

# “Dungeon Crawler” – Programozói dokumentáció

Dúcz Ákos – GC1RTE

## 1. Felépítés

A program 3 fő modulra oszlik, ezen belül pedig különböző forrásfájlokra. A “Dungeon” mappában találhatóak a pálya generálásával, kirajzolásával és kezelésével kapcsolatos függvények és adatstruktúrák. Az “Entities” mappa felelős a játékban szereplő szörnyek, a játékos, és egyéb entitások kódjáért. Hasonló módon az “Items” mappa a játékos által használható tárgyakat és fegyvereket gyűjti össze.

Ezen kívül a “Helpers” mappa tartalmaz minden egyéb máshova nem sorolható segítő függvényt, mint például az A\* útkereső algoritmus, vagy egyes kirajzolást segítő funkciók.

A “game.c” nevű fájl a fő forrásfájl, ez include-ol minden más forrást, illetve itt történik a program fő ciklusának futtatása is.

## 2. “game.c”

Ez a fő forrásfájl, itt található a main() függvény.

```
int main() {  
  
    srand(time(0));  
  
    initscr();  
    start_color();  
    nodelay(stdscr, TRUE);  
    keypad(stdscr, TRUE);  
    curs_set(0);  
  
    printf("> Screen init.\n");
```

Az első néhány sor az “ncurses” nevű könyvtár inicializálására szolgál. Ez teszi lehetővé a karakterenkénti, színes kirajzolást a képernyőre.

Ezután létrehozuk a pályát (**DungeonLevel** struktúra), majd le is generáljuk a benne lévő szobákat. Ezért a “DungeonLevel.c” felelős.

```
DungeonLevel level0;  
  
DungeonLevel_InitLevel(&level0);  
DungeonLevel_ClearLevel(&level0);  
DungeonLevel_GenerateLevel(&level0);  
  
printf("> Map generated.\n");  
  
EPlayer* player = Spawn_EPlayer(&level0, 13,13);  
  
Give_IGold(&player->baseEntity, 13);  
Give_IFlintlock(&player->baseEntity);  
  
printf("> Player spawned.\n");
```

Létrehozuk a játékost (**EPlayer** struktúra), majd adunk neki néhány tárgyat.

Ezután belépünk a program fő ciklusába, mely addig tart ameddig a játékos életben van. Ebben a ciklusban meghívjuk a pálya kirajzolásáért felelős függvényeket, illetve az entitások saját logikáját is.

A játékos halála esetén az összes entitást kitöröljük, majd kilépünk a programból (itt később a dicsőséglista kiírása lesz).

### 3. Dungeon mappa

#### 3.1 DungeonLevel

Ezek a forrásfájlok a **DungeonLevel** struktúráért, és annak függvényeiért felelnek. Ez a struktúra tartalmazza a pályát alkotó csempék 2-dimenziós tömbjét, a pályán élő entitások listáját, a jelenlegi játékosra mutató pointert, illetve néhány egyéb adatot.

Főbb függvények:

**InitLevel:** a pálya alapvető adatainak beállítása. Generálás előtt meghívandó.

**ClearLevel:** a pálya csempe-mátrixát tölti meg az alapértelmezett csempével (CaveWall)

**DrawLevel:** kirajzolja a pálya jelenleg látható részét a képernyőre. (a ViewCenterX/Y változók alapján)

**GenerateLevel:** a pálya szobáinak, útjainak, entitásainak generálása. Ehhez igénybe veszi a DungeonTile és DungeonRoom forrásokat is. Először létrehoz egy kezdőszobát, majd ebből jobbra és lefele újabb szobákat hoz létre randomizált távolságra. Az újonnan generált szobákat összeköti az eddigivel, majd ezt a lépést ismétli az új szobákra is. Így kapjuk a játékban található szoba-labirintust.

Emellett véletlenszerű alagutakat illetve kijáratokat is generál.

**AddEntity:** hozzáad egy (már létező) entitás-objektumot a pályán nyilvántartott entitások közé. Ezzel növelve az EntityCount változót is. Ha a LevelMaxEntities értéket meghaladná az entitások száma, akkor hibát jelez.

**RemoveEntity:** kivesz egy entitást a pályán nyilvántartott entitások közül, csökkenti az EntityCount értéket, illetve a listában eggyel visszamozgat minden utána következő entitás-pointert. Figyelem, ez nem szabadítja fel az entitás objektumot a memóriából!

**DeSpawnAllEntities:** Meghívja a pályán élő minden entitásra az Entity\_deSpawn függvényt, amíg van entitás a pályán. A pálya struktúra törlése előtt kötelező használni.

**FindLoadedEntities:** Megkeresi azon entitásokat, melyek megfelelően közel vannak a játékoshoz, és beteszi őket a LoadedEntities listába. Később csak ezeket fogjuk mozgatni, hiszen túl sok entitás lelassítaná a programot.

**OnTurnEntities:** Meghívja a LoadedEntities listában szereplő összes entitás onTurn függvényét, ezzel mozgatva a szörnyeket és magát a játékost is. Erről bővebben az Entitások mappánál.

#### 3.2 DungeonTile

A "Tile" azaz csempe struktúra definíciójáért és kirajzolásáért felel. Ezekből áll a DungeonLevel struktúra csempe-mátrixa.

Tulajdonságai:

**int id;** // egyedi szám, mely megkülönbözteti a Tile típusokat.

**char symbol;** // a kirajzolandó ASCII karakter

**short foreColor;**

**short backColor;** // előtér és háttér színe

**bool walkable;** // átjárható-e a csempe?

Függvények:

**DrawTile:** kirajzolja a csempét a képernyő megadott koordinátáira.

### 3.3 DungeonRoom

A **DungeonRoom** struktúrát definiálja, mely csupán a pálya generálásához kell, utána nem marad meg a memóriában. Ez a struktúra rendelkezik egy adott szoba helyzetével és méreteivel.

Függvények:

**GenerateRoom:** Megadott x-y koordinátára generál szobát a megadott maximum méreteken belül, és feltölti a **DungeonRoom** struktúrát ennek információjával. A szobába véletlenszerűen rakhat ellenségeket vagy kincsesládákat is.

**GeneratePath:** az A\* útkeresés segítségével utat hoz létre a labirintus két pontja között.

**ConnectRooms:** két megadott szobát köt össze a **GeneratePath** által létrehozott út segítségével, illetve a megfelelő helyekre ajtót is tesz. Logikája viszonylag bonyolult, mivel automatikusan a másik szobához legközelebbi falra teszi az ajtót.

**CanBuildRoom:** egyszerű helper függvény, megnézi egy adott négyzeten belül nincs-e akadálya egy szobát lerakni.

## 4. Helpers mappa

### 4.1 Drawing

Néhány egyszerű kirajzolási függvény gyűjteménye:

**WorldToScreen / ScreenToWorld:** a képernyő / játéktér koordináták közti átváltást kezelik. A **WorldToScreen** visszatérési értéke hamis, ha a képernyőn nem látszik az adott pont.

**CanSee:** megadja, hogy a pálya egyik pontjából lehet-e látni egy másikat. Ennél a csempék átláthatóságát nézi meg a két pont között húzott egyenesen. Az egyeneshez Bresenham algoritmusát alkalmazzuk.

**WaitForInput:** mikor a játék bemenetet vár a felhasználótól, ezt a függvényt hívja meg. Ez addig vár amíg le nincs nyomva egy billentyű, és addig folyamatosan kirajzolja a pályát. Amint egy billentyű lenyomásra kerül, visszatér ennek értékével. A "\*" karakter pedig kilép a programból.

**WriteText:** a játéktér alatt kirajzolt "konzolba" ír ki egy stringet. Itt lehet szöveges információt közölni a játékoskal.

### 4.2 Pathfinding

Az A\* útkeresés implementációja.

Adott **Node** struktúra, mely a megtalált út egyetlen lépését jelenti. Ez tartalmaz egy x-y koordinátát, illetve a "szülő-node" koordinátáit. Ezen kívül tartalmaz az A\* heurisztikához használatos g értéket is.

A **NodeList** struktúra Node-ok listája, mely hozzáadáskor/elvételkor dinamikusan nő és csökken.

NodeList függvényei:

**InitNodeList / FreeNodeList / ClearNodeList:** A NodeList inicializálása, minden elemének kitörlése, vagy a teljes NodeList objektum memóriájának felszabadítása.

**AppendToNodeList/ PopNodeList:** hozzáad/elvesz egy elemet a NodeList-ből, és dinamikusan növeli/csökkenti annak tárhelyét.

A\* függvényei:

**PopLowestFScore:** az A\* heurisztika szerinti legkedvezőbb Node kiszedése egy listából, majd annak visszaadása.

**FindNodeIndex:** Egy adott Node indexének megtalálása egy listában. (x-y alapján)

**Backtrace:** visszamegy az útkeresés által megtalált legrövidebb úton, és a kimenő NodeList-hez csatolja annak elemeit.

**Pathfind:** A\* útkereső algoritmus implementáció ezen forrás alapján:

[https://en.wikipedia.org/wiki/A\\*\\_search\\_algorithm#Pseudocode](https://en.wikipedia.org/wiki/A*_search_algorithm#Pseudocode)

Megadható neki, hogy ignorálja-e a barlang fala illetve ajtók által állított akadályokat.

## 5. Entities mappa

### 5.1 Entity általános struktúra

Az általános **Entity** struktúra tartalmazza egy entitás minden fontos adatát. Ezek többek között: x-y koordináta, név, ASCII karakter és színek, megölésért járó pontok, státusz-effektek, sebesség, életerő, Inventory-lista, jelenlegi pálya, és néhány egyéb.

Ezen felül tartalmaz függvény-pointereket, melyek különböző entitás-típusokhoz tetszőlegesen állíthatók. Ezek megadják az entitással történő lehetséges interakciókat, illetve az entitások logikáját.

Az Entity struktúra virtuális függvényei:

**onTurn:** akkor hívódik meg, mikor a pályán lévő összes entitást mozgatjuk (lásd: DungeonLevel). Itt az entitás saját logikája szerint mozoghat, támadhat, stb.

**draw:** a pálya kirajzolásakor ezt a függvényt is meghívjuk. Itt az entitás kirajzolhatja a különböző speciális effektusait, támadását, stb.

**deSpawn:** az entitásnak itt kell felszabadítania az általa külön foglalt memóriát. Halálakor hívódik meg, illetve itt döntheti el hogy például dob-e valamilyen kincset.

**damage:** az entitás sebzésekor hívódik meg.

**Interaction\_[valami]:** ezek a lehetséges interakciók amiket a játékos az entitással tehet. Egyelőre csak a "loot", azaz kifosztás van definiálva.

Ezeket a függvényeket viszont sosem közvetlenül hívjuk meg, hanem a megfelelő globális függvények segítségével. Ezek kezelik a minden entitásra azonosan vonatkozó kód lefuttatását, illetve meghívják az adott entitás saját virtuális függvényeit.

```
// ha az entitas lehetoseget kap egy akciora, (speed hatarozza meg), akkor ez
// a fuggveny lesz meghivva. Egyes statusz effektek hatassal lehetnek erre (pl: Freeze)
void Entity_OnTurn(Entity* entity);

// ez minden globalis korben (vagy tick-ben) meghivodik, fokent a statusz effektek idejenek csokkentese szol.
void Entity_OnGameTick(Entity* entity);

// az entitas kirajzolasa, ez meghivja az entitas kulon onDraw funkciojat is
void Entity_Draw(Entity* entity);

// Az entitas halalakor jelentos, free-eli az entitas memoriajat, illetve az osszes targyat mely hozza tartozott.
// Ez meghivja az onDeSpawn fuggvenyt, ahol az entitast oroklo objektumnak fel kell szabaditania a saját memoriaigényeit
void Entity_deSpawn(Entity* entity);

// Ha egy entitas megtamad egy masikat, akkor ez a fuggveny fut le.
void Entity_Damage(Entity* entity, Entity* attacker, int points);

// Interakcio: egy entitas kifoszt egy masikat.
void Entity_Interact_Loot(Entity* entity, Entity* looter);

// beallitja az entitas utkeresesenek celpontjat. Ezutan az entitas minden koreben ezt az utat fogja kovetni,
// amig ez meg nem valtozik vagy torlodik.
void Entity_SetDestination(Entity* entity, int y, int x);

// kitorli az entitas celpontjat, innentol nem fog utvonalat kovetni.
void Entity_ClearDestination(Entity* entity);

// hozzaad egy targyat az entitas inventory-hoz, es frissiti az item "owner" valtozojat
bool Entity_AddItemToInventory(Entity* entity, Item* item);

// elvesz egy targyat az inventory-bol, pointer alapjan. Az item regi "owner" erteke megmarad, de ez mar nem valid.
void Entity_RemoveItemFromInventory(Entity* entity, Item* item);
#endif
```

### [A dokumentáció félkész, a további függvények később lesznek részletezve]

A játékban sokféle entitás megtalálható: ilyenek a szörnyek (crawler, rat), maga a játékos, vagy akár a megtalálható eldobott tárgyak és ládák is.

A játékos onTurn függvénye felelős a felhasználótól bekért bemenetért is, mely alapján mozgatja a játékost / támad.

### [A dokumentáció félkész, az egyéni entitások később lesznek részletezve]

Mind a tárgyak, mind az entitások esetében az öröklő struktúrák magukkal hordozzák az eredeti Item vagy Entity struktúra definícióját az alábbi módon, majd erre építenek tovább. A virtuális függvények beállításával pedig egyedi interakciók érhetőek el az öröklő objektumok esetében.

```
typedef struct ECrawler { // crawler öröklő struktúra az eredeti Entity struktúrával, illetve
    Entity baseEntity;    egy extra változóval.
    int biteCooldown;
} ECrawler;
```

## 6. Items mappa

### 6.1 Item általános struktúra

Hasonlóan az entitásokhoz, itt is egy általános "ős-struktúrából" ered az összes egyedi tárgy-struktúra. Ez tartalmazza a név, szimbólum, tulajdonos, halmaz-méret értékeket, és itt is találhatóak egyedien beállítható virtuális függvények. Az itt elérhető interakciók közül egyelőre csak a támadás, újratöltés és elfogyasztás elérhetőek.

Függvények:

**DrawEffects:** a tárgy speciális effektusait rajzolja ki, például a Flintlock pisztoly esetében támadáskor a töltény útját.

**delete:** a tárgy itt szabadul meg extra memóriaigényeitől, mielőtt free-elve lesz

**[A dokumentáció félkész, az egyéni tárgyak később lesznek részletezve]**