A PROGRAMOZÁS ALAPJAI 2.

HÁZI FELADAT DOKUMENTÁCIÓ

Hack And Slash

KÉSZÍTETTE: CSEPPENTŐ VIKTOR, F2MJMI
csviktor97@gmail.com

KÉSZÍTÉS FÉLÉVE: 2016/17/2

TARTALOMJEGYZÉK

Felhasználói dokumentáció

A program egy felülnézetes, szórakoztató videójáték, ahol a karakterünkkel minél több ellenséget le kell győzni mielőtt ők győznek le minket. A program indításakor egy főmenüben találjuk magunkat ahol az egér segítségével tudunk navigálni a menüpontok között. A legfölső menüpont elindít egy új játékot, a középső betölt egy már régebben elmentett játékot, az utolsó pedig kilép teljesen a programból.

Miután elindítunk egy új játékot vagy betöltünk egy régit, egy véletlenszerűen generált pályán találjuk a karakterünk textúráját és néhány ellenséget felülnézetben. A billentyűzet nyilaival tudjuk elindítani a karakterünket, a space gomb nyomására pedig megtudjuk ütni az ellenségeket. Azonban vigyázzunk, mert visszaütnek! Az ellenségek folyamatosan mozognak, próbálnak elkapni minket. Szerencsére mi gyorsabbak és erősebbek előre, de együttes erővel el tudnak bánni velünk, mivel hullámokban érkeznek és egyre többen. Vigyázni kell, hogy a pályán falak is vannak, nem tudunk bárhova menni közvetlenül, így célszerű előre eltervezni az utunkat.

Ha szeretnénk egy kicsit szüneteltetni a játékot vagy elmenteni a jelenlegi állását, akkor a billentyűzet escape gombjával tudunk kilépni egy menübe, ami leszüneteli a játékot. A menüben az egérrel visszatudunk lépni a játékba, el tudjuk azt menteni és ki tudunk lépni a főmenübe. A játék addig tart, amíg az ellenségek le nem győznek minket. A játék célja minél több pont gyűjtése az ellenségek kivégzésével. Ha bármikor ki akarunk lépni rögtön az alkalmazásból, akkor az ablak jobb felső sarkában lévő X-szel ezt meg tudjuk tenni.

Osztályok statikus leírása

Block

Felelőssége

A véletlenszerűen generált pálya 25 darab építőkőből (blockból) fog állni. Egy darab block az alábbiak közül az egyiket reprezentálja valahogyan elforgatva. A piros szín az utat jelzi, a fekete pedig, hogy ott fal van. Látható, hogy a sarkukban mindegyiknek fal van és úgy tudjuk őket egyértelműen meghatározni, hogy vagy nyitottak egy adott irányból vagy nem.

Például a legelső példa fölülről és jobbról nyitott, balról és lentről pedig zárt.

Attribútumok

Privát

```
int up, right, down, left; //-1: meg nem határozott, 0: fal, 1: út; elmondja hogy a block oldalain milyen textúra van
Block *leftNeighbor, *rightNeighbor, *upNeighbor, *downNeighbor; //szomszédblockokra mutató pointerek
int* coords; //2 db 0 és 5 közötti szám; a térképen a koordináták
```

Metódusok

Publikus

```
Block();
~Block();
int* getCoords() const; //Getterek és setterek következnek
void setCoords(int x, int y);
Block* getLeftNeighbor() const;
Block* getDownNeighbor() const;
Block* getUpNeighbor() const;
Block* getRightNeighbor() const;
void setLeftNeighbor(Block& b);
void setDownNeighbor(Block& b);
void setUpNeighbor(Block& b);
void setRightNeighbor(Block& b);
int getUp() const;
int getRight() const;
int getDown() const;
int getLeft() const;
void setUp(int path);
void setRight(int path);
void setLeft(int path);
void setDown(int path);
void setBounds(int up, int right, int down, int left); //Paraméterként megkapott értékek alapján beállítja a block oldalait
```

Map

Felelőssége

Ez maga a pályának az osztálya. Az osztály tagváltozói egy 25x25-ös inteket tartalmazó kétdimenziós tömb, ami meghatározza, hogy egy adott koordinátán út vagy fal van. Emellett az út és a fal textúráját és a grafikus osztályra mutató pointert tartalamazza.

Attribútumok

Privát

```
int **blueprint; //road: 1, wall: 0; 25x25-ös kétdimenziós tömb
Texture* road; //Az út textúrája
Texture* wall; //Fal textúrája
Graphic *g; //Grafikus osztályra pointer
```

Metódusok

Publikus

```
Map(Graphic* graphic = NULL, int** blueprintParam = NULL, bool create = true); //Létrehoz
Map(Map& other);
~Map();
void createMap(); //Főfüggvény, ami a többi függvény segítségével létrehoz egy térképet
void setBlockNeighbours(Block**& blocks); //A Blockok szomszédváltozóit beállítja
void randomBorders(Block**& blocks); //A Blockoknak, aminek tudja megadja a többi Blocktól függően az oldalait, aminek nem tudja annak random adja
bool BFS(Block**& blocks); //BFS-sel bejárjuk a kész térképet, hogy összefüggő-e
bool unique(int actI, int actJ, int arr[25][2]); //Megnézi egy 25 méretű tömbben hogy egy koordináta-pár valóban csak egyszer van benne
void createBlueprint(Block**& blocks); //Az 5x5-ös blocktömbből 25x25-ös int tömböt hoz létre
void printBlueprint() const; //Kiírja a konzolra 0-kal és 1-esekkel a térképet
void printBlockprint(Block**& blocks) const; //Itt a blockokat írja ki rendezetten
Texture* getRoadTexture() const;
Texture* getWallTexture() const;
Graphic * getGraphic() const;
int **getBlueprint() const;
void drawMap() const; //A Graphic *g tagváltozóval kirajzolja az SDL ablakba a térképet szépen
friend std::ostream& operator<<(std::ostream& os, const Map& m);
friend std::istream& operator >> (std::istream& is, Map& m); //Ezek az operátorokra a fájlba írásakor és fájlból olvasásakor lesz szükséges, ha elmentjük a játékot
```

Graphic

Felelőssége

Ez az osztály főleg az SDL inicializásával és a grafikus felülettel törődik. A program indulása után rögtön ezt hozzuk létre és minden másik osztály mely szeretne a képernyőn látszani(pl. Map, Character), erről kap egy pointert, hogy ennek a segítségével rajzolódjanak ki.

Attribútumok

Privát

```
const int screenWidth; //Ablak paraméterei
const int screenHeight;
const std::string title; //Ablak címe
SDL_Window *window; //Maga az elemi SDL ablak
SDL_Surface *windowSurface; //Segédváltozó lesz a textúrák alkotásánál
SDL_Renderer *renderer; //Ezzel égetjük a textúrákat az ablakra
bool SDLRunning; //Igaz/hamis változó, hogy fut-e a grafikus felület
```

Metódusok

Publikus

```
Graphic(int W = 800, int H = 600, std::string T = "Game");
~Graphic();
const int getScreenWidth() const;
const int getScreenHeight() const;
const std::string getTitle() const;
SDL_Window* getWindow();
SDL_Surface* getWindowSurface();
SDL_Renderer* getRenderer();
bool isRunning();
void renderTexture(Texture& texture, double x, double y, double angle = 0); //Egy textúrát ráéget az ablakra a megfelelőt helyre, szögben
void renderText(SDL_Texture* textTexture, double x, double y, int width, int height); //Egy szövegtextúrák éget az ablakra
```

Texture

Felelőssége

Saját textúra osztályom, aminek van egy SDL_Texture* változója és méretei. Minden kirajzolódó objektumnak van legalább egy Texture* változója, és ilyen osztályt kap például a Graphic osztály renderTexture függvénye.

Attribútumok

Privát

```
int width; //Méretek
int height;
SDL_Texture *texture; //SDL elemi textúrája
Graphic *g; //A grafikus objektumra pointer
int mapping; //0 = no mapping, 1 = black mapping, 2 = white mapping; ez arra való, hogy a négyzet alakú textúrának csak a fontos részei látszódjanak a képernyőn
```

Metódusok

Publikus

```
Texture(Graphic* g = NULL, std::string path = NULL, int width = 24, int height = 24, int mapping= 0);
~Texture();
void loadFromFile(std::string path); //Fájlból történő olvasás (.bmp fájl)
void freeTexture(); //Területek felszabadítása
int getWidth() const;
int getHeight() const;
void setWidth(int w);
void setHeight(int h);
SDL_Texture* getTexture() const;
```

Font

Felelőssége

Betűtípus (.ttf fájl) betöltése, pillanatnyi szöveg betöltése és kirajzolása az ablakra. Pl. GAME OVER szöveg.

Attribútumok

Privát

```
int size; //Betűk mérete
TTF_Font *font; //SDL TTF moduljának a font változója
SDL_Texture *textTexture; //Mindig tartalmaz egy szöveget, amit majd kiír/kirajzol
int width;
int height;
int posX;
int posY; //Jelenlegi szövegnek méretei, koordinátái
Graphic *g; //Grafikus objektumra mutató pointer
SDL_Color color; //Szöveg színe
```

Metódusok

Publikus

```
Font(Graphic *g = NULL, int size = 24); //.ttf fájl betöltése
~Font();
void loadText(std::string text); //Egy szöveget betölt textúraként
void drawText(std::string text, int posXParam, int posYParam, SDL_Color newColor); //Szöveg kirajzolása
void freeFont(); //Betűtípus felszabadítása
void setFontSize(int sizeParam); //Betűméret beállítása
```

Menu

Felelőssége

Gyakorlatilag ilyen objetumunk nem lesz, csak öröklődünk belőle két darab menüfajtát.

Attribútumok

Védett

```
Texture *menuTexture; //Menük textúrája
Graphic *g; //Grafikus objektumra mutató pointer
```

Metódusok

Publikus

```
Menu(Graphic* graphic = NULL); //Semmi extra
~Menu();
virtual void chooseMenu() = 0; //Az SDL eseményeket kezeli, menüpontok szabályozása
Graphic *getGraphic() const;
Texture* getMenuTexture() const;
```

MainMenu

Felelőssége

Ez maga a főmenü, ahol ki tudjuk választani, hogy új játékot kezdünk-e, betöltünk egy régit, vagy kilépünk a játékból.

Ősosztályok

Menu: Mivel ez is egy menüfajta, és kibővíti az Menu osztályt.

Attribútumok

Privát

```
bool quit; //Segédváltozó, kilépjünk-e a játékból
```

Metódusok

Publikus

```
MainMenu(Graphic* graphic= NULL);
~MainMenu();
void chooseMenu();
void newGame(); //Új játék kezdése
void loadGame(); //Mentett játék betöltése
```

MinuMenu

Felelőssége

Ha játékot egy kicsit szüneteltetni akarjuk, vagy kilépni, vagy elmenteni akko az Escape gombbal elő tudunk hívni egy menüt, ami kiszolgál minket.

Ősosztályok

Menu: Mivel ez is egy menüfajta, és kibővíti az Menu osztályt.

Metódusok.

Publikus

```
MiniMenu(Graphic* graphic = NULL); //Meghívja ősosztályának a konstruktorát
~MiniMenu();
void chooseMenu(); //Menüpontok kiválasztásához függvény
```

Character

Felelőssége

A játékos és minden ellenség egy karakter objektum, mely meghatározza általános változóit pl. sebézt, sebességet, életerőt. Hozzátartozik még egy enum class is, ami az irányokat foglalja magába. Védett változói vannak, mivel ez egy ősosztály lesz.

```
enum class Direction { UP, RIGHT, DOWN, LEFT, NONE };
```

Attribútumok

Védett

```
Graphic *g; //SDL-t összefoglaló obejktumra egy pointer
bool alive; //Életben van-e a karakter
int health; //Élete
const int maxHealth; //Maximum élete
Direction dir; //Aktuális iránya
double speed; //Sebessége
int damage; //Sebzése
Texture* currentTexture; //Jelenlegi textúra (a játékos osztályban több textúra is lesz)
double x;
double y; //x és y koordináták a térképen
int width;
int height; //Textúra szélessége, magassága
```

Metódusok

Publikus

```
Character(Graphic* graphic = NULL, double posX = 24.0, double posY = 24.0); //Inicializálja a változókat
virtual ~Character();
bool isAlive() const; //Visszaadja, hogy él-e még a karakter
double getX() const;
double getY() const;
double getSpeed() const;
int getWidth() const;
int getHeight() const;
Direction getDir() const;
void setDir(Direction newDirection);
Texture* getCurrentTexture() const;
int getHealth() const;
void beingDamaged(int damageDealt); //Sebzést kap, ha egy másik karakter megütötte
const int getMaxHealth() const;
void attack(Character& other) const; //Megtámad egy másik karaktert
void move(Direction newDirection); //Mozgatja a karaktert a megadott irányba
void drawCharacter() const; //Kirajzolja a karaktert a képernyőre
```

Player

Felelőssége

A játékos ezt az objektumot irányítja, csak egy van belőle.

Ősosztályok

Character: mivel a játékos egy karakter és megkőnnyíti a kódolást, hogy nem kell mindent újra leírni.

Attribútumok

Privát

```
Texture* standingTexture; //Akkor jelenik meg ez a textúra - ha a játékos egy helyben áll
Texture* movingTexture; //Ha mozog
Texture* attackingTexture; //Ha támad
int attackingTimer; //Számolja az időt, hogy mikor támadt utoljára és egy pici ideig nem üthet mégegyszer
int movingTimer; //Hasonló; ezek a timerek azért szükségesek, hogy a textúrák szépen legyenek betöltve
```

Metódusok

Publikus

```
Player(Graphic *graphic = NULL, double posX = 48.0, double posY = 48.0);
~Player();
Texture* getStandingTexture() const;
Texture* getMovingTexture() const;
Texture* getAttackingTexture() const;
void alternateMoving(); //A standing és movingTexture-t váltja egymás után
int getAttackingTimer() const;
int getMovingTimer() const;
void resetAttackingTimer(); //Reseteli az időzítőket
void resetMovingtimer();
void setToStandingTexture(); //CurrentTexturet beállítják a megfelelő értékekre
void setToAttackingTexture();
void setToMovingTexture();
friend std::ostream& operator<<(std::ostream& os, const Player& p);
friend std::istream& operator>>(std::istream& is, Player& p); //Kiíró, beolvasó operátorok
```

Enemy

Felelőssége

Az ellenségek osztálya, ebből több lesz egyszerre a játékban. Annyival különbözik a Character osztálytól, hogy van egy int index tagváltozója, amelyik azonosítja az ellenségeket egyesével. Lassabb és kevesebbet sebez mint a Player.

Ősosztályok

Character: mivel, a játékossal „testvérosztály”, csak az Enemy-nek kevesebb textúrája van és van indexe.

Attribútumok

Privát

```
int index; //Tömbben lesznek az ellenségek és egy indexszel különböztetjük meg őket
Texture *enemyTexture; //Ellenség textúrája (mindig ez lesz a currentTexture)
```

Metódusok

Publikus

```
Enemy(int indexParam, Graphic* graphic = NULL, double posX = 0, double posY = 0);
~Enemy();
int getIndex() const;
void setIndex(int indexParam); //Beállítja az indexet
int getDamage() const;
Texture* getEnemyTexture() const;
friend std::ostream& operator<<(std::ostream& os, const Enemy& e);
friend std::istream& operator >> (std::istream& is, Enemy& e); //Kiír, beolvas
```

Game

Felelőssége

Ez a legfőbb osztály, ő egyesíti a pályát, a játékost, az ellenségeket és sok mást. Ő irányítja a játék lefolyását.

Attribútumok

Privát

```
Graphic *graphic; //Grafikus objektum
Player *player; //Játékos
Enemy **enemies; //Ellenségek tömbje
MiniMenu *miniMenu; //Escape-pel elérhető menü
Map *map; //Pálya
Font* font; //Betűtípus, szövegek
int enemyNumber; //Ellenségek száma
int score; //Pontszám
int waveCounter; //Ellenségek hullámainak száma
unsigned waveTimer; //Ez határozza meg, hogy mikor jön a következő hullám ellenség
```

Metódusok

Publikus

```
Game(Graphic* graphicparam = NULL, Player *playerparam = NULL, Enemy **enemiesparam = NULL, int enemyNumberparam = 0,
    Map* mapparam = NULL, int scoreparam = 0, int waveCounterparam = 1, unsigned waveTimerParam = 0);
~Game();
Graphic* getGraphic() const;
Player* getPlayer() const;
Enemy** getEnemies() const;
MiniMenu* getMiniMenu() const;
Map* getMap() const;
int getEnemyNumber() const;
int getWaveCounter() const;
unsigned getWaveTimer() const;
void start(); //Elindítja a játékot
Direction keyboardEventHandling(SDL_Event& ev, int pressingMoment[5],
    bool& alreadyAttacked); //A leütött billentyűket érzékeli és támad vagy mozog a karakter
void enemyMotion() const; //Ellenségeket mozgatja
void enemiesAttack(unsigned& enemyAttackTimer) const; //Az ellenségek ekkor ütnek
void drawEverything() const; //Mindent, azaz mindent kirajzol
void drawTexts() const; //Csak a szövegeket rajzolja ki
void spawnWaves(); //Előhívja az ellenséghullámokat
void addEnemy(); //Egy ellenséget hozzáad
void removeEnemy(int enemyIndex = -1); //Ellenséget töröl, ha nincs index akkor a legutolsót törli
bool legitPosition(Character*const& ch, Direction d, bool isPlayer) const;
//Egy karakterpointer-referenciát, irányt kap és megkapja, hogy játékosra vagy ellenségre nézzük-e meg, hogy ütközne-e valamivel
bool collideWithPlayer(int posX, int posY, int width = 24, int height = 24) const;
//Megnézi, hogy egy pozíció ütközik-e a játékossal
bool collideWithWall(int posX, int posY, int width = 24, int height = 24) const; //Fallal
int collideWithEnemies(int posX, int posY, int width = 24, int height = 24) const; //Ellenséggel
bool collideWithSpecificEnemy(int enemyIndex, int posX, int posY, int width = 24, int height = 24) const;
//Egy specifikus ellenséggel
int enemyIndexAtPosition(int posX, int posY, int width = 24, int height = 24) const;
//Egy adott pozíción lévő ellenség indexét adja vissza, ha nincs ott ellenség -2-t ad vissza
void attackFrontEnemy(); //Megtámadjuk az előttünk lévő ellenséget
int getFrontEnemy() const; //Visszaadja az előttünk lévő ellenség indexét
void gameOver() const; //Játékot lezáró függvény
void saveGame() const; //Játékot elmentő függvény
```

SDL_error

Felelőssége

Ez egy exception osztály, öröklődik az std::exception osztályból. Ezt fogjuk elkapni, ha valamilyen SDL-lel kapcsolatos kivételt kezelünk.

Ősosztályok
std::exception: mivel ennek is ugyanolyan változói, függvényei vannak, és ez is egy exceptionosztály.

Attribútumok

Privát

```
std::string explanatory; //Elmagyarázza a hiba okát
```

Metódusok.

Publikus

```
SDL_error(std::string explanatory = "SDL_ERROR");
virtual void what(); //Visszaadja a stringet + az SDL_Geterror() függvényt
~SDL_error();
```

TTF_error

Felelőssége
Szinte ugyanaz mint az SDL_error osztály csak ez az SDL TTF moduljának a hibáival foglalkozik

Ősosztályok
std::exception: ugyanamiatt, miért az SDL_error

Attribútumok

Privát

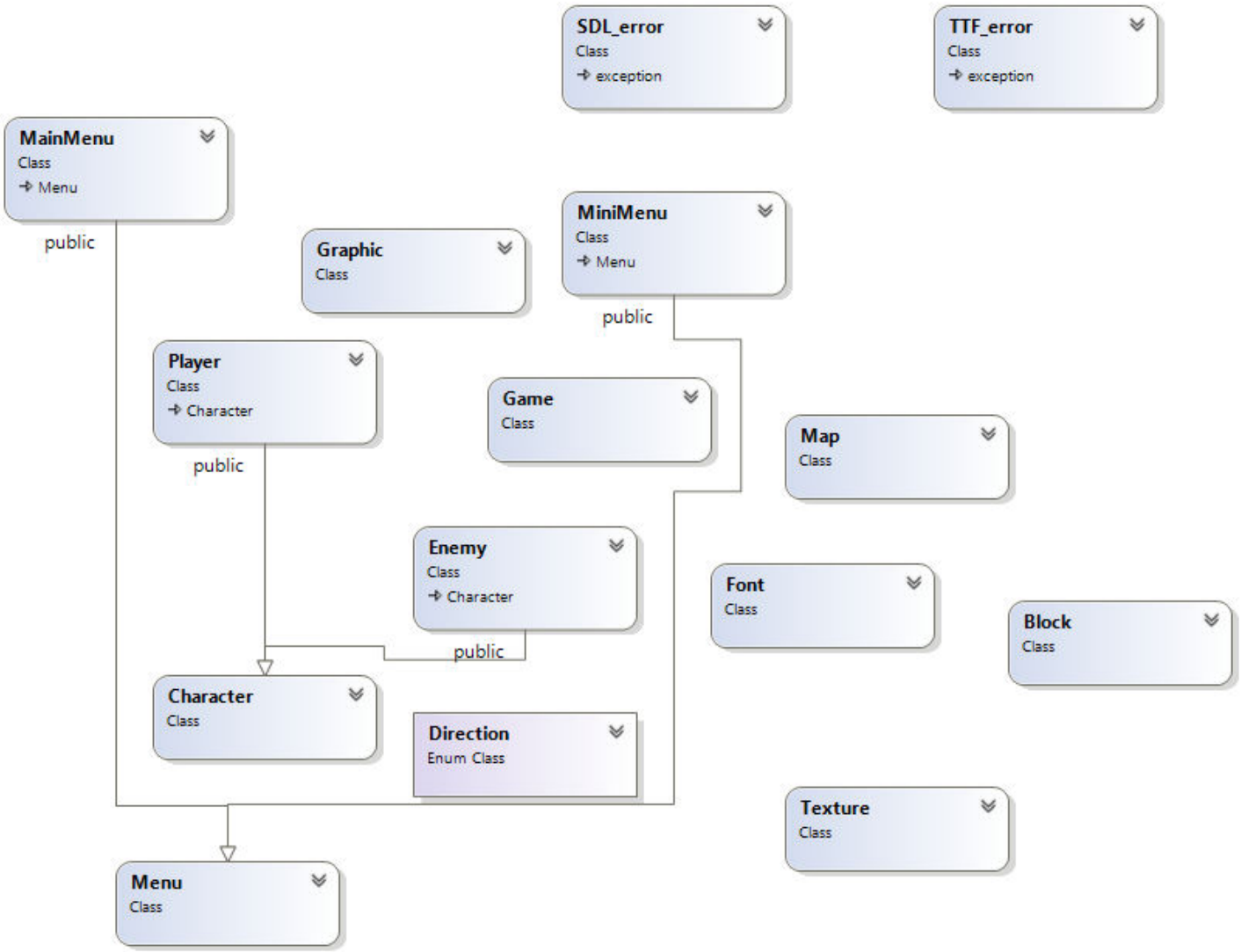
```
std::string explanatory; //Magyarázóstring
```

Metódusok

Publikus

```
TTF_error(std::string explanatory = "TTF_ERROR");
virtual void what(); //Elmagyarázza a hiba okát
~TTF_error();
```

UML osztálydiagramm



Összegzés

Mit sikerült és mit nem sikerült megvalósítani a specifikációból?

Mindent sikerült megvalósítani, sőt adtam hozzá még nem specifikált elemeket is. Ilyen például a játék elmentése és betöltése, az elvártnál szebb grafikus interfész, ellenségek intelligens irányítása, pontozásrendszer, SDL_TTF modul használata.

Mit tanultál a megvalósítás során?

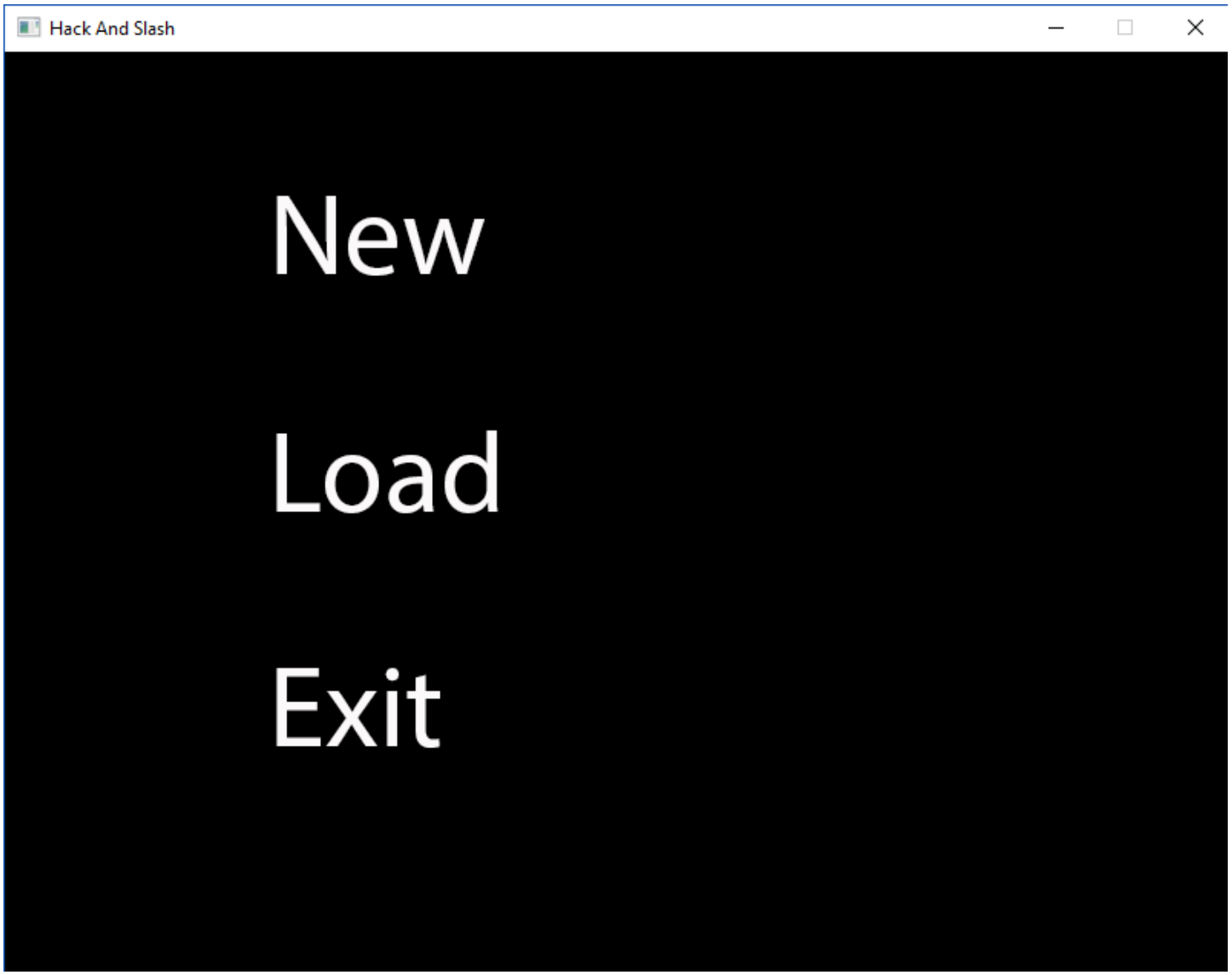
Nagyon sok mindent, főleg az SDL használatát és azt, hogy C++-ban soha többet nem fogok SDL-t használni, átállok az objektum orientált SFML-re. Betekintést nyertem a játékfejlesztésbe, el kezdett jobban érdekelni a szoftverfejlesztési irány.

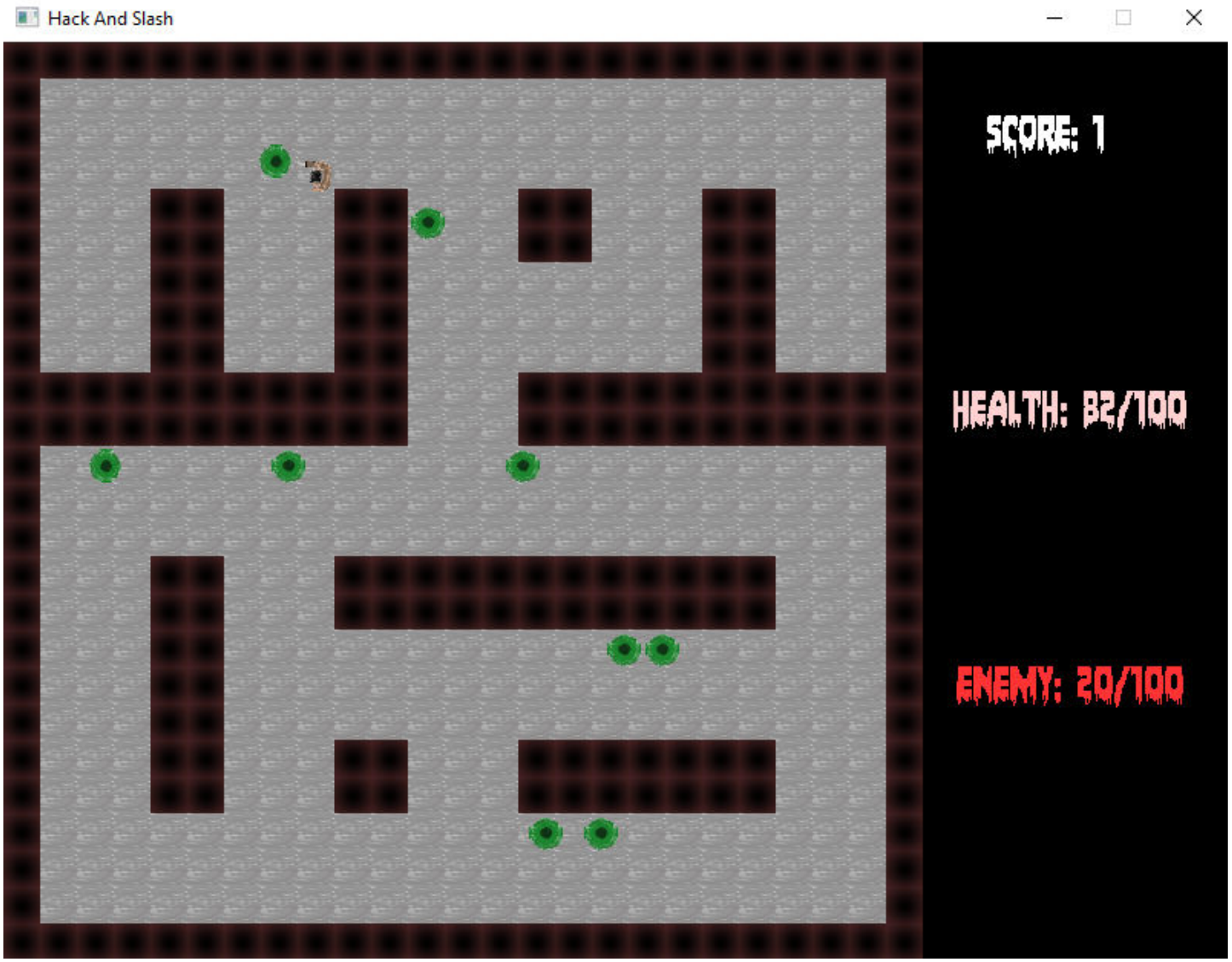
Továbbfejlesztési lehetőségek

Nagyon szívesen tennék még bele szerepjáték-elemeket, többféle fegyvert, okosabb mesterséges intelligenciát, jobb kivételkezelést és talán egy Highscores menüpont is elférne. Szeretnék nyáron egy hasonló, de teljesebb programot írni.

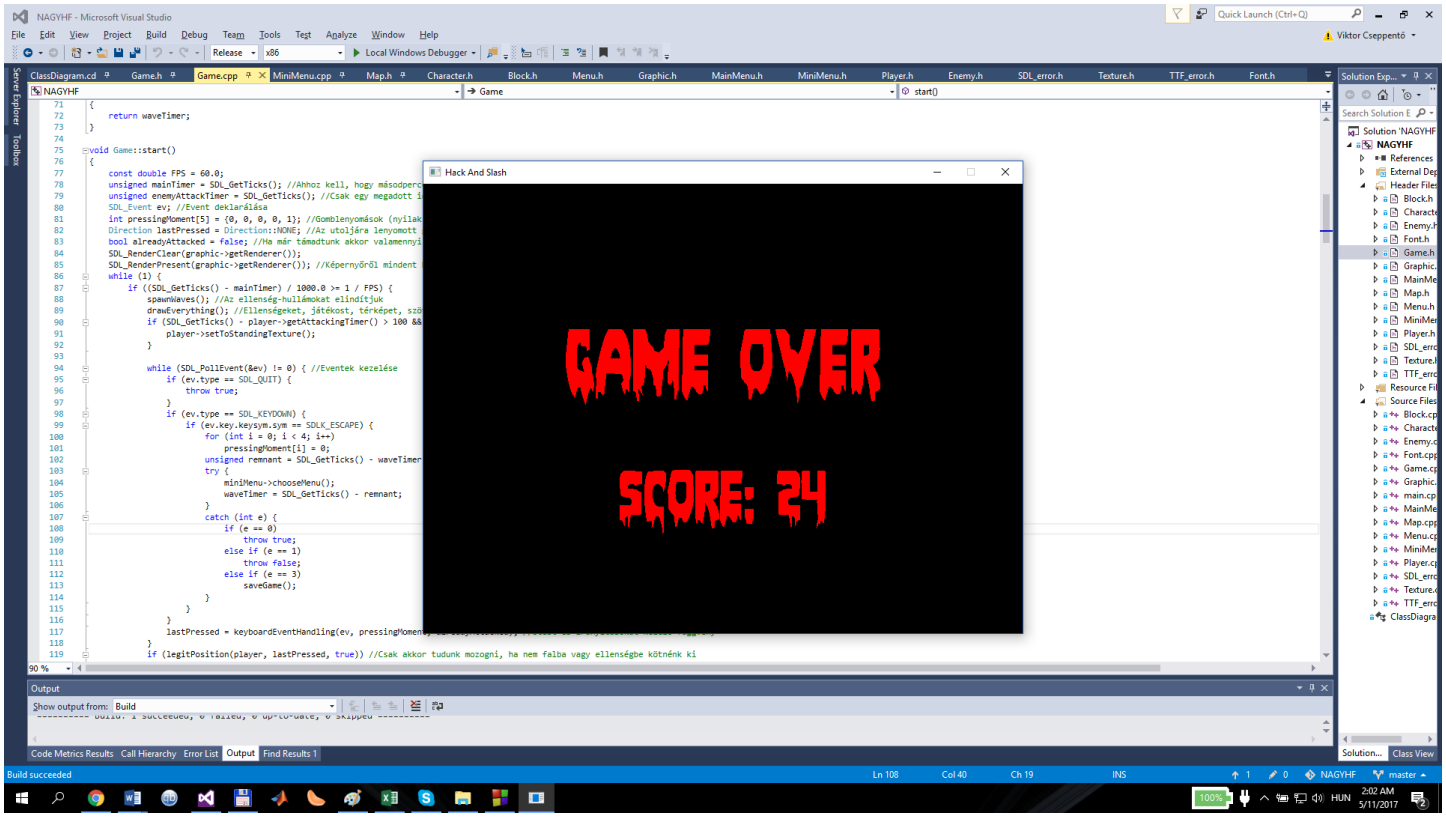
Úgy gondolom, hogy a játékot feldobná egy háttérzene, és egy szebb grafikus felületet is megérdemelne, de sajnos ahhoz nem nagyon értek.

Képernyőképek a futó alkalmazásról




Hack And Slash
SCORE: 1
HEALTH: 82/100
ENEMY: 20/100


Hack And Slash
SCORE: 25
HEALTH: 2/100
ENEMY: 40/100


NAGYHF - Microsoft Visual Studio
Hack And Slash
GAME OVER
SCORE: 24