OwlDodgeGame fejlesztői dokumentáció

Készítette: Foki Lénárd

Tartalomjegyzék

1. Bevezető

Az “owldodgegame” egy felülnézetes játék, amiben a játékos az infoc oldalról ismert baglyot irányítja, úgy hogy minél tovább életben maradjon, azáltal hogy kikerüli az ellenségeket és a tűzgolyókat.

Ebben a dokumentációban szó lesz arról, hogyan kell futtatni a programot, hogyan van implementálva a kód, és hogy miként ajánlott változtatni/fejleszteni a kódon.

2. Telepítési útmutató

A programhoz gcc és az SDL2 könyvtár szükséges, ennek a letöltéséhez nyújt segítséget a hivatalos honlap:

https://wiki.libsdl.org/SDL2/Installation

gcc-vel a következőképpen lehet lefordítani, ha az owldodgegame mappában vagyunk:

```
gcc src/*.c -g -o [fájlnév] `sdl2-config --cflags --libs` -lSDL2_gfx -
lSDL2_image -lSDL2_ttf -lm -Wall -Werror
```

A kapott [fájlnév] nevű fájlt futtatva indul el a program.

3. Fájlszerkezet

A owldodgegame gyökérkönyvtárában a következő mappák találhatóak meg:

- docs: a két dokumentációt tartalmazza
- src: az összes *.c alakú fájl itt található meg
- resources: a program futtatásához szükséges képeket tárolja
- include: a *.c fájlokhoz tartozó fejléc fájlok találhatóak meg (tehát *.h alakúak)
- lib: a külső könyvtárak tárolódnak itt (jelen esetben a debugmalloc.h)
- fonts: a használt betűtípus található itt meg

Ezen kívűl a dicsőséglista a projekt gyökérmappájában helyezkedik el *“scoreboard.txt”* néven.

A futtatható fájl is a gyökérmappában jelenik meg fordítás után.

4. Adatszerkezetek

Először is a program állapotait kell számontartani. Több állapot is van, ezeket célszerű egy felsorolás típusú változóban tárolni. Mivel az állapotra a program nagyrészében szükség lehet, ezért célszerű egy modulban tárolni a jelenlegi állapotot. Az enkapszuláció miatt nem szimplán globális változóként lesz nyilvántartva, hanem függvényeken keresztül lehet elérni és felülírni a jelenlegi állapotot.

4.1 state

A programot három fő állapotra lehet szétbontani, ezt a következő felsorolt típusban lehet eltárolni:

```
typedef enum MainState { MENU, GAME, QUIT } MainState;
```

Mivel a menün belül még vannak almenük, ezért azokat is el kell tárolni szintén egy felsorolt típusban:

```
typedef enum MenuState {STARTMENU,HELPMENU,GAMEOVERMENU,} MenuState;
```

A játék állapotának tekinthető a nehézségi fokozat is, amely több modulban is el kell érni, ezért ugyanúgy lesz eltárolva mint az előző állapotok:

```
typedef enum Difficulty { EASY, MEDIUM, HARD } Difficulty;
```

Alapjáraton a főállapot és a menüállapot tartozik logikusan egy változó alá, ez egy States típusú változó lesz.

```
typedef struct States {MainState main;MenuState menu;} States;
```

A state modulban kezdeti értéket állítunk az állapotoknak:

```
States state = {MENU, STARTMENU};
```

```
Difficulty difficulty = EASY;
```

Ezeket az állapotokat getter és setter függvények segítségével lehet elérni, ezek a függvények a modul fejlécében vannak definiálva a következő módon:

```
MainState getmainstate() { return state.main; }
MenuState getmenustate() { return state.menu; }
Difficulty getdifficulty() { return difficulty; }
void setmainstate(MainState s) { state.main = s; }
void setsubmenustate(MenuState s) { state.menu = s; }
void setdifficulty(Difficulty diff) { difficulty = diff; }
```

4.2 math_helper

Ezek után a program középpontjában a különböző entitások vannak, és annak eltárolási módjai. Először a játékos adatstruktúráját kellene leírni, de ahhoz szükség van más struktúrákra is.

A matematikai struktúrákat kell először felépíteni.

Először szükség van egy pont eltárolására. A következő struktúra egy pontot jelképez a koordinátarendszerben valós típusú x és y változókkal.

```
typedef struct Point {double x, y;} Point;
```

Egy origó kezdőpontú vektort ír le az alábbi struktúra valós típusú x és y változókkal:

```
typedef struct Vector2 {double x, y;} Vector2;
```

Egy méretet egész típusú szélesség és magasság változókkal leírt struktúra.

```
typedef struct Size {int width, height;} Size;
```
Egy téglalapot ír le egy pont és egy mérettel. A pont a téglalap közepére mutat.

```
typedef struct Rect {Point pos;Size size;} Rect;
```

4.3 entity

A különböző entitások struktúrája található itt meg.

Az entitásoknál nagyon sok a közös elem és tulajdonság, ezért van az, hogy a tűzgolyók és az ellenségeknek közös struktúrájuk van.

A gameobject struktúra szedi össze az úgy általánosságban megtalálható tulajdonságokat egy játékobjektumnál, mint például a pozíciója, sebessége, kinézete stb.

Az entitásokat egy láncolt listában tárolom, mivel folyamatosan az idő előrehaladtával jönnek létre és tűnnek el, ennek az adatszerkezetnek gyors a beszúrás és törlés művelete.

A játékos lövedékeiből is elméletileg több lehet, hiszen ha kilő egyet, és mielőtt a kilőtt darab megtette volna az útját, ha elég rövid a töltési ideje a képességnek, akkor még egy lövedéket képes lenne használni.

A játékos objektumnak van ebben a programban a legtöbb mezője, mivel ez a program központi adatszerkezete.

A Spell struktúra tárolja az olyan tulajdonságait egy képességnek, amelyek közösek az összes elemre nézve.

```
/**
* @struct Cooldown
* @brief A töltési idő kezeléséhez szükséges adatokat tárolja
*/
typedef struct Cooldown {
const double cd; /**< Az az idő amely két képesség használat között el
kell teljen (másodperc) */
double
cdcounter; /**< ez az adat számol vissza a cooldown értékéről nulláig */
bool oncd; /**< azt tartja számon, hogy töltési időn van-e a képesség, ha
igen
akkor nem használhatja a képességet a játékos */
} Cooldown;
/**
* @struct Spell
* @brief A játékos egy képességhez tartozó tulajdonságokat tároló struktúra
* Egy képességhez közösen tartozó tulajdonságokat tárol, tehát több
képesség
* esetén ezek a tulajdonságokat érvényesek mindegyikre
*/
```

```
typedef struct Spell {
Cooldown cooldown;
const double range; /**< A képesség hatótávja */

double speed; /**< A képesség sebességét tárolja */

SDL_Texture *texture; /**< a képesség textúrájára mutató pointer */
Size imgsize; /**< A megjelenítendő kép méretét tárolja */
} Spell;

/**
* @struct Missile
* @brief A játékos egy képességét, a lövedéket tároló struktúra
* Egy adott lövedék tulajdonságait tárolja
*/
typedef struct Missile {
Point position; /**< A lövedék pozíciója */
Vector2 direction; /**< A lövedék iránya normalizált vektor alakban */
double angle; /**< A lövedék képét milyen szögben kell elforgatni (az x
tengelytől jobbra lefele pozitív) */
double distancetraveled; /**< A megtett távolságát tárolja, mert ha eléri a
hatótávját akkor megsemmisül */
} Missile;
/**
* @struct MissileNode
* @brief A lövedékeket tároló láncolt lista egy elemét leíró struktúra
*/
typedef struct MissileNode {
Missile missile; /**< maga a lövedék adatai */
struct MissileNode *next; /**< a következő lövedékelemre mutató pointer*/
} MissileNode;
typedef struct GameOjbect {
Point position; /**< Az objetkum pozíciója */
Vector2 direction; /**< Az objetkum iránya ahova tart, 1 hosszúságú
vektor*/
int hitboxradius; /**< Az objetkum köré rajzolt kör sugarának nagysága.
Ütközésvizsgálat esetén ez számít az objektum
körvonalának */
double speed; /**< A objetkum sebessége */
Size imgsize; /**< A objetkum képének nagysága */
SDL_Texture *texture; /**< A objetkum textúrájára mutató pointer */
} GameObject;
/**
* @struct Player
* @brief A játékost és az ahhoz tartozó összes adatot tároló struktúra
*/
typedef struct Player {
GameObject character; /**< a játékos alapadait tartalmazza*/
Point destination; /**< Az a pont ahova a játékos tart */
```

```
Spell flash; /**< A játékos villanás képessége */
Spell missileprops; /**< A játékos lövedék képességéhez tartozó adatok,
amelyek mindegyik lövedékre egyaránt érvényesek */
MissileNode *missiles; /**< A játékos lövedékeinek láncolt listája */
} Player;
/**
* @struct Entity
* @brief Az egyes ellenségek vagy tűzgolyók adatainak tárolására levő
struktúra
*/
typedef struct EntityNode {
GameObject entity; /**< az entitás alapadatait tartalmazza*/
bool followplayer; /**< Ez a változó tartalmazza, hogy minden egyes
mozgatásnál a játékos irányába mozogjon, vagy ne*/
struct EntityNode *next; /**< A következő entitáselemre mutató pointer*/
} EntityNode;
/**
* @struct SpawnProps
* @brief Az egyes entitások létrehozásának tulajdonságait tároló struktúra
* Előírja, hogy milyen körülmények között jönnek létre ezek az entitások
*/
typedef struct SpawnProps {
int rate; /**< Az a mérték, hogy milyen gyorsan jönnek létre az entitások
(frissítés gyakorisága ms-ben * rate = milyen gyakran jönnek
létre entitások)*/
int lowerlimit; /**< Az alsó határa annak, hogy milyen gyorsan jönnek létre
entitások*/
int counter; /**< Minden frissítési ciklusban itt tárolódik, hogy hol
tartunk,
mindig az incrementer értékével növekszik*/
int incrementer; /**< Ezt a változót adódik hozzá a counter-hez, az érték
növelésével lehet gyorsítani az entitások létrehozását*/
double initspeed; /**< az alap sebessége egy entitásnak amikor létrejön*/
} SpawnProps;
```

4.4 game

A játék modul csupán az jobbegérgomb lenyomás visszajelzéséhez szükséges struktúrát tartalmazza.

```
/**
* @brief tárolja azokat az adatokat, amelyek szükségesek ahhoz hogy a
* játékosnak visszajelezze azt ahova kattintott
*
*/
typedef struct showposclickfeedback {
int counter;
int limit;
bool show;
```

```
Point pos;
} showposclickfeedback;
```

4.5 macros

Ebben a fájlban csak két makró található meg, amik az ablakméretet mutatják.

```
#define WINDOWWIDTH 1280
#define WINDOWHEIGHT 960
```

4.6 scoreboard

Egy adott pontszámot eltároló struktúra:

```
typedef struct Score {

double points; /**< Az adott pontszám*/
char playername[50 + 1]; /**< A játékos neve (maximum 50 karakter hosszú)*/
} Score;
```

Az eredményeket tartalmazó láncolt lista egy elemét leíró struktúra

```
typedef struct ScoreNode {
Score score; /**< az elért eredmény*/
struct ScoreNode *prev, *next; /**< a lista következő eleme*/
} ScoreNode;
```

4.7 render

A render modulban struktúrák nincsenek létrehozva, csupán a globálisan használt színek vannak megosztva a többi modullal. Ezek a színek a következők:

```
extern SDL_Color c_white;
extern SDL_Color c_green;
extern SDL_Color c_red;
extern SDL_Color c_menubg;
extern SDL_Color c_btbg;
extern SDL_Color c_btbghover;
extern SDL_Color c_btselected;
```

4.8 overlay

Ez a modul nem tartalmaz adatszerkezetek deklarációját

4.9 menu

Ez a modul csak belső használatra lévő Button struktúrát írja le, ami egy kirajzolandó gomb kinézetét tárolja el, illetve a gombhoz tartozó adatokat.

```
typedef struct Button {
Point pos; /**< a gomb pozíciója*/
char text[30]; /**< a gomb szövege*/
void (*onclick)(struct Button *bt); /**< a gombra való kattintáskor ez a
függvény hívódik meg*/
bool selected; /**< ha a felhasználó utoljára erre a gombra kattintott
akkor igaz (csak azoknál a gomboknál van beállítva amelynél számít
(nehézségkiválasztás gombjai))*/
SDL_Color bgcolor; /**< a gomb háttérszíne*/
} Button;
```

5. Modulok és függvényleírások

A programban lévő modulok rövid leírása, illetve a modulokban lévő függvények leírása:

A függvényleírásnál első behúzásnál van a függvény deklarációja, alatta az első bekezdés az, hogy mi a feladat, azután három behúzásra vannak a függvény paraméterei, majd végül két behúzásra a visszatérési értéke. Ha nincs visszatérési értéke, vagyis void, akkor nincs odaírva semmi a paraméterek után. Ha rövid a függvény, akkor össze lehet vonni a visszatérési értéket és a magyarázatát.

5.1 main:

Ez a modul a lényegi inicializáló műveletek lebonyolításáért felel, többek között:

- Az ablak létrehozása
- Az srand-nak kezdeti értéket ad
- Inicializálja a betűtípust
- Betölti a dicsőséglistát
- Beállítja a program alapértelmezett állapotait és a nehézséget

Ezek után pedig elindítja a fő ciklust, amely addig fut, amíg az állapot QUIT-re nem vált.A ciklusban lévő kód csak bizonyos időközönként fut le, így egységesen minden gépen ugyanannyiszor fut le a ciklus belső része. Ezáltal a processzor is kevésbé van terhelve, és a legtöbb számítógépen egységesen fog futni a program, egészen addig amíg elég gyors az adott számítógép. Ha a játék véget ért akkor pedig elvégez pár műveletet a bezárás előtt. Ezek a következők:

- dicsőséglista elmentése
- a dicsőséglistát tároló lista felszabadítása
- a betűtípust tároló változó felszabadítása
- legvégül az SDL_Quit() függvényt hívja meg

5.2 math_helper

Itt találhatóak a különböző matematikai segédfüggvények, a matematikához kötődő struktúrák, illetve a matematikai műveletekhez köthető függvények.

- `Point randomspawnpoint();`
    - Létrehoz a pálya szélein (egy picivel beljebb) egy véletlenszerűen generált koordinátát
    - visszatérési értéke: a generált pont

- `Point gettopleftpoint(Point pos, Size size);`
  - a pos középpontú és size méretű téglalap bal felső koordinátáját adja vissza
  - pos a téglalap középpontja
  - size a téglalap mérete
  - egy Point változóban a bal felső koordinátát adja vissza
- `Point addvectortopoint(Point p, Vector2 v);`
  - hozzáad egy ponthoz egy vektort
  - paraméterek:
    - p a pont
    - v a vektor
  - az így kapott pontot adja vissza
- `Point rectdownrightpoint(Rect rect);`
  - a rect-ként téglalap jobb alsó koordinátáját adja vissza
  - paraméter:
    - rect a téglalap
  - a jobb alsó koordinátát adja vissza pontként
- `Vector2 vectorfromtwopoints(Point start, Point end);`
  - Két pontból egy vektort csinál.
    - start A kezdőpont koordinátája
    - end A végpont koordinátája
  - egy Vector2 struktúrában a két pont által kapott vektort adja vissza
- `Vector2 normalizevector(Vector2 v);`
  - Normalizálja a vektort, vagyis megtartja az irányát, de a hosszúságát egyre állítja be
  - paraméterek:
    - v a vektor amit normálizálni szeretnénk
  - egy Vector2 struktúrában a kapott egy hosszúságú vektor adja vissza
- `bool outofscreen(Point pos, Size size);`
  - megnézi, hogy a kapott pos középpontú és size méretű téglalap a képernyőn kívül van-e
    - pos a téglalap középpontja
    - size a téglalap mérete
  - ha a képernyőn kívül van, akkor igazzal tér vissza, ha nem akkor meg hamissal
- `bool withinbounds(Rect r, Point p);`
  - Megnézi, hogy az adott téglalapban benne van-e a pont
    - r a téglalap
    - p a pont
  - boolean ha benne van a pont akkor igazzal tér vissza, ha nincs benne, akkor hamissal
- `int twopointsdistance(Point p1, Point p2);`
  - Kiszámolja a paraméterként kapott két pont távolságát és visszaadja annak egész típusú értékét
    - p1 első pont
    - p2 második pont
  - a két pont távolsága
- `int vectorlength(Vector2 v);`
  - Megadja, hogy az adott vektornak mekkora a hossza
    - v vektor
  - a vektor hossza egész típusú változóként

- `double getangle(Vector2 v);`
  - Az sdl más formátumban kéri a forgatáshoz szükséges szöget, ez a függvény ezt az új szöget számolja ki
    - v a vektor
  - visszaadja valós típusú értékben, hogy sdl-nek hogyan kell megadni a forgatás értékét
- `Vector2 rotatevectorbyangle(Vector2 v, double angle);`
  - a kapott vektort elfordítja adott szöggel
    - v a forgatni kívánt vektor
    - angle a szög
  - Vector2 az elforgatott vektor

5.3 render

Ez a modul felelős a megjelenítésért, itt van eltárolva az ablak és a megjelenítő, illetve a betűtípus változója is.

- `void createwindow(Size windowsize, char *title);`
  - Inicializálja az ablakot és a megjelenítőt. Ha hibába ütközik, akkor kiírja a konzolra és kilép a program. Az infoc oldalról származik a kódrészlet.
    - windowsize Ekkora méretű lesz a létrehozott ablak
    - title Ez lesz az ablak fejlécére írva
- `void initfont(char *fonttype, int size);`
  - Betölti az adott betűtípust. A program végén meg kell hívni a closefont függvényt.
    - pathname a betűtípushoz vezető útvonal
    - size a betűtípus mérete
- `void closefont();`
  - Bezárja a betöltött betűtípust
- `SDL_Texture *loadimage(char *pathname);`
  - Betölti egy texturúba a képet. Ha nem sikerült betölteni a képet akkor kiírja a hibát a konzolra és bezáródik a program.
    - pathname Ezen az útvonalon lévő képet tölti be
  - SDL_Texture* A betöltött texturára mutató pointer adja vissza értékül
- `void renderbox(Point topleft, Point downright, SDL_Color color);`
  - Egy dobozt jelenít meg a kijelzőn
    - topleft A doboz bal felső sarkának koordinátája
    - downright A doboz jobb alsó sarkának koordinátája
    - color A doboz belsejének a színe
- `void renderrectangle(SDL_Texture *t, Rect dest);`
  - Egy téglalapban texturát jelenít meg a kijelzőn
    - t A textúra
    - dest A téglalap
    - rotation Az adott szög
- `void renderrectanglerotated(SDL_Texture *t, Rect dest, double rotation);`
  - Egy téglalapban texturát jelenít meg a kijelzőn, de az x tengelytől óramutató járásával megegyező irányban adott szöggel
    - t A textúra
    - dest A téglalap
    - rotation Az adott szög

- `void rendertext(Point pos, SDL_Color color, char *text);`
  - Egy adott pozícióba adott színű szöveget ír ki
    - pos A szöveg pozíciója
    - color A szöveg színe
    - text Maga a kiírandó szöveg
- `void rendercircle(Point p, int radius, SDL_Color c);`
  - Kirajzol egy kört a kijelzőre
    - p A kör középpontja
    - radius A kör sugara
    - c A kör körvonalának színe
- `void renderupdate();`
  - Frissíti a kijelző jelenlegi állását
- `bool input_text(char *dest, size_t size, SDL_Rect rect, SDL_Color bgcolor, SDL_Color textcolor);`
  - Beolvas egy szoveget a billentyuzetrol.A rajzolashoz hasznalt font es a megjelenito az utolso parameterek.Az elso a tomb, ahova a beolvasott szoveg kerul.A masodik a maximális hossz, ami beolvasható.A visszatérési értéke logikai igaz, ha sikerult a beolvasas. Az infoc oldalról származik a kódrészlet.

5.4 state

- `MainState getmainstate();`
  - Visszaadja a főállapotának jelenlegi értékét.
- `MenuState getmenustate();`
  - Visszaadja a menü alállapotának jelenlegi értékét
- `Difficulty getdifficulty();`
  - Visszaadja a játék jelenleg beállított nehézségét
- `void setmainstate(MainState state);`
  - Beállítja a paraméterként kapott állapotra a főállapotot
- `void setsubmenustate(MenuState s);`
  - Beállítja a paraméterként kapott állapotra az almenüállapotot
- `void setdifficulty(Difficulty diff);`
  - Beállítja a program nehézségét a paraméterként kapott nehézségértékre

5.5 entity

- `void moveplayer(Player *player);`
  - A játékos mozgatását végző függvény. Frissíti a játékos pozícióját a sebessége és az iránya alapján, emellett a render modul segítségével kirajzoltatja az új pozíción lévő karaktert.
    - a játékos struktúrára mutató pointer
- `void playerflash(Player *player);`
  - Végrehajta a villanás képességet. Az egér irányába, egy bizonyos távolságon belülre teleportálja a karaktert.
    - a játékos struktúrára mutató pointer
- `EntityNode *moveentities(EntityNode *entities, bool rotatedimage);`
  - Megváltoztatja az összes entitás pozícióját.

        - entities entitásokat tartalmazó láncolt lista
        - rotatedimage el legyen-e forgatva a megjelenített kép?
    - EntityNode* a láncolt lista elejére mutató pointert adja vissza (el kell tárolni az értékét, és felszabadítani később a hívónak)
- `void entitychangedir(EntityNode *entities, Point playerpos);`
    - Megváltoztatja az összes entitás irányát a játékos irányába
        - entities a láncolt lista elejére mutató pointert adja vissza (el kell tárolni az értékét, és felszabadítani később a hívónak)
        - playerpos a játékos karakterének jelenlegi pozíciója
- `EntityNode *spawnentity(EntityNode *list, Point playerpos, GameObject props);`
    - Létrehoz egy új entitást a megadott tulajdonságok alapján egy véletlenszerűen generált pozícióba. Az entitás a játékos irányába indul el. Ezt az entitást hozzáfűzi az entitásokat tartalmazó láncolt lista elejére.
        - List az entites-t tartalmazó láncolt lista.
        - playerpos a játékos
        - *props az új entitás alaptulajdonságait tartalmazó gameobject, a következő változókat kell benne definiálni: speed, texture, imgsize, hitboxradius
    - EntityNode* a láncolt lista elejére mutató pointert adja vissza (el kell tárolni az értékét, és felszabadítani később a hívónak)
- `void updatespellcooldown(Spell *spell, int ms);`
    - frissíti a képesség töltési idejét a megadott milliszekundumot figyelembe véve
        - spell a játékos képességére mutató pointer
        - ms az a miliszekundumérték, amely közönként lefut a userevent
- `void freeentities(EntityNode *entities);`
    - Felszabadítja az entitások tartalmazó láncolt listát. A program végén meg kell hívni.
        - entities a felszabadítandó lista elejére mutató pointer
- `MissileNode *spawnmissile(Player *player);`
    - Létrehoz egy lövedéket. A kezdőpozíciója a lövedéknek a játékos, és az egér irányába indul el
        - player a játékosra mutató pointer (inicializálni kell a lövedékek listát)
    - MissileNode* a láncolt lista elejére mutató pointert adja vissza (el kell tárolni az értékét, és felszabadítani később a hívónak)
- `MissileNode *movemissiles(Player *player);`
    - Mozgatja a lövedékeket. Mozgatja a lövedékeket, és ha megtették a maximálisan megtehető útat, akkor felszabadítja és kitörli őket a listából.
        - player a játékosra mutató pointer
    - MissileNode* a láncolt lista elejére mutató pointert adja vissza (el kell tárolni az értékét, és felszabadítani később a hívónak)
- `void freemissiles(Player *player);`
    - Felszabadítja a lövedékeket. A program végén meg kell hívni.
        - player a játékosra mutató pointer (inicializálni kell a lövedékek listát)
- `bool checkcollisioncircles(Player *player, EntityNode *entities);`
    - Megnézi, hogy a játékos ütközött-e valamilyen entitással
        - player a játékosra mutató pointer
        - entities az entitás láncolt listára mutató pointer

- boolean igaz értékkel tér vissza, ha ütközött egy entitással, és hamis értékkel, ha egyikkel sem ütközött

- `void checkcollisionmissileenemy(Player *player, EntityNode **enemies);`
  - Megnézi, hogy ütköztek-e a játékos lövedékei az entitásokkal ha ütköztek, akkor az adott lövedék és entitás megsemmisíti egymást, tehát felszabadítódnak és kitörlődnek a láncolt listájukból.
    - player a játékosra mutató pointer
    - enemies az entitás láncolt listára mutató pointer
- `void setspeedbydiff(SpawnProps *p, double basespeed);`
  - Ez a függvény beállítja, hogy milyen sebessége legyen egyes nehézségi fokozatok mellett a spawnprops struktúrának
    - p az adott spawnprops struktúrára mutató pointer, amelynek a sebesség változója fog megváltozni
  - basespeed az alapsebességérték
- `bool updatespawnprops(SpawnProps *p);`
  - Frissíti az adott spawnprops struktúra számlálóját ha a számláló elérte a rate (gyakoriság) változó értékét, akkor alaphelyzetbe állítja a számlálót, és a rate változót csökkenti egyel az előbbi esetben, mivel a számláló elérte a kívánt értéket, ezért igazzal tér vissza, vagyis létre lehet hozni egy új entitást
    - p a spawnprops struktúrára mutató pointer
  - boolean ha a számláló elszámolt a rate változóig, akkor igazzal tér vissza, ha még nem akkor hamissal

5.6 game

Ennek a modulnak csak egy függvénye van. A játékban egy kör lefutásáért felelős modul. Miután véget ért egy kör, átállítja a program állapotat menüre, vagy kilépésre Az összes dinamikusan foglalt memóriaterületet felszabadítja magától. A főbb csomópont a megjelenítő és az adatművelettel foglalkozó modulok között.

- `void game();`
  - Levezényel egy kört a játékon belül létrehozza a pályát, játékost, és ellenségeket, majd a render modult használva megjeleníti őket. Kiértékeli az entitások mozgását matematikai számításokkal, és érzékeli, hogy űtközött-e a játékos ellenséggel, mert ha igen, akkor a kör véget ér, és visszatér a függvény.

5.7 scoreboard

- `void loadscoresfromfile(char *filename);`
  - Betölti a dicsőséglista állását a fájlból ha nincs ilyen fájl, akkor létrehozza azt
    - filename a fájl elérési útvonala
- `void savescoreboardtofile(char *filename);`
  - Elmenti a dicsőséglista állását az adott fájlba
    - filename a fájl elérési útvonala
- `ScoreNode *getscores();`
  - Visszaadja a modul által tárolt láncolt lista első elemére mutató pointert
    - ScoreNode* a láncolt lista
- `double getcurrentpoint();`

- Visszaadja a jelenlegi körben elért pontszámot, ami egy valós típusú érték

- `void incrementcurrentscore(int point);`
  - Növeli a modul által tárolt pontszámot a nehézség szerint
    - point pontszám ami alapján növeli a jelenleg elért pontszámot.
- `void resetcurrentpoint();`
  - Alaphelyzetbe állítja a modul által tárolt pontszámot
- `void insertnewscore(char *name);`
  - A modul által tárolt listába beszúrja az új eredményt
    - name Ehhez a névhez fog tartozni a körben elért pontszám
- `void freescoreboard();`
  - Felszabadítja a modul által tárolt dicsőséglistát (ami egy láncolt lista). A program végén meg kell hívni

5.8 overlay

A játéknézetben megjelenő felhasználói felület, a játékélményt elősegítő jelzők és szövegek megjelenítéséért felelős modul.

- `void showseconds(double seconds);`
  - A jobb felső sarokban mutatja egy valós típusú változóban az eltelt másodperceket
    - seconds A kör kezdete óta eltelt másodpercek
- `void showpoints();`
  - Kimutatja a jelenlegi körben elért pontok számát a jobb felső sarokban, a másodpercszámláló alatt.
- `void showcooldowns(Player *player);`
  - A játékos két képességéhez tartozó töltési időt mutatja két kis négyzetben Ha zöld a négyzet, akkor használható a képesség, ha piros, akkor még nem.
    - player a játékosra mutató pointer