AKNAKERESŐ – FEJLESZTŐI DOKUMENTÁCIO

[Készitette: Balogh Tamás (Neptun: K0JJ6X)]

1. BEVEZETŐ

Ez a dokumentáció az Aknakereső projekt technikai áttekintését, fejlesztői környezetét, telepítését, és az API (Application Programming Interface) részletes leírását tartalmazza. A dokumentáció célja, hogy bemutassa a projekt belső felépítését és struktúráját, hogy akár az újonnan jött fejlesztők egy áttekinthetőbb képet kapjanak a programról. A forráskód is tartalmaz megjegyzéseket a program megvalósított elemeiről, tehát a fejlesztők az IntelliSense segítségével például fejlesztés kőzben is kapnak leírást.

2. FEJLESZTŐI KÖRNYEZET

A projekt a C programozási nyelven lett fejlesztve a C99-es standard szabályai alapján. A fejlesztés Linux alapú operációs rendszeren történt a GCC fordítóprogram segítségével. A program build folyamatát a CMake segítségével automatizáljuk, ezáltal mind Windowson, mind Linux alapú operációs rendszereken is elfut a program. A forráskód a könnyebb és gördülékenyebb fejlesztés érdekében modulakra van bontva. A lib mappában található meg a deklarációs fájlok, ahol a struktúrák definícói vannak megadva, továbbá egyes függvényeké is. A src mappában található található az implementációs fájlok, ahol a lib mappában található függvények viselkedései vannak megvalósítva. A program az SDL3, SDL3_image, és az SDL3_ttf külsős könyvtárakkal valósítja meg a grafikus megjelenítést. A projekt verziókövetéséhez Githubot használunk.

3. TELEPÍTÉS

A projekt telepítése előtt győződjünk meg arról, hogy a rendszerre fel vannak telepítve az alábbi csomagok (SDL3, SDL3_image, SDL3_ttf), ha nincs, ajánljuk a vcpkg csomagtelepítő rendszer használatát. Windowsos rendszereken kifejezetten ajánljuk a Visual Studio 2022-es fejlesztői környezetet feltelepíteni a C++ fejlesztői csomaggal a könnyebb és gördülékenyebb fejlesztés érdekénben

MinGW telepítési linkje: https://sourceforge.net/projects/mingw/

VCPKG telepítési linkje:
https://learn.microsoft.com/en-us/vcpkg/get_started/get-started?pivots=shell-powershell

A Visual Studio 2022 telepítése:
https://visualstudio.microsoft.com/

CMake telepítési linkje: https://cmake.org/

Telepítés Linuxon:
- Telepítsük a csomagokat a csomagkezelő segítségével
- Arch Linux
  - sudo pacman -S SDL3 SDL3_image SDL3_ttf
- Ubuntu
  - sudo apt-get install libsdl3-dev libsdl3-ttf-dev libsdl3-image-dev

Telepítés Windowson:
- Telepítsük fel a Visual Studio 2022-es fejlesztői környezetet a C++ fejlesztői csomaggal
- Miután sikeresen fel van telepítve a vckpg, futtassuk ezt a parancsot
  - vckpg install sdl3 sdl3-image sdl3-ttf

Miután sikeresen megvannak a szükséges csomagok, klőnozzuk le a repót.

- git clone https://github.com/Ordinary56/Minesweeper

Ezután lépjünk be a főmappába:
- cd minesweeper

Miután beléptünk, lefordíthatjuk a programot egy preset segítségével a build mappába, például:

- cmake . --preset [PRESET_NEVE]
- cmake --build ./bin/[PRESET_MAPPA]
- a projekt jelenleg két preset-el rendelkezik: linux_x86_64 és windows_x86_64. Azt kérjük, hogy azt a preset-et futtasd le, amelyik operációs rendszeren szeretnéd fejleszteni a programot

A program sikeresen lefordult és megtalálható a bin mappában. Futtatásához rákattinthatunk, vagy pedig lefutatjuk parancssorban az alábbi parancsot.
- ./bin/minesweeper (.\bin\minesweeper.exe Windowson)

4. MAPPA STRUKTÚRA ÉS API

A projekt főkönyvtárában 6 mappa található, assets, bin, build, lib, src és toolchains. Az assets mappában található meg a programhoz szükséges fájlok, leginkább egyes játékelemeknek a képe. A bin mappába fordítjuk le a kész programot és 2 almappa található benne, windows és linux. Mindkét mappába az adott operációs rendszerhez fordított programot találjuk. A build mappa tartalmazza a Cmake által generált fájlokat és mappákat (pl.: CmakeCache.txt, MakeFile, stb…). A lib mappa tartalmazza a program fejléceit (header). Két fő mappája van, a core, és a gui. A core mappában deklaráljuk a játék logikai részét (AppState, timer, GameContext, stb), és a gui mappában deklaráljuk a játék grafikus megjelenítését (Scene, renderer). A src mappa megegyezik a lib mappa felépítésével, viszont a src mappában találhatjuk az implementációs részét a programnak. Ezenkívül a src mappában található meg a főprogram fájlunk, a main C fájl. A dokumentum következő része a program API részletes leírásáról fog szólni

4.1 Core/core.h

- AppState (struct)
  - Az AppState struct magában foglalja a játék legfőbb részeit. Célja, hogy a program futási idejének végéig hozzáférést biztosítson az egyes programrészekhez.
  - Tagok: [lásd: GameContext, RenderContext, TextureList, Scene]
- függvények
  - void App_set(AppState* context)
    - Megadja az AppState referenciáját a core.c implementációs fájlban. Szükséges, hogy a program grafikus részei is hozzáférjenek egyes komponensekhez, ha esetleg meg kell változtatni egyes tagokat.
  - Void App_quit(bool flag)
    - Megváltoztatja az AppState is_running boolean tagját hamisra, így ki fog lépni a program és felszabadítja az eddigi foglalt memóriát
  - const AppState* App_get(void)
    - Visszaad egy mutatót egy konstans AppState-re. Akkor érdémes használni, ha az a függvény amely meghívja ezt a függvényt nem fogja megváltoztatni a struktüra tagjait.
  - Const GameContext* App_get_gamecontext(void)
    - Visszaad egy mutatót egy konstans GameContext-re. Akkor érdémes használni, ha az a függvény amely meghívja ezt a függvényt nem fogja megváltoztatni a struktüra tagjait.
  - const RenderContext *App_get_rendercontext(void)

        - Visszaad egy mutatót egy konstans RenderContext-re. Akkor érdémes használni, ha az a függvény amely meghívja ezt a függvényt nem fogja megváltoztatni a struktüra tagjait.
    - AppState* App_get_mut(void)
    - GameContext* App_get_gamecontext_mut(void)
    - RenderContext* App_get_rendercontext_mut(void)
        - Ezek a függvények is úgy viselkednek, mint a konstans megfelelőjük, de viszont a hívónak biztosítania kell, hogy az alábbi objektumot megváltoztatják
    - SDL_Cursor *App_get_pointer_cursor(void)
    - SDL_Cursor *App_get_default_cursor(void)
        - Ez a két függvény lekéri az AppState kurzorait. Kétféle kurzor van, egy mutató, és egy alap. Mindkettő megfelel az egér egyes állapotának és azt az adott kurzort adják vissza. A kurzorok a main.c-ben vannak definiálva és hozzárendelve az AppState-hez.

4.2 Core/enums.h

- Itt található meg az összes globális felsoroáls típusú változók.
    - Enum GRID_SIZES
        - Megadja az adott rács méretét. Három állapota van. Beginner (9x9), Intermediate (16x16) és Expert (30x16).
    - Enum CURRENT_GAME_STATE_STATE
        - Megadja a játék jelenlegi állapotát
            - STOPPED: A játék megállt, nem megy tovább
            - PLAYING: A játék jelenleg folyamatban van
            - WIN, LOSE: Megadja, hogy a játékos nyert, vagy veszített
    - enum TIMERS
        - Megadja, hogy a játékos milyen időzítőt állított be. A tagokban vannak meg az adott időzítők percben megadva.
            - MIN_15: 15 perc
            - MIN_25: 25 perc
            - MIN_30: 30 perc

4.3 Core/game.h

- Ebben a fájlban található meg a játék logikai részéért felelős struct, a GameContext. A GameContext magában foglalja a rácsot, a sor és oszlopok számát, a lerakott aknák számát és a jelenlegi állapotot. A rács egy kétdimenziós, dinamikusan kezelt Cell tömb. A Cell struktúra megmondja, hogy fel van-e fedve a pályán, a játékos zászlót rakott-e rá vagy sem, és egy értéket. Az érték megmondja, hogy jelenleg hány szomszédos cellája akna. Az érték 0-tól 8-ig megy.

Létezik -1-es érték is, ami megmondja a celláról, hogy aknát tartalmaz.

- Függvények
  - void game_init_default(GameContext* ctx)
    - beállítja a játékot egy alapértelmezett értékre. Ez akkor kerül meghívásra, amikor a játékos nyert vagy veszített és törli az elöző rács tartalmait
  - void game_set_state(CURRENT_GAME_STATE state)
    - beállítja a GameContext state-jét (a játék jelenlegi állapotát)-
  - bool game_create_grid(GameContext* ctx, GRID_SIZES grid_size)
    - Dinamikusan lefoglalja a rácsot. A GRID_SIZES paraméter határozza meg a méretét. Igazat add vissza, ha sikeresen lefoglalta, ellenkező esetben hamist.
  - bool game_cell_in_bounds(GameContext *ctx, int r, int c)
    - Megmondja, hogy az adott sor és oszlopon lévő cella benne van-e a rácsban. Igaz, ha benne van, hamis, ha nincs benne.
  - void game_place_mines(GameContext *context, int num)
    - Lerak [num] db aknát a pályára, ezután beállítja az összes cella érték úgy, hogy azok megmondják hogy hány szomszédos cellájuk akna összesen.
  - void game_flood_fill(GameContext *context, int r, int c)
    - Rekurzívan felderíti az összes olyan cellát, aminek 0 az értéke, tehát egyik szomszédja sem akna. Ez a függvény akkor fut le, ha a játékos egy 0-s értékű cellára kattintott, és a függvény megáll, ha egy olyan cellához ért, aminek szomszédja legalább 1 akna.
  - void game_update(GameContext *context, void *data)
    - Ez a fúggvény felel a játék frissítési logikájának, ami minden egyes ciklusban meghívódik. Feladat, hogy nézze hogy a játékos melyik cellára kattintott illetve aszerint dönt a játék jelenlegi állapotáról. A függvény nem csinál semmit ha:
      - A játék jelenlegi állapota STOPPED
      - ha a játékos egy rácson kívüli helyre kattintott.
  - void game_cleanup(GameContext *context)
    - Felszabadítja a GameContext által lefoglalt memóriát. Ez a függvény a kilépés pillanatában hívódik meg.

4.4 Core/texture_list.h

- Itt található meg a TextureList láncolt listánk, amit a játék használ. A láncolt lista egy TextureEntry struktúrát

tartalmaz, ami egy kulcs és egy textúra tagból áll. Maga a lista egyetlen tagja az első eleme.

- Függvények
  - void texture_list_init(TextureList *list)
    - Dinamikusan lefoglalja a listát, és az első elemét NULL-ra teszi.
  - void texture_list_append(TextureList *list, const char *key, SDL_Texture *texture)
    - Hozzáad a listához egy elemet, egy adott kulcssal és a hozzá tartozó textúrával. A lista mindig előre fűzi a hozzáadott elemeket.
  - void texture_list_append_text(TextureList* list, SDL_Renderer* renderer, TTF_Font* font,UIButton button)
    - Hozzáad egy szöveget a listához, ami egy gombhoz van rendelve. Ezt a függvényt akkor hívódik meg, ha azt a gombot amit meg szeretnénk jeleníteni, még nincs szövege.
  - SDL_Texture *texture_list_get(const TextureList *list, const char *key)
    - Visszaad egy textúrát az adott [key] szerint. Ha megvan, akkor ahhoz a textúrához mutató pointer-t adja vissza. Ellenkező esetben NULL-t.
  - void texture_list_cleanup(TextureList *list)
    - Felszabadítja a lista által lefoglalt memóriát. Ez a függvény a program kilépésének pillanatában hívódik meg.

4.5 Core/timer.h

- Definiálja az időzítőt
- függvények
  - void timer_init(TIMERS timer)
    - Elindítja az időzítőt, amelyeta játékos beállított. Ez a függvény elindít egy SDL_AddTimer nevű függvényt, amely minden egyes másodpercen levon 1-et az időzítőből. Ha a hátra maradt idő 0, akkor a függvény nem fog tovább futni.
      - MEGJEGYZÉS: Az SDL_AddTimer külön szálon futtatja az átadott adatot.
  - int timer_get()
    - Visszaadja a hátramaradt időt milliszekundumban.
      - MEGJEGYZÉS: egy atomic változót add vissza, hogy elkerülje a fő és a mellékszál közt a holtversenyt.
  - Void timer_destroy()
    - meghívja az SDL_RemoveTimer függvényt, ezáltal eltávolítva a programból.

Core/utils.h

- Ebben a fájlban található az esetleg segédfüggvények, amiket általában a program indításán hívunk meg.
- Függvények:
  - void load_assets(TextureList *list, SDL_Renderer *renderer)
    - betölti az assets könyvtárban található összes képet és beszúrja a TextureList listába.
  - void load_tile_nums(TextureList* list, RenderContext* renderer)
    - Betölti a számokat 0-tól 8-ig, amit majd később a játék rácsa fog használni, hogy megjelenítse, hogy hány szomszédos cella akna.

4.6 Gui/render.h

- Ez a fájl tartalmazza a RenderContext struktúrát, ami a grafika megjelenítéséért felelős. A struktúrának van egy SDL_Window pointere, egy SDL_Renderer pointere, és egy TTF_Font pointere. Az ablak alapértelmezett mérete 800x600.
- Függvények
  - bool render_init(RenderContext *rc)
    - Inicializálja a RenderContext objektumnak tagjait. Ha sikerült, akkor egy true értékkel tér vissza, ha egyet nem sikerült, akkor false-al tér vissza. A program működéséhez szükséges, hogy mind a három mező sikeresen be legyen töltve a RenderContext-be.
  - void render_draw(RenderContext *rc, void *scene)
    - Az adott jelenetet rajzolja le az ablakra. A rajzolás menete a következő
      - A függvény fehérre állítja az ablakot.
      - Ezután kirajzolja, amit a scene draw függvényében van leírva.
      - Végül ezt kivetíti az ablakra az SDL_RendererPresent függvénnyel.
  - void render_cleanup(RenderContext *rc)
    - Felszabadítja a RenderContext struktúra által lefoglalt összes memóriát.

4.7 Gui/scene.h

- Ez a fájl tartalmazza a Scene nevű struktúrát. Ahhoz hogy külön válasszük a főmenüt és a játékot grafikusan, „jelenetekre” kell bontani, amik között váltogathatunk. Egy scene függvényekre mutató tagokat tartalmaznak. Az alábbi függvényekre tudnak mutatni a jelenetek:
  - init: inicializálja az adott jelenetet és egyéb objektumokat

    - update: egy függvény, amit a főprogramunk ciklusa minden egyes lefutáskor meghív. Célja, hogy frissítse az adott jelenet logikáját (pl.: játékos rákattintott egy gombra)
    - draw: Ezt függvény fogja meghívni a RenderContext. Ez a függvény írja, hogy hogyan néz ki a jelenet grafikusan.
    - cleanup: Felszabadítja az esetleges objektumokat, referenciákat, amit az adott jelenet lefoglalt
- függvények:
    - void scene_manager_init(void* app)
        - A scene.c implementációs fájlban eltárolt referenciát állítja be úgy, hogy az AppState-re mutasson.
    - void scene_change_to(Scene* newScene, void* data)
        - megváltoztatja az AppState jelenleg eltárolt jelenetjét

4.8 Gui/ui_element.h

- Ez a deklarációs fájl tartalmazza a UIButton struktúráját. A fájlban megtalálható még egy ButtonAction típus is, ami egy void visszatérési értékű függvényre mutat. A UIButton-nek van egy doboza (SDL_Frect), szövege (label), a szövegnek van színe (color), és egy ButtonAction-je is, vele együtt az átadandó adattal.
- Függvények
    - UIButton button_create(const char* label, SDL_Color color, void* data, ButtonAction action)
        - A kapott paraméterek alapján ad vissza egy UIButton structot. Érdemes ezt a függvényt egy makróba tenni.
    - void button_set_pos(UIButton* button, float x, float y, float w, float h)
        - beállítja az átadott [button]-nek a helyét.
    - void button_play(void* data)
        - elindítja a játékot
    - void button_exit(void* data)
        - bezárja a programot
    - void button_set_grid_beginner(void* data)
        - Beállítja a rácsot 9x9-es méretre
    - void button_set_grid_intermediate(void* data)
        - Beállítja a rácsot 16x16-os méretre.
    - void button_set_grid_expert(void* data)
        - Beállítja a rácsot 30x16-os méretre.
    - void button_set_timer_15min(void* data)
        - Beállítja az időzítőt 15 perce.
    - void button_set_timer_25min(void* data)
        - Beállítja az időzítőt 25 perce.
    - void button_set_timer_30min(void* data)
        - Beállítja az időzítőt 30 perce.

- Az alábbi felsorolt függvények azok, amík az egyes gombok ButtonAction tagjai meghívnak ha rákattintottak.

4.9 Gui/ui_states.h

- Ez a fájl tárolja az egyes menüknek az állapotát. A program jelenleg a főmenü állapotát tárolja el egy UIState struktúrűban, ami számon tartja a felhasználó által bekért rács méretet és időt.

TARTALOMJEGYZÉK