“Dungeon Crawler” – Programozói dokumentáció

Dúcz Ákos – GC1RTE

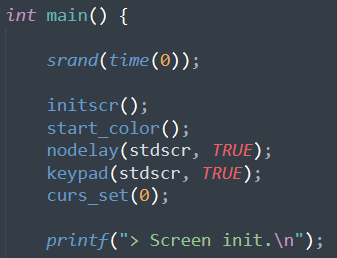
# Felépítés

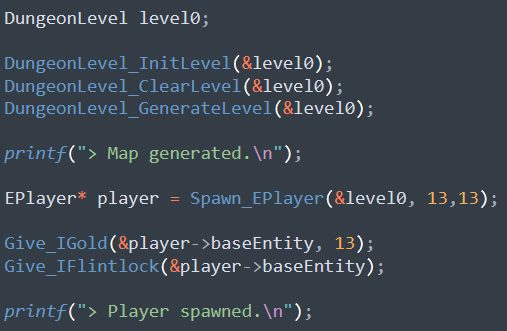
A program 3 fő modulra oszlik, ezen belül pedig különböző forrásfájlokra. A “Dungeon” mappában találhatóak a pálya generálásával, kirajzolásával és kezelével kapcsolatos függvények és adatstruktúrák. Az “Entities” mappa felelős a játékban szereplő szörnyek, a játékos, és egyéb entitások kódjáért. Hasonló 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.

# “game.c”

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

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

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

Létrehozzuk 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, majd kiírjuk a dicsőséglistát.

A dicsőséglista a “scoreboard.txt” nevű fájlban van tárolva. A program először megnyitja a fájlt, majd elkezd végiterálni a benne tárolt sorokon. Amint egy olyan értéket talál, mely kisebb mint a saját pontszámunk, beilleszti a saját pontszámot ezen sor elé. Ezzel eléri, hogy a dicsőséglista mindig pontok alapján csökkenően rendezve legyen. Az új lista a “scoreboard\_updated.txt” nevű fájlba kerül, melyet a program kilépés előtt átnevez “scoreboard.txt”-re.

# Dungeon mappa

## 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:

**void InitLevel (DungeonLevel\*):** a pálya alapvető adatainak beállítása. Generálás előtt meghívandó.

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

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

**void GenerateLevel(DungeonLevel\*):** 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.

**void AddEntity(DungeonLevel\*, Entity\*):** 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.

**void RemoveEntity(DungeonLevel\*, Entity\*):** 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!

**void DeSpawnAllEntities(DungeonLevel\*):** Meghívja a pályán évő minden entitásra az Entity\_deSpawn függvényt, amíg van entitás a pályán. A pálya struktúra törlése előtt kötelező hasznáni.

**void FindLoadedEntities(DungeonLevel\*):** Megkeresi azon entitásokat, melyek megfelelően közel Vannak a játékoshoz, és beteszi őket a LoadedEntities listába. Később cask ezeket fogjuk mozgatni, hiszen túl sok entitás lelassítaná a programot.

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

## 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:

**void DrawTile(int y, int x, Tile):** kirajzolja a csempét a képernyő megadott x-y koordinátáira.

# 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:

**bool GenerateRoom(DungeonLevel\*, int y, int x, int maxH, int maxW, DungeonRoom\* roomInfo):** Megadott x-y koordinátára general szobát a megadott maxmimum méreteken belül, és feltölti a DungeonRoom struktúrát ennek információjával. A szobába véletlenszerűen rakhat ellenségeket vagy kincsesládákat is.

**bool GeneratePath(DungeonLevel\*, int startY, int startX, int endY, int endX, bool special):** az A\* útkeresés segítségével utat hoz létre a labirintus két pontja között. Ha nem lehetséges, akkor hamis a visszatérési értéke, egyébként igaz.

**void ConnectRooms(DungeonLevel\*, DungeonRoom\* roomA, DungeonRoom\* roomB, bool specialPath):** 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.

**bool CanBuildRoom(DungeonLevel\* level, int y, int x, int height, int width):** egyszerű helper függvény, megnézi egy adott négyzeten belül nincs-e akadálya egy szobát lerakni.

# Helpers mappa

## Drawing

Néhány egyszerű kirajzolási függvény gyűjteménye:

**bool WorldToScreen(int y, int x, int \*sy, int \*sx)** és **void ScreenToWorld(int y, int x, int \*wy, int \*wx):** a képernyő / játéktér koordináták közti átváltást kezelik. A WorldToScreen visszatérési értéke hamis, ha a képernyőn nem látszik az adott pont.

**bool CanSee(DungeonLevel\*, int fromY, int fromX, int toY, int toX):** 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.

**char WaitForInput(DungeonLevel\*):** 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.

**void WriteText(char\* text):** a játéktér alatt kirajzolt “konzolba” ír ki egy stringet. Itt lehet szöveges információt közölni a játékossal.

## 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 egy láncolt lista, mely tartalmazza az útkereséskor megtalált út koordinátáit.

NodeList függvényei:

**int NodeListLength(NodeList\*):** Egy NodeList hosszának megadása int-ként.

**NodeList\* AppendToNodeList(NodeList\*, Node n)** és **NodeList\* PopNodeList(NodeList\*, int index, Node\* out):** hozzáad/elvesz egy elemet a NodeList-ből. Az Append mindig a végére teszi az új elemet, Pop esetén pedig a megadott indexű elemet veszi ki a listából, és írja be az out változó helyére.

**Node GetNodeAtIndex(NodeList\*, int index):** visszaadja az adott indexű Node-ot a listából.

**void FreeNodeList(NodeList\*):** NodeList memória felszabadítás.

A\* függvényei:

**NodeList\* PopLowestFScore(NodeList\*, int goalY, int goalX, Node\* outNode):** az A\* heurisztika szerinti legkedvezőbb Node kiszedése egy listából, majd annak visszaadása.

**int FindNodeIndex(NodeList\*, Node\*):** 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.

# Entities mappa

## 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:

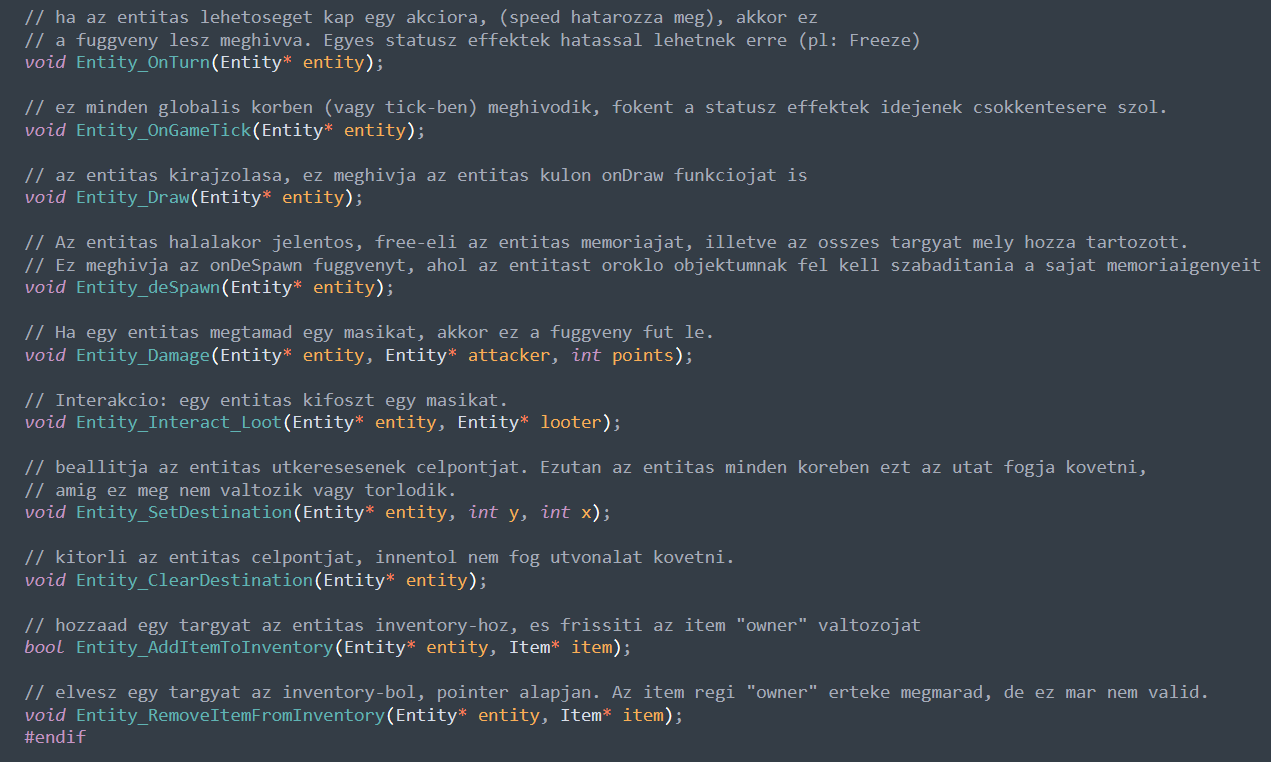
**onTurn:** akkor hívódik meg, mikor a pályán lévő összes entitást mozgatjuk (lásd: DungeonLevel). Itt az entitás saját logikája szerint mozoghat, támadhat, stb.

**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.

Globális Entitás függvények:

**void Entity\_OnTurn(Entity\*):** Akkor kell meghívni, ha az entitásnak “lépnie” kell egyet, azaz ha éppen az adott entitás következik. Meghívja az entitás saját onTurn függvényét, illetve ha van beállítva útkeresés (Set/ClearDestination), akkor előremozgatja ezen az úton. Ha az entitás “frozen” státuszban van, akkor nem mozgatja.

**void Entity\_OnGameTick(Entity\*):** Minden körben meghívandó ha az entitás jönne, akkor is ha ebben a körben neki nem szabad mozognia. Ez csökkenti a rá vonatkozó effektek élettartamát. (Pl: fagyás számláló csökkentése).

**void Entity\_Draw(Entity\*):** kirajzolja az entitást a pályára. Emellett meghívja az entitás saját kirajzoló függvényét is.

**void Entity\_deSpawn(Entity\*):** meghívja az entitás saját deSpawn függvényét, kitörli az útkeresést (ha van), és felszabadítja a foglalt memóriát. Emellett törli az inventory-ban tárolt összes tárgyat is. Ezután meghívja a DungeonLevel\_RemoveEntity függvényt, ezzel törölve az entitást a pályáról.

**void Entity\_Damage(Entity\* entity, Entity\* attacker, int points):** ezzel a függvénnyel lehet megsebezni egy entitást. Paraméterei a támadott, a támadó, és az elveszítendő életpontok száma. Ha ezután a támadott entitásnak nem marad életereje, akkor meghívja az Entity\_DeSpawn függvényt, és növeli a támadó pontjait.

**void Entity\_Interact\_Loot(Entity\* entity, Entity\* looter):** “kifosztás” interakció egy adott entitással. Az egyik entitás összes tárgyát a másik entitás kapja meg. Ládák / eldobott tárgyak esetén érvényes.

**void Entity\_SetDestination(Entity\*, int y, int x)** és void **Entity\_ClearDestination(Entity\*):** útkeresést indít egy megadott pontba az entitás számára. Ezután az minden körben megpróbál a megadott ponthoz közelebb jutni, míg oda nem ér, vagy a ClearDestination függvénnyel ki nem töröljük az utat.

**bool Entity\_AddItemToInventory(Entity\*, Item\*)** és **RemoveItemFromInventory:**

Hozzáad/elvesz egy tárgyat az entitás inventory-ból. Hozzáadáskor, ha a tárgy stackelhető (pl: arany), akkor csak növeli a stack méretét.

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

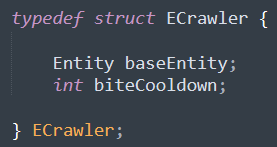
A játékos onTurn függvénye felelős a felhasználótól bekért bemenetért is, mely alapján mozgatja a játékost / támad.

Minden szörny / entitás esetében a döntést hozó logika az onTurn függvényben van leírva. A játékos entitás is itt kér bemenetet a felhasználótól, és ez alapján mozgatja a karaktert. Az egyes entitások kódja külön mappában van az Entities mappán belül. Itt egy NÉV.h és egy NÉV.c fájl található, melyek definiáják az Entity struktúrától örökölt specifikus struktúrát, és annak függvényeit. Az onTurn, onDamage, stb függvények mellett megtalálható egy **Spawn\_[Név]** függvény is, mely meghívásakor létrehoz a memóriában egy adott entitást, és elhelyezi azt a pályán.

A játékos logikája az Entities/Player/EPlayer.c nevű fájlban van leírva. A játékos onTurn függvénye kér egy karaktert a felhasználótól. Ha ez (wasd), akkor a megfelelő irányban mozgatja, ha “e”, akkor meghívja az interakció-menü függvényt, ha pedig (0-9), akkor az inventory-ban tárolt megfelelő tárgy interakcióit kínálja fel lehetőségként. Az onTurn azt is ellenőrzi hogy a játékos “kijárat” csempén áll-e, és ha igen akkor ezzel befejezi a játékot.

A játékos onDraw függvénye pedig kiírja a játékos életerejét, aktuális pontszámát, és kirajzolja az inventory-t is.

Ez a felépítés minden más entitás esetében is hasonlóan működik.

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. 

// crawler öröklő struktúra az eredeti Entity struktúrával, illetve

egy extra vátozóval.

# Items mappa

## 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 taláhatóak 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

**Interaction\_[valami]:** ezek a lehetséges interakciók amiket a játékos a tárggyal tehet.

Az entitásokhoz hasonló módon itt is globális függvények segítségével kell meghívni a tárgy-specifikus függvényeket.

Tárgy globális függvények:

**void ItemInteractMenu(Item\*):** felkínálja a játékosnak a tárggyal való lehetséges interakciókat. Ez a játékos onTurn függvényéből kerül meghívásra.

**void Item\_Delete(Item\*):** felszabadít egy tárgyat a memóriából, és kiveszi a pointerét a tulajdonosának inventoryjából.

**void Item\_DrawEffects(Item\*):** meghívja az adott tárgy DrawEffects függvényét.

**Item\_Interact\_[valami]:** interakció az adott tárggyal, meghívja a hozzátartozó interakció függvényt.

A különböző tárgyak az Items mappán belül saját mappákban vannak, ahol egy NÉV.h és egy NÉV.c fájl írja le a logikájukat. A tárgy-specifikus függvényeken (DrawEffects, delete, stb) kívül egy **Give\_[NÉV]** függvény is definiálva van itt, amely meghíváskor létrehoz egy adott tárgyat és elhelyezi a megadott entitás inventoryjában.

Az összes tárgy kódja a tárgy-specifikus függvényeken belül van leírva, így a program képes mindet a közös “Tárgy” struktúraként kezelni. Megvalósításuk (ahogy az entitások esetében is) ugyanazokon a függvényeken alapszik, és lényegében nem tér el egymástól, így a különböző tárgyak pontos logikáját részletesebben nem tárgyalom.

# NCurses könyvtár

Az ncurses könyvtár lehetővé teszi a konzolba való karakterenkénti (színes) kirajzolást.

Függvényei közül főként az mvprintv-t használja a program, mely egy megadott stringet ír ki a képernyő valamely pontjára. A színes kirajzolásért a Drawing.c ColorPrintChar függvénye felel, mely egyetlen karaktert rajzol ki színesen az ncurses segítségével.

NCurses-ben csupán 8 féle háttér- és előtér szín érhető el, így maximum 64 féle színkombináció választható ki, de ez bőven elegendő a játék grafikájához.