], int size);

#endif

bool pausemenu(bool hotseat);

bool quitdialog(QUITACTION action, bool playing, bool hotseat);

void showscore(bool won, int score, DIFFICULTY difficulty, const char \*gameMode);

void showscore(bool won[2], int score[2], DIFFICULTY difficulty, const char \*gameMode);

void timer(bool isColored);

// WARNING!!! Prints only alphabet, numbers and space

void bigtext(const char \*text);

// These must be in each game

void controls();

bool game(DIFFICULTY difficulty, bool hotseat);

//void customdifficultyselection(); - returned data type depends on game

void initialize(const char \*name, const char \*version, const char \*branch, int minimalConsoleWidth, int minimalConsoleHeight)

{

gameName = strdup(name);

gameVersion = strdup(version);

gameBranch = strdup(branch);

minimalConsoleSize.X = minimalConsoleWidth;

minimalConsoleSize.Y = minimalConsoleHeight;

#ifndef DEBUG

setcursor(false);

srand(time(NULL));

#else

setcursor(true);

#endif

// Assign variables

#ifndef DISABLE\_CHEATS

consoleSizeChangedCheatPrompt = true;

#endif

consoleSizeChanged = true;

memset(cheats, false, sizeof(cheats));

assignconsolesize();

sizecheckthreadHandle = CreateThread(NULL, 0, consolesizecheckthread, NULL, 0, NULL);

// Set Lucida Console font

CONSOLE\_FONT\_INFOEX font;

font.cbSize = sizeof(font);

font.nFont = 7;

font.dwFontSize.X = 12;

font.dwFontSize.Y = 16;

font.FontFamily = FF\_DONTCARE;

font.FontWeight = FW\_NORMAL;

wcscpy(font.FaceName, L"Lucida Console");

SetCurrentConsoleFontEx(GetStdHandle(STD\_OUTPUT\_HANDLE), false, &font);

// Use OEM Code Page

SetConsoleOutputCP(CP\_OEM);

SetConsoleTitle(name);

setforeground(White);

setbackground(Black);

}

DWORD WINAPI consolesizecheckthread(LPVOID lpvoid)

{

while (true)

{

if (!consoleSizeChanged)

{

consoleSizeChanged = assignconsolesize();

consoleSizeChangedCheatPrompt = consoleSizeChanged;

if (consoleSizeChanged)

{

if (consoleSize < minimalConsoleSize)

{

consoleSize = minimalConsoleSize;

setconsolesize(consoleSize);

}

else if (consoleSize.X < minimalConsoleSize.X)

{

consoleSize.X = minimalConsoleSize.X;

setconsolesize(consoleSize);

}

else if (consoleSize.Y < minimalConsoleSize.Y)

{

consoleSize.Y = minimalConsoleSize.Y;

setconsolesize(consoleSize);

}

}

}

Sleep(50);

}

return EXIT\_FAILURE;

}

#ifndef DISABLE\_CHEATS

DWORD WINAPI cheatpromptthread(LPVOID lpvoid)

{

char \*message = NULL;

int timer, length = 0;

DWORD attribsWritten, charsWritten;

COLOR foregroundBackup, backgroundBackup;

while (true)

{

if (showCheatPrompt || consoleSizeChangedCheatPrompt)

{

showCheatPrompt = false;

consoleSizeChangedCheatPrompt = false;

timer = 0;

FillConsoleOutputAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), backgroundColor + (backgroundColor << 4), length, COORD\_ORIGIN, &attribsWritten);

message = strdup(cheatActivated ? " Cheat activated " : " Cheat deactivated ");

length = strlen(message);

// Print message

foregroundBackup = foregroundColor;

backgroundBackup = backgroundColor;

setforeground(Black);

setbackground(Gray);

movecursor(COORD\_ORIGIN);

fputs(message, stdout);

setforeground(foregroundBackup);

setbackground(backgroundBackup);

// char \*strdup(const char \*src)

// {

// char \*output = (char \*)malloc(strlen(src) + 1);

//

// if (output == NULL)

// {

// return NULL;

// }

//

// strcpy(output, src);

// return output;

// }

//

// ^ this is how strdup works

// So the string returned from strdup is dynamically allocated and should be freed

// If not it'll be allocated 'till program closes and another apps couldn't use it

free(message);

}

Sleep(200);

timer += 200;

if (timer == 2000)

{

FillConsoleOutputAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), backgroundColor + (backgroundColor << 4), length, COORD\_ORIGIN, &attribsWritten);

SuspendThread(cheatpromptthreadHandle);

}

}

return EXIT\_FAILURE;

}

#endif

void setconsolesize(COORD size)

{

SMALL\_RECT windowSize = { 0 };

// Cannot set greater size than (buffer - 1) and buffer size cannot be less than window size

if (consoleSize.X < size.X || consoleSize.Y < size.Y)

{

windowSize.Right = consoleSize.X < size.X ? consoleSize.X : size.X;

windowSize.Bottom = consoleSize.Y < size.Y ? consoleSize.Y : size.Y;

SetConsoleWindowInfo(GetStdHandle(STD\_OUTPUT\_HANDLE), true, &windowSize);

}

COORD bufferSize = { size.X + 1, size.Y + BUFFER\_ADDITION };

SetConsoleScreenBufferSize(GetStdHandle(STD\_OUTPUT\_HANDLE), bufferSize);

windowSize.Right = size.X;

windowSize.Bottom = size.Y;

SetConsoleWindowInfo(GetStdHandle(STD\_OUTPUT\_HANDLE), true, &windowSize);

assignconsolesize();

consoleSizeChanged = true;

}

bool assignconsolesize()

{

COORD temp;

CONSOLE\_SCREEN\_BUFFER\_INFO stdOutInfo;

// Get console sizes and assign consoleSize

GetConsoleScreenBufferInfo(GetStdHandle(STD\_OUTPUT\_HANDLE), &stdOutInfo);

temp.X = stdOutInfo.srWindow.Right - stdOutInfo.srWindow.Left + 1;

temp.Y = stdOutInfo.srWindow.Bottom - stdOutInfo.srWindow.Top + BUFFER\_ADDITION;

// Do not write in consoleSize if console window size does not really changed

if (temp.X - 1 != consoleSize.X || temp.Y - BUFFER\_ADDITION != consoleSize.Y)

{

SetConsoleScreenBufferSize(GetStdHandle(STD\_OUTPUT\_HANDLE), temp);

consoleSize.X = temp.X - 1;

consoleSize.Y = temp.Y - BUFFER\_ADDITION;

#ifdef DEBUG

consoleSizeXPrecision = (int)floor(log10(consoleSize.X)) + 1;

#endif

return true;

}

return false;

}

void forceconsolesize()

{

assignconsolesize();

if (consoleSize != forcedConsoleSize)

{

setconsolesize(forcedConsoleSize);

}

}

void copybuffer(const HANDLE \*dest, const HANDLE \*source)

{

COORD bufferSize = { consoleSize.X + 1, consoleSize.Y + 1 };

CHAR\_INFO charInfo[bufferSize.Y][bufferSize.X];

SMALL\_RECT region = { 0, 0, consoleSize.X, consoleSize.Y };

ReadConsoleOutput(\*source, charInfo[0], bufferSize, COORD\_ORIGIN, &region);

WriteConsoleOutput(\*dest, charInfo[0], bufferSize, COORD\_ORIGIN, &region);

}

char getcheat(char key)

{

#ifndef DISABLE\_CHEATS

if (key == -32 || key == 224)

{

return key;

}

for (int i = 0; i < sizeof(CHEATS) / sizeof(CHEATS[0]); i++)

{

// Compare key with first letter on every line

if (key == CHEATS[i][0] || key == toupper(CHEATS[i][0]))

{

// If key equals first letter continue reading keys for cheat

for (int j = 1; j < strlen(CHEATS[i]); j++)

{

key = getch();

if (key != CHEATS[i][j] && key != toupper(CHEATS[i][j]))

{

return key;

}

}

cheats[i] = !cheats[i];

#ifndef DEBUG

showCheatPrompt = true;

cheatActivated = cheats[i];

if (cheatpromptthreadHandle == NULL)

{

cheatpromptthreadHandle = CreateThread(NULL, 0, cheatpromptthread, NULL, 0, NULL);

}

else

{

ResumeThread(cheatpromptthreadHandle);

}

#endif

return key;

}

}

#endif //DISABLE\_CHEATS

return key;

}

#ifdef DEBUG

char printarrow(char arrow)

{

switch(arrow)

{

case Up: return 'U';

case Down: return 'D';

case Left: return 'L';

case Right: return 'R';

default: return 'N';

}

}

#endif

void mainmenu(COLOR titleColor)

{

#ifndef DISABLE\_HOTSEAT

const int SELECTION\_MAX = 4;

#else

const int SELECTION\_MAX = 3;

#endif

const int CENTERING = 10;

bool renderMain = true, renderSelection = true;

char key;

int selection = 0;

COORD cursorPosition;

while (true)

{

if (consoleSizeChanged || renderMain)

{

cls();

vcenter(13);

setforeground(titleColor);

bigtext(gameName);

setforeground(White);

#ifdef DEBUG

putchar('\n');

hcenter(-20, "DEBUG MODE\n");

putchar('\n');

#else

printf("\n\n\n");

#endif

cursorPosition = hcenter(CENTERING, "Start game\n");

cursorPosition.X -= 2;

#ifndef DISABLE\_HOTSEAT

hcenter(CENTERING, "Hotseat\n");

#endif

hcenter(CENTERING, "Controls\n");

hcenter(CENTERING, "About\n");

putchar('\n');

hcenter(CENTERING, "Quit game");

consoleSizeChanged = false;

renderMain = false;

renderSelection = true;

}

#ifdef DEBUG

DEBUG\_MAIN;

#endif

if (renderSelection)

{

for (int i = (selection == SELECTION\_MAX ? -2 : -1); i <= (selection == SELECTION\_MAX - 1 ? 2 : 1); i++)

{

movecursor(cursorPosition.X, cursorPosition.Y + selection + (selection == SELECTION\_MAX ? 1 : 0) + i);

putchar(' ');

}

movecursor(cursorPosition.X, cursorPosition.Y + selection + (selection == SELECTION\_MAX ? 1 : 0));

putchar('>');

renderSelection = false;

}

if (kbhit())

{

key = getkey(getcheat(getch()));

if ((key == Enter && selection == SELECTION\_MAX) || key == Esc)

{

quitdialog(Quit, false, false);

renderMain = true;

}

else if (key == Enter)

{

if (selection == 0)

{

difficultyselection(false);

}

#ifndef DISABLE\_HOTSEAT

else if (selection == 1)

{

difficultyselection(true);

}

#endif

else if (selection == SELECTION\_MAX - 2)

{

controls();

}

else //if (selection == SELECTION\_MAX - 1)

{

about();

}

renderMain = true;

}

else if (key == Up && selection != 0)

{

selection--;

renderSelection = true;

}

else if (key == Down && selection != SELECTION\_MAX)

{

selection++;

renderSelection = true;

}

}

Sleep(MENU\_SLEEP);

}

}

bool difficultyselection(bool hotseat)

{

#ifdef DISABLE\_DIFFICULTY

game(NoDifficulty, hotseat);

#else

#ifndef DISABLE\_CUSTOMDIFFICULTY

const int SELECTION\_MAX = 4;

#else

const int SELECTION\_MAX = 3;

#endif

const int CENTERING = 6;

bool renderMain = true, renderSelection = true;

char key;

int selection = 1;

COORD cursorPosition;

while (true)

{

if (consoleSizeChanged || renderMain)

{

cls();

vcenter(7);

hcenter("CHOOSE DIFFICULTY");

fputs("\n\n", stdout);

cursorPosition = hcenter(CENTERING, "Easy\n");

cursorPosition.X -= 2;

hcenter(CENTERING, "Medium\n");

hcenter(CENTERING, "Hard\n");

putchar('\n');

#ifndef DISABLE\_CUSTOMDIFFICULTY

hcenter(CENTERING, "Custom\n");

#endif

hcenter(CENTERING, "Back");

consoleSizeChanged = false;

renderMain = false;

renderSelection = true;

}

#ifdef DEBUG

DEBUG\_MAIN;

#endif

if (renderSelection)

{

for (int i = (selection == 3 ? -2 : -1); i <= (selection == 2 ? 2 : 1); i++)

{

movecursor(cursorPosition.X, cursorPosition.Y + selection + (selection >= 3 ? 1 : 0) + i);

putchar(' ');

}

movecursor(cursorPosition.X, cursorPosition.Y + selection + (selection >= 3 ? 1 : 0));

putchar('>');

renderSelection = false;

}

if (kbhit())

{

key = getkey(getcheat(getch()));

if ((key == Enter && selection == SELECTION\_MAX) || key == Esc)

{

return false;

}

else if (key == Enter)

{

if (game((DIFFICULTY)selection, hotseat))

{

return true;

}

renderMain = true;

}

else if (key == Up && selection != 0)

{

selection--;

renderSelection = true;

}

else if (key == Down && selection != SELECTION\_MAX)

{

selection++;

renderSelection = true;

}

}

Sleep(MENU\_SLEEP);

}

#endif

}

void wheelselection(COORD position, int selected, int min, int max, int step)

{

int temp;

COLOR foregroundBackup = foregroundColor;

movecursor(position);

hcenter(19);

for (int i = -2; i <= 2; i++)

{

setforeground(i == 0 ? White : (i == -1 || i == 1 ? Gray : DarkGray));

temp = selected + (i \* step);

printf("%3d ", (temp >= min && temp <= max ? temp : (temp < min ? max - (min - temp) + step : min + (temp - max) - step)));

}

setforeground(foregroundBackup);

}

void about()

{

const int CENTERING = 24;

bool renderMain = true, renderSelection = true;

bool printGameBranch = gameBranch != NULL && strcmp(gameBranch, "");

char key;

int selection = 2;

COORD cursorPosition;

while (true)

{

if (consoleSizeChanged || renderMain)

{

cls();

#ifndef DISABLE\_CHANGELOG

vcenter(10 + (printGameBranch ? 1 : 0));

#else

vcenter(7 + (printGameBranch ? 1 : 0));

#endif

hcenter("ABOUT");

fputs("\n\n", stdout);

hcenter(CENTERING, "%s %s\n", gameName, gameVersion);

#ifndef DEBUG

if (printGameBranch)

{

#endif

hcenter(CENTERING, "Current branch: %s", gameBranch);

#ifndef DEBUG

}

#endif

#ifdef DEBUG

printf("%s(DEBUG)", (printGameBranch ? " " : ""));

#endif

putchar('\n');

hcenter(CENTERING, "Using unigfcs.h v1.4\n");

hcenter(CENTERING, "Using Project Superior CFC v1.0 beta 1\n");

hcenter(CENTERING, "Current console size: %dx%d\n\n", consoleSize.X + 1, consoleSize.Y + 1);

SetConsoleOutputCP(1250);

hcenter(CENTERING, "(c) 2016 Marian Dolinský\n\n");

SetConsoleOutputCP(CP\_OEM);

#ifndef DISABLE\_CHANGELOG

cursorPosition = hcenter(9, "Changelog\n");

cursorPosition.X -= 2;

hcenter(9, "Changelog (unigfcs.h)\n");

hcenter(9, "Back");

#else

hcenter("> Back");

#endif

consoleSizeChanged = false;

renderMain = false;

renderSelection = true;

}

#ifdef DEBUG

DEBUG\_MAIN;

#endif

#ifndef DISABLE\_CHANGELOG

if (renderSelection)

{

for (int i = -1; i <= 1; i += 2)

{

movecursor(cursorPosition.X, cursorPosition.Y + selection + i);

putchar(' ');

}

movecursor(cursorPosition.X, cursorPosition.Y + selection);

putchar('>');

renderSelection = false;

}

if (kbhit())

{

key = getkey(getcheat(getch()));

if ((key == Enter && selection == 2) || key == Esc)

{

return;

}

else if (key == Enter)

{

if (selection == 0)

{

changelog(CHANGELOG\_GAME, sizeof(CHANGELOG\_GAME));

}

else //if (selection == 1)

{

changelog(CHANGELOG\_UNIGFCS, sizeof(CHANGELOG\_UNIGFCS));

}

selection = 2;

renderMain = true;

}

else if (key == Up && selection != 0)

{

selection--;

renderSelection = true;

}

else if (key == Down && selection != 2)

{

selection++;

renderSelection = true;

}

}

Sleep(MENU\_SLEEP);

#else

if (kbhit())

{

key = getch();

if (key == Enter || key == Esc)

{

return;

}

}

Sleep(MENU\_SLEEP);

#endif

}

}

#ifndef DISABLE\_CHANGELOG

void changelog(const char CHANGELOG[][CHANGELOG\_COLUMNS], int size)

{

const int CENTERING = 37;

const int ROWS\_MAX = 15;

bool renderMain = true, renderChangelog = true;

char key;

int row = 0;

COORD changelogPosition;

while (true)

{

if (consoleSizeChanged || renderMain)

{

cls();

vcenter(ROWS\_MAX + 4);

hcenter(CHANGELOG == CHANGELOG\_GAME ? "CHANGELOG" : "CHANGELOG (UNIGFCS.H)");

fputs("\n\n", stdout);

changelogPosition = getcursorposition();

movecursor(0, changelogPosition.Y + ROWS\_MAX + 1);

hcenter("> Back ");

consoleSizeChanged = false;

renderMain = false;

renderChangelog = true;

}

if (renderChangelog)

{

movecursor(changelogPosition);

for (int i = 0; i < ROWS\_MAX; i++)

{

hcenter(CENTERING, "%s%-\*s \n", (CHANGELOG[row + i][0] == '-' ? " " : ""), CHANGELOG\_COLUMNS, CHANGELOG[row + i]);

}

renderChangelog = false;

}

#ifdef DEBUG

movecursor(COORD\_ORIGIN);

printf("row: %2d | ", row);

printf("size: %d | ", size);

DEBUG\_KEY(key);

DEBUG\_CHEATS;

#endif

if (kbhit())

{

key = getkey(getcheat(getch()));

if (key == Enter || key == Esc)

{

return;

}

else if (key == Up && row != 0)

{

row--;

renderChangelog = true;

}

else if (key == Down && row != (size / CHANGELOG\_COLUMNS) - ROWS\_MAX)

{

row++;

renderChangelog = true;

}

}

Sleep(MENU\_SLEEP);

}

}

#endif

bool pausemenu(bool hotseat)

{

const int CENTERING = 8;

bool renderMain = true, renderSelection = true;

char key;

int selection = 0;

COORD cursorPosition;

HANDLE gameBoardBuffer = CreateConsoleScreenBuffer(GENERIC\_READ | GENERIC\_WRITE, 0, NULL, CONSOLE\_TEXTMODE\_BUFFER, NULL);

HANDLE stdOut = GetStdHandle(STD\_OUTPUT\_HANDLE);

copybuffer(&gameBoardBuffer, &stdOut);

while (true)

{

if (consoleSizeChanged || renderMain)

{

cls();

vcenter(7);

hcenter("PAUSE MENU");

fputs("\n\n", stdout);

cursorPosition = hcenter(CENTERING, "Resume\n");

cursorPosition.X -= 2;

hcenter(CENTERING, "Restart\n");

hcenter(CENTERING, "Controls\n");

hcenter(CENTERING, "Go to main menu\n");

hcenter(CENTERING, "Quit game");

consoleSizeChanged = false;

renderMain = false;

renderSelection = true;

}

#ifdef DEBUG

DEBUG\_MAIN;

#endif

if (renderSelection)

{

for (int i = -1; i <= 1; i += 2)

{

movecursor(cursorPosition.X, cursorPosition.Y + selection + i);

putchar(' ');

}

movecursor(cursorPosition.X, cursorPosition.Y + selection);

putchar('>');

renderSelection = false;

}

if (kbhit())

{

key = getkey(getcheat(getch()));

if ((key == Enter && selection == 0) || key == Esc)

{

forceconsolesize();

copybuffer(&stdOut, &gameBoardBuffer);

return false;

}

else if (key == Enter)

{

if (selection == 2)

{

controls();

}

else

{

if (quitdialog(selection == 1 ? Restart : (selection == 3 ? Menu : Quit), true, hotseat))

{

return true;

}

}

renderMain = true;

}

else if (key == Up && selection != 0)

{

selection--;

renderSelection = true;

}

else if (key == Down && selection != 4)

{

selection++;

renderSelection = true;

}

}

Sleep(MENU\_SLEEP);

}

}

bool quitdialog(QUITACTION action, bool playing, bool hotseat)

{

bool renderMain = true, renderSelection = true;

char key;

int selection = 1;

COORD cursorPosition;

while (true)

{

if (consoleSizeChanged || renderMain)

{

cls();

vcenter(playing ? 4 : 3);

hcenter(action == Restart ? "Are you sure you want to restart the game?" : (action == Menu ? "Are you sure you want to go to main menu?" : "Are you sure you want to quit the game?"));

if (playing)

{

putchar('\n');

hcenter("Your game progress will be lost!");

}

fputs("\n\n", stdout);

cursorPosition = hcenter(18, "Yes");

cursorPosition.X += 14;

movecursor(cursorPosition.X + 2, cursorPosition.Y);

fputs("No", stdout);

consoleSizeChanged = false;

renderMain = false;

renderSelection = true;

}

#ifdef DEBUG

DEBUG\_MAIN;

#endif

if (renderSelection)

{

movecursor(cursorPosition.X - (selection == 1 ? 16 : 0), cursorPosition.Y);

putchar(' ');

movecursor(cursorPosition.X - (selection == 0 ? 16 : 0), cursorPosition.Y);

putchar('>');

renderSelection = false;

}

if (kbhit())

{

key = getkey(getcheat(getch()));

if ((key == Enter && selection == 1) || key == Esc)

{

return false;

}

else if (key == Enter)

{

if (action == Restart)

{

return difficultyselection(hotseat);

}

else if (action == Menu)

{

return true;

}

else //if (action == Quit)

{

cls();

exit(EXIT\_SUCCESS);

}

renderMain = true;

}

else if (key == Left && selection != 0)

{

selection--;

renderSelection = true;

}

else if (key == Right && selection != 1)

{

selection++;

renderSelection = true;

}

}

Sleep(MENU\_SLEEP);

}

}

void showscore(bool won, int score, DIFFICULTY difficulty, const char \*gameMode)

{

const int CENTERING = 7;

bool renderMain = true, renderSelection = true;

char key;

int selection = 1;

COORD cursorPosition;

while (true)

{

if (consoleSizeChanged || renderMain)

{

cls();

vcenter(14);

if (won)

{

setforeground(Green);

bigtext("YOU WON");

}

else

{

setforeground(Red);

bigtext("GAME OVER");

}

setforeground(White);

printf("\n\n\n");

hcenter(14, "Your score: %d\n\n", score);

hcenter(18, "Difficulty: %s\n", difficulty == Easy ? "Easy" : (difficulty == Medium ? "Medium" : (difficulty == Hard ? "Hard" : "Custom")));

hcenter(18, "Game mode: %s\n\n", gameMode);

cursorPosition = hcenter(CENTERING, "Restart\n");

cursorPosition.X -= 2;

hcenter(CENTERING, "Back");

consoleSizeChanged = false;

renderMain = false;

renderSelection = true;

}

#ifdef DEBUG

DEBUG\_MAIN;

#endif

if (renderSelection)

{

movecursor(cursorPosition.X, cursorPosition.Y + (selection == 0 ? 1 : 0));

putchar(' ');

movecursor(cursorPosition.X, cursorPosition.Y + selection);

putchar('>');

renderSelection = false;

}

if (kbhit())

{

key = getkey(getcheat(getch()));

if ((key == Enter && selection == 1) || key == Esc)

{

return;

}

else if (key == Enter)

{

difficultyselection(false);

return;

}

else if (key == Up && selection != 0)

{

selection--;

renderSelection = true;

}

else if (key == Down && selection != 1)

{

selection++;

renderSelection = true;

}

}

Sleep(MENU\_SLEEP);

}

}

void showscore(bool won[2], int score[2], DIFFICULTY difficulty, const char \*gameMode)

{

const int CENTERING = 7;

bool renderMain = true, renderSelection = true;

char key;

int selection = 1;

COORD cursorPosition;

while (true)

{

if (consoleSizeChanged || renderMain)

{

cls();

vcenter(15);

if (won[0] || won[1])

{

setforeground(Green);

bigtext("YOU WON");

}

else

{

setforeground(Red);

bigtext("GAME OVER");

}

setforeground(White);

printf("\n\n\n");

hcenter(27, won[0] ? "Player 1 won with score %d\n" : "Player 1 died with score %d\n", score[0]);

hcenter(27, won[1] ? "Player 2 won with score %d\n" : "Player 2 died with score %d\n\n", score[1]);

hcenter(18, "Difficulty: %s\n", difficulty == Easy ? "Easy" : (difficulty == Medium ? "Medium" : (difficulty == Hard ? "Hard" : "Custom")));

hcenter(18, "Game mode: %s\n\n", gameMode);

cursorPosition = hcenter(CENTERING, "Restart\n");

cursorPosition.X -= 2;

hcenter(CENTERING, "Back");

consoleSizeChanged = false;

renderMain = false;

renderSelection = true;

}

#ifdef DEBUG

DEBUG\_MAIN;

#endif

if (renderSelection)

{

movecursor(cursorPosition.X, cursorPosition.Y + (selection == 0 ? 1 : 0));

putchar(' ');

movecursor(cursorPosition.X, cursorPosition.Y + selection);

putchar('>');

renderSelection = false;

}

if (kbhit())

{

key = getkey(getcheat(getch()));

if ((key == Enter && selection == 1) || key == Esc)

{

return;

}

else if (key == Enter)

{

difficultyselection(true);

return;

}

else if (key == Up && selection != 0)

{

selection--;

renderSelection = true;

}

else if (key == Down && selection != 1)

{

selection++;

renderSelection = true;

}

}

Sleep(MENU\_SLEEP);

}

}

void timer(bool isColored)

{

const int SLEEP = 1100;

cls();

if (isColored)

{

setforeground(Red);

}

vcenter(5);

bigtext("3");

Sleep(SLEEP);

cls();

if (isColored)

{

setforeground(Yellow);

}

vcenter(5);

bigtext("2");

Sleep(SLEEP);

cls();

if (isColored)

{

setforeground(Green);

}

vcenter(5);

bigtext("1");

Sleep(SLEEP);

cls();

vcenter(5);

bigtext("GO!");

Sleep(SLEEP);

setforeground(White);

}

void bigtext(const char \*text)

{

// centering is assigned to 4 because of four spaces on left and on right side

// strlen(text) - 1 = count of spaces between each letters

// int centering = 4 + strlen(text) - 1;

int centering = 3 + strlen(text);

char str[strlen(text)];

strcpy(str, text);

// Rewrite str as uppercase and set centering

for (int i = 0; i < strlen(str); i++)

{

str[i] = toupper(str[i]);

if (str[i] == 'M' || str[i] == 'N' || str[i] == 'Q' || str[i] == 'T' || str[i] == 'W' || str[i] == 'X' || str[i] == 'Y')

{

centering += 8;

}

else if (str[i] == 'V' || str[i] == '0' || (str[i] >= '2' && str[i] <= '9'))

{

centering += 6;

}

else if (str[i] == '1')

{

centering += 4;

}

else if (str[i] == 'I' || str[i] == ' ')

{

centering += 2;

}

else if (str[i] == '!')

{

centering++;

}

else

{

centering += 7;

}

}

for (int i = 0; i < 5; i++)

{

hcenter(centering);

for (int j = 4; j > i; j--)

{

putchar(' ');

}

for (int j = 0; j < strlen(str); j++)

{

switch(str[j])

{

case ' ':

fputs(" ", stdout);

break;

case 'A':

if (i == 0 || i == 2)

{

fputs("///////", stdout);

}

else

{

fputs("// //", stdout);

}

break;

case 'B':

if (i == 1 || i == 3)

{

fputs("// //", stdout);

}

else

{

fputs("////// ", stdout);

}

break;

case 'C':

if (i == 0 || i == 4)

{

fputs("///////", stdout);

}

else

{

fputs("// ", stdout);

}

break;

case 'D':

if (i == 0 || i == 4)

{

fputs("////// ", stdout);

}

else

{

fputs("// //", stdout);

}

break;

case 'E':

if (i == 0 || i == 4)

{

fputs("///////", stdout);

}

else if (i == 2)

{

fputs("///// ", stdout);

}

else

{

fputs("// ", stdout);

}

break;

case 'F':

if (i == 0)

{

fputs("///////", stdout);

}

else if (i == 2)

{

fputs("///// ", stdout);

}

else

{

fputs("// ", stdout);

}

break;

case 'G':

if (i == 1)

{

fputs("// ", stdout);

}

else if (i == 2)

{

fputs("// ////", stdout);

}

else if (i == 3)

{

fputs("// //", stdout);

}

else

{

fputs("///////", stdout);

}

break;

case 'H':

if (i != 2)

{

fputs("// //", stdout);

}

else

{

fputs("///////", stdout);

}

break;

case 'I':

fputs("//", stdout);

break;

case 'J':

if (i < 3)

{

fputs(" //", stdout);

}

else if (i == 3)

{

fputs("// //", stdout);

}

else

{

fputs("///////", stdout);

}

break;

case 'K':

if (i == 0 || i == 4)

{

fputs("// //", stdout);

}

else if (i == 2)

{

fputs("/// ", stdout);

}

else

{

fputs("// // ", stdout);

}

break;

case 'L':

if (i < 4)

{

fputs("// ", stdout);

}

else

{

fputs("///////", stdout);

}

break;

case 'M':

if (i == 1)

{

fputs("/// ///", stdout);

}

else if (i == 2)

{

fputs("// // //", stdout);

}

else

{

fputs("// //", stdout);

}

break;

case 'N':

if (i == 1)

{

fputs("/// //", stdout);

}

else if (i == 2)

{

fputs("// // //", stdout);

}

else if (i == 3)

{

fputs("// ///", stdout);

}

else

{

fputs("// //", stdout);

}

break;

case 'O':

if (i == 0 || i == 4)

{

fputs("///////", stdout);

}

else

{

fputs("// //", stdout);

}

break;

case 'P':

if (i == 0 || i == 2)

{

fputs("///////", stdout);

}

else if (i == 1)

{

fputs("// //", stdout);

}

else

{

fputs("// ", stdout);

}

break;

case 'Q':

if (i == 0)

{

fputs("/////// ", stdout);

}

else if (i == 3)

{

fputs("// /// ", stdout);

}

else if (i == 4)

{

fputs("////////", stdout);

}

else

{

fputs("// // ", stdout);

}

break;

case 'R':

if (i == 0)

{

fputs("///////", stdout);

}

else if (i == 2)

{

fputs("////// ", stdout);

}

else

{

fputs("// //", stdout);

}

break;

case 'S':

if (i == 1)

{

fputs("// ", stdout);

}

else if (i == 3)

{

fputs(" //", stdout);

}

else

{

fputs("///////", stdout);

}

break;

case 'T':

if (i != 0)

{

fputs(" // ", stdout);

}

else

{

fputs("////////", stdout);

}

break;

case 'U':

if (i != 4)

{

fputs("// //", stdout);

}

else

{

fputs("///////", stdout);

}

break;

case 'V':

if (i != 4)

{

fputs("// //", stdout);

}

else

{

fputs(" // ", stdout);

}

break;

case 'W':

if (i == 0)

{

fputs("// //", stdout);

}

else if (i == 4)

{

fputs(" // // ", stdout);

}

else

{

fputs("// // //", stdout);

}

break;

case 'X':

if (i == 0 || i == 4)

{

fputs("// //", stdout);

}

else if (i == 2)

{

fputs(" // ", stdout);

}

else

{

fputs(" // // ", stdout);

}

break;

case 'Y':

if (i == 0)

{

fputs("// //", stdout);

}

else if (i == 1)

{

fputs(" // // ", stdout);

}

else

{

fputs(" // ", stdout);

}

break;

case 'Z':

if (i == 1)

{

fputs(" //", stdout);

}

else if (i == 2)

{

fputs(" /// ", stdout);

}

else if (i == 3)

{

fputs("// ", stdout);

}

else

{

fputs("///////", stdout);

}

break;

case '1':

if (i == 0)

{

fputs("////", stdout);

}

else

{

fputs(" //", stdout);

}

break;

case '2':

if (i == 0 || i == 4)

{

fputs("//////", stdout);

}

else if (i == 1)

{

fputs(" //", stdout);

}

else if (i == 2)

{

fputs(" // ", stdout);

}

else

{

fputs("// ", stdout);

}

break;

case '3':

if (i == 0 || i == 4)

{

fputs("//////", stdout);

}

else if (i == 2)

{

fputs(" /////", stdout);

}

else

{

fputs(" //", stdout);

}

break;

case '4':

if (i == 0 || i == 1)

{

fputs("// //", stdout);

}

else if (i == 2)

{

fputs("//////", stdout);

}

else

{

fputs(" //", stdout);

}

break;

case '5':

if (i == 3)

{

fputs(" //", stdout);

}

else if (i == 1)

{

fputs("// ", stdout);

}

else

{

fputs("//////", stdout);

}

break;

case '6':

if (i == 3)

{

fputs("// //", stdout);

}

else if (i == 1)

{

fputs("// ", stdout);

}

else

{

fputs("//////", stdout);

}

break;

case '7':

if (i == 0)

{

fputs("//////", stdout);

}

else if (i == 1)

{

fputs(" //", stdout);

}

else if (i == 2)

{

fputs(" // ", stdout);

}

else if (i == 3)

{

fputs(" // ", stdout);

}

else

{

fputs(" // ", stdout);

}

break;

case '8':

if (i % 2 == 0)

{

fputs("//////", stdout);

}

else

{

fputs("// //", stdout);

}

break;

case '9':

if (i % 2 == 0)

{

fputs("//////", stdout);

}

else if (i == 1)

{

fputs("// //", stdout);

}

else

{

fputs(" //", stdout);

}

break;

case '0':

if (i == 0 || i == 4)

{

fputs("//////", stdout);

}

else

{

fputs("// //", stdout);

}

break;

case '!':

if (i == 3)

{

fputs(" ", stdout);

}

else

{

fputs("//", stdout);

}

break;

default:

fputs("!ERROR!", stdout);

break;

}

if (j < strlen(str) - 1)

{

putchar(' ');

}

}

putchar('\n');

}

}

#endif

## superior.hpp

/\*

Name: superior.hpp (Project Superior Core for C) v1.0 beta 1

Copyright: (c) 2016 Marian Dolinský

Author: Marian Dolinský

Date: 29/05/16 23:44

Description: Lots of usefull functions, enumerations and structures for creating simple UI in Windows console.

TODO:

more rows in button, textblock, textbox text, in centers

listbox

combobox

textbox

fontwriter - 3d array for fonts

popups, notifications

\*/

#ifndef SUPERIOR\_HPP

#define SUPERIOR\_HPP

#include <stdio.h>

#include <stdarg.h>

#include <windows.h>

#include <conio.h>

typedef enum

{

Backspace = 8,

Enter = 13,

Esc = 27,

Space = 32,

Up = 72, // \*

Left = 75, // \*\*

Right = 77, // \*\*\* sending -32 || (unsigned)224 before these values

Down = 80, // \*\*

Delete = 127 // \*

} KEY;

typedef enum

{

SingleLine,

DoubleLine,

Block,

None

} LINETYPE;

typedef enum

{

Horizontal,

Vertical,

Edge\_Top\_Left,

Edge\_Top\_Right,

Edge\_Bottom\_Left,

Edge\_Bottom\_Right,

T\_Horizontal\_Up,

T\_Horizontal\_Down,

T\_Vertical\_Left,

T\_Vertical\_Right,

Cross

} LINESHAPE;

typedef enum

{

HLeft,

HCenter,

HRight

} HORIZONTALALIGNMENT;

typedef enum

{

VTop,

VCenter,

VBottom

} VERTICALALIGNMENT;

typedef enum

{

Visible,

Collapsed

} VISIBILITY;

typedef enum

{

Black,

DarkBlue,

DarkGreen,

DarkCyan,

DarkRed,

DarkMagenta,

DarkYellow,

Gray,

DarkGray,

Blue,

Green,

Cyan,

Red,

Magenta,

Yellow,

White

} COLOR;

typedef struct

{

char Char;

COLOR Color;

} VISUAL;

typedef struct

{

LINETYPE BorderType;

VISIBILITY Visibility;

bool BackgroundUnderBorder;

COLOR BorderColor;

COLOR Background;

int Width;

int Height;

HORIZONTALALIGNMENT HorizontalAlignment;

VERTICALALIGNMENT VerticalAlignment;

COORD Margin;

} BORDER;

typedef struct

{

char \*Text;

VISIBILITY Visibility;

COLOR Foreground;

COLOR Background;

int Width;

int Height;

HORIZONTALALIGNMENT HorizontalAlignment;

VERTICALALIGNMENT VerticalAlignment;

COORD Margin;

} TEXTBLOCK;

typedef struct

{

char \*Text;

HORIZONTALALIGNMENT HorizontalTextAlignment;

VERTICALALIGNMENT VerticalTextAlignment;

bool BackgroundUnderBorder;

bool IsEnabled;

bool IsFocused;

LINETYPE BorderType;

VISIBILITY Visibility;

COLOR Foreground;

COLOR UnfocusedForeground;

COLOR DisabledForeground;

COLOR Background;

COLOR UnfocusedBackground;

COLOR DisabledBackground;

COLOR BorderColor;

COLOR UnfocusedBorderColor;

COLOR DisabledBorderColor;

int Width;

int Height;

HORIZONTALALIGNMENT HorizontalAlignment;

VERTICALALIGNMENT VerticalAlignment;

COORD Margin;

} BUTTON;

/\* advio.hpp \*/

void cls();

char getkey(char key);

char getkey();

char \*createline(char start, char middle, char end, int countOfMiddle);

char \*createline(char ch, int count);

void drawline(char ch, int count);

COORD hcenter(int left, int right, int width);

COORD hcenter(int left, int right, int width, const char \*format, ...);

COORD hcenter(int left, int right, const char \*format, ...);

COORD hcenter(int width);

COORD hcenter(int width, const char \*format, ...);

COORD hcenter(const char \*format, ...);

COORD vcenter(int top, int bottom, int height);

COORD vcenter(int height);

COORD center(int left, int right, int width, int top, int bottom, int height);

COORD center(int left, int right, int width, int top, int bottom, int height, const char \*format, ...);

COORD center(int left, int right, int top, int bottom, const char \*format, ...);

COORD center(int width, int height);

COORD center(int width, int height, const char \*format, ...);

COORD center(const char \*format, ...);

COORD vhcenter(int left, int right, const char \*format, va\_list args);

/\* border.hpp \*/

BORDER newBORDER();

void draw(const BORDER &b);

void erase(const BORDER &b);

/\* button.hpp \*/

BUTTON newBUTTON();

void draw(const BUTTON &b);

void erase(const BUTTON &b);

/\* cnslsz.hpp \*/

COORD getconsolesize();

/\* cursor.hpp \*/

void setcursor(bool isVisible, int size);

void setcursor(bool isVisible);

COORD getcursorposition();

void movecursor(COORD position);

void movecursor(int x, int y);

/\* operators.hpp \*/

bool operator ==(COORD c1, COORD c2);

bool operator !=(COORD c1, COORD c2);

bool operator <(COORD c1, COORD c2);

bool operator >(COORD c1, COORD c2);

bool operator <=(COORD c1, COORD c2);

bool operator >=(COORD c1, COORD c2);

bool operator ==(SMALL\_RECT r1, SMALL\_RECT r2);

bool operator !=(SMALL\_RECT r1, SMALL\_RECT r2);

/\* strext.hpp \*/

char \*stradd(char \*str, char ch, int index);

char \*strrem(char \*str, int index);

char \*strndup(const char \*source, int count);

char \*strcat(char \*dest, int sourcesCount, const char \*source, ...);

/\* textblock.hpp \*/

TEXTBLOCK newTEXTBLOCK();

void draw(const TEXTBLOCK &t);

void erase(const TEXTBLOCK &t);

/\* uibscs.hpp \*/

char line(LINETYPE type, LINESHAPE shape);

COLOR getforeground();

COLOR getbackground();

void fillforeground(COLOR foreground, COORD startPosition, int count);

void fillforeground(COLOR foreground, int startPositionX, int startPositionY, int count);

void fillbackground(COLOR background, COORD startPosition, int count);

void fillbackground(COLOR background, int startPositionX, int startPositionY, int count);

void setforeground(COLOR foreground);

void setbackground(COLOR background);

void draw(const VISUAL &v);

SMALL\_RECT measure(int left, int right, int width, int top, int bottom, int height, COORD margin, HORIZONTALALIGNMENT horizontalAlignment, VERTICALALIGNMENT verticalAlignment);

SMALL\_RECT measure(int width, int height, COORD margin, HORIZONTALALIGNMENT horizontalAlignment, VERTICALALIGNMENT verticalAlignment);

// Returned from measure functions when arguments are out of enums ranges

const SMALL\_RECT RECT\_ERROR = { -1, -1, -1, -1 };

// Returned from line function when arguments are out of íenums ranges

const char LINE\_ERROR = 'X';

const COORD COORD\_ORIGIN = { 0, 0 };

/\* Drawing chars \*/

const char DOT\_SMALL = 250;

const char DOT\_BIG = 249;

const char BLOCK\_HALF\_TOP = 223;

const char BLOCK\_HALF\_BOTTOM = 220;

const char BLOCK\_HALF\_LEFT = 221;

const char BLOCK\_HALF\_RIGHT = 222;

const char BLOCK\_HALF\_CENTER = 254;

const char BLOCK\_PERFORATED\_MUCH = 176;

const char BLOCK\_PERFORATED\_NORMAL = 177;

const char BLOCK\_PERFORATED\_FEW = 178;

const char BLOCK\_NORMAL = 219;

/\* Defaults for UI elements \*/

const int DCURSOR\_SIZE = 20;

const COLOR DCOLOR\_FORE\_FOCUSED = White;

const COLOR DCOLOR\_FORE\_UNFOCUSED = Gray;

const COLOR DCOLOR\_FORE\_DISABLED = DarkGray;

const bool DBACKGROUND\_UNDER\_BORDER = true;

const COLOR DCOLOR\_BACK\_FOCUSED = Black;

const COLOR DCOLOR\_BACK\_UNFOCUSED = Black;

const COLOR DCOLOR\_BACK\_DISABLED = Black;

const int DWIDTH = 10;

const int DHEIGHT = 3;

const LINETYPE DBORDER\_TYPE = SingleLine;

#include "advio.hpp"

#include "border.hpp"

#include "button.hpp"

#include "cnslsz.hpp"

#include "cursor.hpp"

#include "operators.hpp"

#include "strext.hpp"

#include "textblock.hpp"

#include "textbox.hpp"

#include "uibscs.hpp"

#endif

## advio.hpp

/\*

Name: advio.hpp (Advanced I/O) v1.0

Copyright: (c) 2016 Marian Dolinský

Author: Marian Dolinský

Date: 29/05/16 23:28

Description: Advanced I/O functions for Windows console.

\*/

#ifndef ADVIO\_HPP

#define ADVIO\_HPP

#include "superior.hpp"

void cls()

{

DWORD attribsWritten;

COORD consoleSize = getconsolesize();

FillConsoleOutputAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), getbackground() + (getbackground() << 4), consoleSize.X \* consoleSize.Y, COORD\_ORIGIN, &attribsWritten);

system("cls");

}

char getkey(char key)

{

if (key == -32 || key == 224)

{

return getch();

}

return key;

}

char getkey()

{

return getkey(getch());

}

char \*createline(char start, char middle, char end, int countOfMiddle)

{

if (countOfMiddle < 0)

{

return NULL;

}

char \*output = (char \*)malloc(countOfMiddle + 3);

if (output == NULL)

{

return NULL;

}

output = ((char \*)memset(output + 1, middle, countOfMiddle)) - 1;

\*output = start;

\*(output + countOfMiddle + 1) = end;

\*(output + countOfMiddle + 2) = 0;

return output;

}

char \*createline(char ch, int count)

{

if (count < 0)

{

return NULL;

}

char \*output = (char \*)malloc(count + 1);

if (output == NULL)

{

return NULL;

}

output = (char \*)memset(output, ch, count);

\*(output + count) = 0;

return output;

}

void drawline(char ch, int count)

{

if (count <= 0)

{

return;

}

while(count-- > 0)

{

putchar(ch);

}

}

COORD hcenter(int left, int right, int width)

{

COORD position = getcursorposition();

position.X = (right - left - width) / 2 + left;

movecursor(position);

return position;

}

COORD hcenter(int left, int right, int width, const char \*format, ...)

{

va\_list ap;

va\_start(ap, format);

COORD position = hcenter(left, right, width);

vprintf(format, ap);

va\_end(ap);

return position;

}

COORD hcenter(int left, int right, const char \*format, ...)

{

va\_list ap;

va\_start(ap, format);

COORD position = vhcenter(left, right, format, ap);

va\_end(ap);

return position;

}

COORD hcenter(int width)

{

return hcenter(0, getconsolesize().X, width);

}

COORD hcenter(int width, const char \*format, ...)

{

va\_list ap;

va\_start(ap, format);

COORD position = hcenter(width);

vprintf(format, ap);

va\_end(ap);

return position;

}

COORD hcenter(const char \*format, ...)

{

va\_list ap;

va\_start(ap, format);

COORD position = vhcenter(0, getconsolesize().X, format, ap);

va\_end(ap);

return position;

}

COORD vcenter(int top, int bottom, int height)

{

COORD position = getcursorposition();

position.Y = (bottom - top - height) / 2 + top;

movecursor(position);

return position;

}

COORD vcenter(int height)

{

return vcenter(0, getconsolesize().Y, height);

}

COORD center(int left, int right, int width, int top, int bottom, int height)

{

COORD position =

{

hcenter(left, right, width).X,

vcenter(top, bottom, height).Y

};

return position;

}

COORD center(int left, int right, int width, int top, int bottom, int height, const char \*format, ...)

{

va\_list ap;

va\_start(ap, format);

COORD position = center(left, right, width, top, bottom, height);

vprintf(format, ap);

va\_end(ap);

return position;

}

COORD center(int left, int right, int top, int bottom, const char \*format, ...)

{

COORD position;

position.Y = vcenter(top, bottom, 1).Y;

va\_list ap;

va\_start(ap, format);

position.X = vhcenter(left, right, format, ap).X;

va\_end(ap);

return position;

}

COORD center(int width, int height)

{

COORD consoleSize = getconsolesize();

return center(0, consoleSize.X, width, 0, consoleSize.Y, height);

}

COORD center(int width, int height, const char \*format, ...)

{

va\_list ap;

va\_start(ap, format);

COORD consoleSize = getconsolesize();

COORD position = center(0, consoleSize.X, width, 0, consoleSize.Y, height);

vprintf(format, ap);

va\_end(ap);

return position;

}

COORD center(const char \*format, ...)

{

COORD position;

position.Y = vcenter(1).Y;

va\_list ap;

va\_start(ap, format);

position.X = vhcenter(0, getconsolesize().X, format, ap).X;

va\_end(ap);

return position;

}

COORD vhcenter(int left, int right, const char \*format, va\_list args)

{

size\_t length = (size\_t)(strlen(format) \* 1.5);

char \*output = (char \*)malloc(length);

if (output == NULL)

{

return COORD\_ORIGIN;

}

while (vsprintf(output, format, args) < 0)

{

length = (size\_t)(length \* 1.5);

if ((output = (char \*)realloc(output, length)) == NULL)

{

return COORD\_ORIGIN;

}

}

COORD position = hcenter(left, right, strlen(output));

if (position != COORD\_ORIGIN)

{

fputs(output, stdout);

}

va\_end(args);

free(output);

return position;

}

#endif

## cnslcz.hpp

/\*

Name: cnslsz.hpp (Console size) v1.0

Copyright: (c) 2016 Marian Dolinský

Author: Marian Dolinský

Date: 29/05/16 12:51

Description: Functions for working with console size in Windows console.

\*/

#ifndef CNSLSZ\_HPP

#define CNSLSZ\_HPP

#include "superior.hpp"

COORD getconsolesize()

{

CONSOLE\_SCREEN\_BUFFER\_INFO bufferInfo;

GetConsoleScreenBufferInfo(GetStdHandle(STD\_OUTPUT\_HANDLE), &bufferInfo);

COORD consoleSize =

{

bufferInfo.srWindow.Right - bufferInfo.srWindow.Left + 1,

bufferInfo.srWindow.Bottom - bufferInfo.srWindow.Top + 1

};

return consoleSize;

}

#endif

## cursor.hpp

/\*

Name: cursor.hpp v1.0

Copyright: (c) 2016 Marian Dolinský

Author: Marian Dolinský

Date: 29/05/16 16:40

Description: Functions for controlling cursor in Windows console.

\*/

#ifndef CURSOR\_HPP

#define CURSOR\_HPP

#include "superior.hpp"

void setcursor(bool isVisible, int size)

{

CONSOLE\_CURSOR\_INFO cursor;

cursor.bVisible = isVisible;

cursor.dwSize = size;

SetConsoleCursorInfo(GetStdHandle(STD\_OUTPUT\_HANDLE), &cursor);

}

void setcursor(bool isVisible)

{

setcursor(isVisible, DCURSOR\_SIZE);

}

COORD getcursorposition()

{

CONSOLE\_SCREEN\_BUFFER\_INFO bufferInfo;

GetConsoleScreenBufferInfo(GetStdHandle(STD\_OUTPUT\_HANDLE), &bufferInfo);

return bufferInfo.dwCursorPosition;

}

void movecursor(COORD position)

{

SetConsoleCursorPosition(GetStdHandle(STD\_OUTPUT\_HANDLE), position);

}

void movecursor(int x, int y)

{

COORD position = { x, y };

movecursor(position);

}

#endif

## operators.hpp

/\*

Name: operators.hpp v1.0

Copyright: (c) 2016 Marian Dolinský

Author: Marian Dolinský

Date: 29/05/16 12:54

Description: Operators used for comparing structures.

\*/

#ifndef OPERATORS\_HPP

#define OPERATORS\_HPP

#include "superior.hpp"

bool operator ==(COORD c1, COORD c2)

{

return c1.X == c2.X && c1.Y == c2.Y;

}

bool operator !=(COORD c1, COORD c2)

{

return !(c1 == c2);

}

bool operator <(COORD c1, COORD c2)

{

return c1.X < c2.X && c1.Y < c2.Y;

}

bool operator >(COORD c1, COORD c2)

{

return c1.X > c2.X && c1.Y > c2.Y;

}

bool operator <=(COORD c1, COORD c2)

{

return c1.X <= c2.X && c1.Y <= c2.Y;

}

bool operator >=(COORD c1, COORD c2)

{

return c1.X >= c2.X && c1.Y >= c2.Y;

}

bool operator ==(SMALL\_RECT r1, SMALL\_RECT r2)

{

return r1.Left == r2.Left && r1.Top == r2.Top && r1.Right == r2.Right && r1.Bottom == r2.Bottom;

}

bool operator !=(SMALL\_RECT r1, SMALL\_RECT r2)

{

return !(r1 == r2);

}

#endif

## strext.hpp

/\*

Name: strext.hpp (String extensions) v1.0

Copyright: (c) 2016 Marian Dolinský

Author: Marian Dolinský

Date: 29/05/16 12:54

Description: New and overload functions for working with strings in Windows console.

\*/

#ifndef STREXT\_HPP

#define STREXT\_HPP

#include "superior.hpp"

char \*stradd(char \*str, char ch, int index)

{

if (index >= 0 && index <= strlen(str))

{

str = (char \*)memmove(str + index + 1, str + index, strlen(str) - index + 1);

\*(str + index) = ch;

}

return str;

}

char \*strrem(char \*str, int index)

{

if (index >= 0 && index < strlen(str))

{

// don't add -1 to copy even terminating char

str = (char \*)memmove(str + index, str + index + 1, strlen(str) - index);

}

return str;

}

char \*strndup(const char \*source, int count)

{

if (count < 0)

{

count = 0;

}

else if (count >= strlen(source))

{

return (char \*)source;

}

char \*output = (char \*)malloc(count + 1);

if (output == NULL)

{

return NULL;

}

strncpy(output, source, count);

\*(output + count) = 0;

return output;

}

char \*strcat(char \*dest, int sourcesCount, const char \*source, ...)

{

strcat(dest, source);

if (sourcesCount <= 1)

{

return dest;

}

va\_list ap;

va\_start(ap, source);

while (sourcesCount-- > 0)

{

strcat(dest, va\_arg(ap, char \*));

}

va\_end(ap);

return dest;

}

#endif

## uibscs.hpp

/\*

Name: uibscs.hpp (UI basics) v1.0

Copyright: (c) 2016 Marian Dolinský

Author: Marian Dolinský

Date: 29/05/16 16:43

Description: Basic function, enumerations and structures for creating UI in Windows console.

\*/

#ifndef UIBSCS\_HPP

#define UIBSCS\_HPP

#include "superior.hpp"

char line(LINETYPE type, LINESHAPE shape)

{

switch (type)

{

case SingleLine:

switch (shape)

{

case Horizontal: return 196;

case Vertical: return 179;

case Edge\_Top\_Left: return 218;

case Edge\_Top\_Right: return 191;

case Edge\_Bottom\_Left: return 192;

case Edge\_Bottom\_Right: return 217;

case T\_Horizontal\_Up: return 193;

case T\_Horizontal\_Down: return 194;

case T\_Vertical\_Left: return 180;

case T\_Vertical\_Right: return 195;

case Cross: return 197;

default: return LINE\_ERROR;

}

case DoubleLine:

switch (shape)

{

case Horizontal: return 205;

case Vertical: return 186;

case Edge\_Top\_Left: return 201;

case Edge\_Top\_Right: return 187;

case Edge\_Bottom\_Left: return 200;

case Edge\_Bottom\_Right: return 188;

case T\_Horizontal\_Up: return 202;

case T\_Horizontal\_Down: return 203;

case T\_Vertical\_Left: return 185;

case T\_Vertical\_Right: return 204;

case Cross: return 206;

default: return LINE\_ERROR;

}

case Block: return BLOCK\_NORMAL;

case None: return ' ';

default: return LINE\_ERROR;

}

}

// These two variables shouldn't be visible from another files

// But in Dev-C++ 'static' it most likely doesn't work as it is supposed to

// Maybe because of this files are compiled by C++ compiler instead of C conpiler

static COLOR foregroundColor = White;

static COLOR backgroundColor = Black;

COLOR getforeground()

{

return foregroundColor;

}

COLOR getbackground()

{

return backgroundColor;

}

void fillforeground(COLOR foreground, COORD startPosition, int count)

{

foregroundColor = foreground;

DWORD written;

FillConsoleOutputAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), foregroundColor + (backgroundColor << 4), count, startPosition, &written);

}

void fillforeground(COLOR foreground, int startPositionX, int startPositionY, int count)

{

COORD startPosition = { startPositionX, startPositionY };

fillforeground(foreground, startPosition, count);

}

void fillbackground(COLOR background, COORD startPosition, int count)

{

backgroundColor = background;

DWORD written;

FillConsoleOutputAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), foregroundColor + (backgroundColor << 4), count, startPosition, &written);

}

void fillbackground(COLOR background, int startPositionX, int startPositionY, int count)

{

COORD startPosition = { startPositionX, startPositionY };

fillbackground(background, startPosition, count);

}

void setforeground(COLOR foreground)

{

foregroundColor = foreground;

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), foregroundColor + (backgroundColor << 4));

}

void setbackground(COLOR background)

{

backgroundColor = background;

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), foregroundColor + (backgroundColor << 4));

}

void draw(const VISUAL &v)

{

COLOR foregroundBackup = getforeground();

setforeground(v.Color);

putchar(v.Char);

setforeground(foregroundBackup);

}

SMALL\_RECT measure(int left, int right, int width, int top, int bottom, int height, COORD margin, HORIZONTALALIGNMENT horizontalAlignment, VERTICALALIGNMENT verticalAlignment)

{

SMALL\_RECT output;

COORD consoleSize = getconsolesize();

switch (horizontalAlignment)

{

case HLeft: output.Left = left + margin.X; break;

case HCenter: output.Left = ((right - left - width) / 2) + left + margin.X; break;

case HRight: output.Left = right - width + margin.X; break;

default: return RECT\_ERROR;

}

switch (verticalAlignment)

{

case VTop: output.Top = top + margin.Y; break;

case VCenter: output.Top = ((bottom - top - height) / 2) + top + margin.Y; break;

case VBottom: output.Top = bottom - height + margin.Y; break;

default: return RECT\_ERROR;

}

output.Right = output.Left + (width - 1);

output.Bottom = output.Top + (height - 1);

return output;

}

SMALL\_RECT measure(int width, int height, COORD margin, HORIZONTALALIGNMENT horizontalAlignment, VERTICALALIGNMENT verticalAlignment)

{

COORD consoleSize = getconsolesize();

return measure(0, consoleSize.X, width, 0, consoleSize.Y, height, margin, horizontalAlignment, verticalAlignment);

}

#endif

# Vývojový diagram

STOP

mainmenu(Green)

cls()

Hcenter("Press any key to close the game . . .")

vcenter(3)

initialize("Snake", "v1.7 'A better way'", "beta 4", 79, 24)

START

hcenter("SOMETHING WENT WRONG :-(\n")

getch()

exit(0)

# 

vcenter(int height)

return position

SetConsoleCursorPosition(GetStdHandle(STD\_OUTPUT\_HANDLE), position)

movecursor(COORD position)

return position

COORD position = getcursorposition();

position.Y = (bottom - top - height) / 2 + top;

movecursor(position);

vcenter(int top, int bottom, int height)

return vcenter(0, getconsolesize().Y, height)

# Závěr

Jsem velmi rád, že jsem mohl na této hře pracovat, i když ji nakonec odevzdávám ne v tak „dokonalém“ stavu jak bych si ji představoval. Tuto hru jsem sice začal vytvářet už v září roku 2015 a až do půlky prosince jsem na ní velmi intenzivně pracoval, avšak poté mě již takovéto nadšení přešlo, a i přesto, že za posledních pět měsíců jsem se naučil velkou spoustu nových věcí, které jsem chtěl do této hry použít, ne všechny jsem do ní z časových důvodů nakonec zakomponoval, popřípadě nejsou úplně dokončené.

I přes nedostatek času jsem však od ledna stihl přepsat prakticky všechen kód. Původně totiž hra používala můj vlastní vykreslovací engine, který však byl velmi pomalý, protože bylo nutno vše vykreslovat od levého horního rohu konzole. V lednu jsem tedy začal pracovat na verzi 1.7, která přinesla do hry nový dech a právě díky novému, rychlejšímu způsobu vykreslování jsem měl možnost přidat spoustu nových věcí, i když hra nakonec zdaleka neobsahuje všechny, které jsem měl v plánu. Bohužel, časové nedostatky také způsobily, že mi nezbylo dost času na optimalizace a je tedy možné, že některé části programu by mohly fungovat lépe. I přesto si myslím, že vývoj této hry mi hodně pomohl v celkovém pochopení jazyka C a obecně mě posunul v programování o velký kus dále. Právě při jejím vývoji jsem se totiž naučil spoustu nových věcí, jako jsou například struktury, pointery, bitový posun, vlastní datové typy, enumerace, funkce s proměnlivým počtem argumentů a další, které mi také pomohly v pochopení přesného fungování základních funkcí jazyka C, jako jsou printf, scanf a podobné. Spoustu z těchto znalostí jsem měl také možnost naučit i své spolužáky, kteří některé tyto znalosti později mohli využít ve svých ročníkových pracích.