Program dokumentáció

Programozói dokumentáció:

A program 5 forrásfájlból épül fel, ezek a következők:

main.c **snake.c** **map.c** **filemanagement.c** **move.c**

A main.c a főprogram, melyben található a menü is, a snake.c a kígyó felépítéséért, a move.c az irányításáért felelős, a map.c a pálya megrajzolásához szükséges, a filemanagement.c pedig a fájlkezeléssel foglalkozik.

main.c:

A következő függvények találhatóak meg benne:

- void executeMenuItem (int menu)
- int main()

Az *executeMenuItem* nevű függvény végrehajtja a „switchcase" szerkezetet, mely lehetőséget biztosít arra, hogy a menüpontokat megnyissuk. Ebben a függvényben van deklarálva a *head, és a *secondhead változó, melyek az első illetve a második kígyó fejére mutató pointerek, ezek lesznek a későbbi lista első elemei, illetve az első rekordra mutató pointer. Paraméternek egy int változót vár, amelyet a menü kirajzolása után beütünk, ezt fogja felhasználni a „switchcase" szerkezet. Továbbá van benne még egy int game is, mely a második „switchcase" szerkezetben lévő választást valósítja meg.

A *main* függvényen belül egy while ciklus fut, ez azért szükséges, hogy csak abban az esetben lépjen ki a program, ha a felhasználó úgy dönt, hogy ki szeretne lépni.

map.c:

A következő függvények találhatóak benne:

- void clrScr()
- void gotoxy (int x, int y)
- void map1(int size)
- void map2(int edge)
- void drawMenu(int* menu)
- void drawQuestion()
- void drawMap()
- void drawScore(int score)
- void drawMultiScore(int score)
- void checkmapSize ()

A map.c - ben továbbá található egy globális változó, az int mapSize, amelyet a pálya méretét kérdő függvényt követően adhatunk meg. A pálya méretéért felelős, a szélesség ennek harmada.

A *clrScr* függvény a képernyőtörlést valósítja meg, paraméterként nem vár semmit.

A *gotoxy* függvény által lehetőségem volt a képernyő bármely pontjába mozgatni a kurzort, emiatt kellett az elején a „ COORD coord = {0, 0}; ”-ot deklarálni, hiszen ez beállítja a kezdeti x és y koordinátákat nullára. A *gotoxy* függvény paraméternek két int változót vár, először az x koordinátáét, majd az y koordinátáét, ahova vinni szeretnénk a kurzort.

A *map1* függvény a pálya tetejét, illetve alját rajzolja meg rekurzívan. Paraméterként a mapSize változót várja.

A *map2* függvény a pálya széleit rajzolja meg, paraméterként ez is a mapSize változót várja.

A *drawQuestion* függvény felteszi a kérdést, hogy mekkora legyen a pálya, és bekéri a számot a pályaméretre mindaddig, amíg az 30 és 71 között nem lesz. Paramétere nincs.

A *drawMap* függvény meghívja a *map1* illetve *map2* függvényeket, és ezáltal kirajzolja a pályát.

A *drawScore* függvény a pontok kiírásáért felelős, a *drawMultiScore* pedig ugyanígy működik, csak eltolva megjeleníti a második játékos pontjait. Paraméterként mindketten egy int változót várnak, ami az adott játékos pontja.

A *drawMenu* a menü kirajzolásáért felelős, paraméternek egy int* változót vár, ami esedékesen a main.c-ben lévő menu változó. A választás miatt fontos, hogy pointer legyen, különben az értéke elveszne.

move.c:

A move.c-ben található 2 struktúra, az egyik a Coordinate, a másik a SnakeElement. A Coordinate struktúra tartalmaz két változót, egy x és egy y változót, melyek int típusúak. A SnakeElement struktúra pedig egy duplán láncolt lista, tartalmaz egy koordinátát, illetve az előtte és az utána lévő listaelemre mutat. Ebből fog felépülni a kígyó.

```
typedef struct Coordinate {
    int x;
    int y;
} Coordinate;

typedef struct SnakeElement {
    Coordinate cord;
    struct SnakeElement *next;
    struct SnakeElement *before;
} SnakeElement;
```

Továbbá tartalmazza az alábbi függvényeket:

- void changeSnakeElementExceptHead(SnakeElement *head)
- SnakeElement *moveSnake(SnakeElement *head, char direction)
- SnakeElement *moveSecondSnake (SnakeElement *secondhead, char direction)
- void Death (SnakeElement *head)
- void deathMultiplayer (SnakeElement *head, SnakeElement *secondhead)

A *changeSnakeElementExceptHead* függvény paraméterként a lista elejére mutató pointert, jelen esetben a kígyó fejére mutató pointert várja. Először az iterator végigmegy a listán, ezáltal tudjuk, hogy hol a vége. Ezt követően az iteratorBack az utolsó listaelem lesz, és visszafelé indul a listán, értékül adva az utolsó koordinátáját az előzőnek. Visszatérési értéke void.

A *moveSnake* függvény paraméternek a lista elejére mutató pointert, illetve az irányt várja, amely irányban mozdítani szeretnénk a kígyónkat. A visszatérési értéke SnakeElement*, hiszen a végén visszaadjuk az újonnan kapott első listaelemre mutató pointert. A függvény „switchcase” szerkezettel épül fel. Ha a kígyó a pálya keretein belül mozog, akkor a függvény minden egyes utasításra meghívja a *changeSnakeElement* függvényt, s ezek után a fej koordinátáját megváltoztatja az adott iránynak megfelelően. Ha a kígyó a pálya kereteit túl szeretné lépni, a gamOver nevezetű, bool típusú változó igazra értékelődik ki.

A *moveSecondSnake* függvény működése teljesen azonos a *moveSnake* függvénnyel, azzal az eltéréssel, hogy az előre definiált változókat használja, úgy mint az UP, DOWN, stb., és if szerkezetekből lett felépítve.

A *Death* függvény a kígyó haláláért felelős. Paraméterként kapja a listaelem elejére mutató pointert. A függvényben deklarált iterator végigmegy a listán, és ha a fej és a test többi részének koordinátái között egyezést talál a gameOver változót igaz értékre állítja. Visszatérési értéke nincsen.

A *deathMultiPlayer* is ugyanolyan elven alapul, mint a *Death*, csak itt két iterator van, az egyik végigmegy az egyik listán, másik a másik listán, és ha a lista elejére mutató pointer a saját kígyóval, a másik kígyóval, vagy egymás fejével ütközik, a gameOver változót igazra állítja. Visszatérési értéke ennek sincsen.

snake.c

A snake.c elején van deklarálva egy gameOver nevű, bool típusú változó. Ez dönt a későbbiekben arról, hogy a kígyó ütközött-e fallal, önmagával, esetleg egy másik kígyóval.

Továbbá megtalálhatóak benne az alábbi függvények:

- void drawFood (Coordinate cord)
- Coordinate generateNewFoodPlace()
- SnakeElement *addSnakeElementToList(SnakeElement *head, Coordinate cord)
- SnakeElement *buildStarterSnake(SnakeElement *head)
- SnakeElement *buildStarterSecondSnake(SnakeElement *secondhead)
- void drawSnake(SnakeElement *head)
- void drawSecondSnake(SnakeElement *secondhead)
- void singlePlayer (SnakeElement *head)
- void MultiPlayer(SnakeElement *head, SnakeElement *secondhead)
- void freeSnakeList(SnakeElement *first)

A függvény elején definiálva vannak a nyílbillentyűk értékei, illetve a STARTER_SNAKE_SIZE_WITH_HEAD, ez megadja, hogy kezdetben a kígyó a fejből, és 3 testelemből fog állni.

A *drawFood* függvény paraméterként egy koordinátát vár, amely x és y koordinátájába kirajzol egy „o”-t, amely az étel a kígyó számára.

A *generateNewFoodPlace* függvény nem kap semmit paraméternek, visszatérési értéke pedig Coordinate, hiszen valamely új koordinátába fog generálni ételt. A függvény generál egy x és egy y koordinátát a pálya keretein belül, azonban ha azok éppen a keretek, akkor a szükséges mennyiségű koordinátát hozzáadja, illetve kivonja.

Az *addNewSnakeElement* függvény paraméterként a lista elejére mutató pointert és egy koordinátát vár, visszatérési értéke pedig SnakeElement*, hiszen visszatér majd a megváltozott listával. A függvényen belül deklarálva van egy newElement listaelem, melynek foglalva van egy listaelemnyi terület dinamikusan. A newElement koordinátájának beállítja a kapott koordinátát, és beállítja NULL pointerre az őutána következő elemet. Ezt a newElement-et hozzáfűzi a lista elejére mutató pointerhez, hogyha az nem NULL, mert ha igen, akkor visszatér az új elemmel.

A *buildStarterSnake* függvény paraméternek a listaelem elejére mutató pointert várja, visszatérési értéke a listaelem. A kezdeti 4:4 koordinátában elkezdi felépíteni a kígyót, hozzáfűzve folyamatosan egy listaelemet az *addNewSnakeElement* függvénnyel az x koordinátát növelve, addig, amíg a STARTER_SNAKE_SIZE_WITH_HEAD értékét el nem éri a for ciklus.

A *buildStarterSecondSnake* függvény hasonlóan jár el, mint a *buildStarterSnake*, csak nem a 4:4 koordinátával kezdi, hanem a 8:7 koordinátával.

A *drawSnake* függvény iteratora végigmegy a listán, a fej koordinátájában kirajzolja a fejet, a többiben pedig a testet.

A *drawSecondSnake* ugyan az mint a *drawSnake*, csak a testet más karakterekkel rajzolja ki.

A *singlePlayer* függvény az egyjátékos módért felel. Visszatérési értéke nincs, paraméterként pedig a listaelem elejére mutató pointert, a head-et kéri. A függvényen belül deklarálva van egy foodCord koordináta, ami 3,3 értékre van állítva. Ez lesz a játék elején a legelső étel, amit a kígyónak meg kell ennie. Továbbá deklarálva van egy snakeFoodCounter nevű változó, ami egész típusú, ez fogja számolni, hogy mennyi ételt evett meg a kígyó, van egy score egész típusú változó, ami a pontokat számolja, illetve van egy karaktertömb, ami a játékos nevét fogja eltárolni. A függvény egy while ciklusban van felépítve, hiszen ekkor fog a kígyó folyamatosan mozogni, hisz ez mindig igazra értékelődik ki. Gombnyomásra pedig elindul a játék, az irány mindig a lenyomott billentyű lesz. Van benne egy if(gameOver) szerkezet, ami arra vonatkozik, hogyha a gameOver változó kiértékelődik 1-re, tehát meghalt a kígyó, akkor törli a képernyőt, felszabadítja a listát, bekér egy nevet és eltárolja egy külön .txt fájlba. Van még egy if szerkezet, ez arra vonatkozik, ha a kígyó feje elérné az ételt, akkor a snakeFoodCountert növeli egyel, és ha elért 5-ig, akkor növeli a kígyó méretét egy lista hozzáadásával. Ha a fej egy étel koordinátájához ért, a foodCord megkap egy újonnan generált ételkoordinátát, és a pontszám nő 10-zel. Ezután kirajzolódik a kígyó, megnézi, hogy meghalt-e, kirajzolódik az étel, a pontszám, és van benne egy 100 milliszekundumos késleltetés, amely a lassabb játékmenet végett van belerakva, majd törli a képernyőt és újrarajzolja az egészet.

A *freeSnakeList* függvény paraméternek kapja a lista első elemét és felszabadítja a listát. Visszatérési értéke void típusú.

A *MultiPlayer* függvény hasonló a *singlePlayer* függvényhez, azonban annyi a különbség, hogy paraméterként mindkét lista elejére mutató pointert vár, ezenfelül minden duplán szerepel benne. A pont, a számláló, illetve a név is. Az mozgás úgy van megoldva, hogy egy ideiglenes tempDirection változóba bekér egy billentyűleütést, ekkor egy „switchcase” szerkezeten belül megnézi, hogy ha az első játékos irányítására szolgáló gomb nyomódott le, akkor a direction1 változónak adja értékül az ideiglenes irányváltozót, ha pedig a második játékos kígyójának irányítására szolgáló gomb nyomódik le, akkor a direction2 változónak adja értékül az ideiglenes változó.

filemanagement.c

A filemanagement.c a fájlkezelésre szolgál. Az elején van egy struktúra, amely egy láncolt lista a Rekord elemeknek. Van benne egy karaktertömbre mutató pointer, illetve egy legjobb pontszám nevű egész változó, s továbbá a következő listaelemre mutató pointer.

```
typedef struct RecordElement {
    char *name;
    int bestScore;
    struct RecordElement *next;
} RecordElement;
```

A következő függvények találhatóak benne:

- RecordElement *addNewRecord(RecordElement *firstRecord, RecordElement *newRecord)
- RecordElement *sortRecordListElement(RecordElement *firstRecord, int listSize)
- void printRecords(RecordElement *firstRecord)
- RecordElement *readRecords(RecordElement *firstRecord)
- void saveRecords(int score, char *name)
- void freeRecordList(RecordElement *firstRecord)

Az *addNewRecord* függvény visszatérési értéke a lista struktúra, hiszen egy új listaelemet hozzáfűz a listához, majd visszatér az első rekorddal. Paraméterként várja a legelső listaelemet és az új rekordot, amit hozzá szeretnénk fűzni. Ha az első rekord NULL, akkor az új rekorddal tér vissza, egyébként pedig a lista végére fűzi.

A *sortRecordListElement* függvény visszatérési értéke a lista struktúra, paraméterként pedig az első rekordot kapja és a lista méretét. Ez a rendezés egy buborékrendezéses módszer. Létrehozunk egy iteratort és egy egész változót. Egy for cikluson belül megyünk végig az előbb létrehozott változótól a listahosszig. Ezen a for cikluson belül található még egy for ciklus, ami végigmegy a listán, és megnézi, hogy az adott listaelem pontszáma kisebb-e a következő listaelem pontszámánál, és ha igen, akkor fut bele az if szerkezetbe. Ezen belül deklarálva van egy egész típusú változó, amelyben elmentjük a következő listaelem pontszámát, létrehozunk egy ideiglenes névtároló pointert, aminek lefoglalunk dinamikusan memóriát, majd ennek értékül adjuk a következő listaelem név változóját.
A következő listaelem pontszámának értékül adjuk az aktuális pontszámát, majd ugyanezt csináljuk a névvel is, és ezután az aktuális listaelemnek értékül adjuk az ideiglenes változókat, a nevet és a pontszámot.

A *printRecords* függvény kiírja a listában található adatokat addig, amíg a függvényen belül deklarált számlálót beállítjuk. Jelen esetben a legjobb 10 eredményt írja ki.

A *saveRecords* függvény visszatérési értéke void, paraméterként vár egy pontszámot és egy karaktertömböt. Létrehoz egy record.txt nevű fájlt, amiben eltárolja a paraméterként kapott adatokat, úgy, hogy azt mindig a fájl végére hozzáfűzi.

A *readRecords* függvény visszatérési értéke a láncolt lista, paraméterként várja az első listaelemre mutató pointert. A függvényen belül létrehozunk egy karaktertömböt, egy számlálót és egy egész típusú számot.
Amíg van beolvasandó szöveg, addig hajtódik végre a ciklus, amelyben létrehoz egy új rekordelemet, és foglal neki dinamikusan memóriát. Az újrekord névnek szintén foglal dinamikusan memóriát, és átmásolja a beolvasott sztringet az újrekord nevéhez, majd a végére rak egy lezáró nullát. Hasonlóan cselekszik a pontszámmal is, és a számlálót mindig növeli egyel, ha hozzáadta a listához az új rekordot. Meghívja a rendezés függvényt aztán visszatér az listaelem elejére mutató pointerrel.

A *freeRecordList* függvény visszatérési értéke void, paraméterként a lista elejére mutató pointert várja, majd felszabadítja a listaelemeket.

Felhasználói dokumentáció

A program egy Snake nevezetű játék, amelyben a cél, hogy minél több ételt egyél meg a kígyóval úgy, hogy nem ütközöl falba, illetve saját magadba.

A program indításával egy menü fogadja a felhasználót, amelyben 3 opciója van a játékosnak:

1. Start Game
2. Records
3. Quit

Ha a felhasználó a 2-est választja, megnézheti a top10 rekordot, ha pedig a 3-asat, akkor a program leáll.

Ha az 1-es opciót választotta, felkínálkozik egy újabb menü:

1. Single player
2. Multiplayer

Mindkét opciót választva ugyanaz történik, megjelenik egy kérdés, hogy milyen nagyra szeretné készíteni a pályát. Az ideális méret 30 és 71 között van, ezen az intervallumon belül kell választania.

Single player opciót választva kirajzolódik a pálya és a kígyó. Karakterlenyomásra indul a játék, irányítani a W-előre, S-lefele, A-balra, D-jobbra lehetőséggel lehet. Bármely más billentyűleütéssel szüneteltetni lehet a játékot. A lényeg, hogy felvegye az „o” betűkből rajzolt ételt, és minél nagyobbra nőjön anélkül, hogy nekiütközne a falnak vagy önmagának.

Multiplayer opciót választva szintén kirajzolódnak a kígyók, és ez is karakterlenyomásra indul. Az egyesjátékos által irányított kígyó ugyanúgy mozog, mint a single player opcióban, a második játékos által irányított kígyó pedig a fel, le, jobbra, balra nyilakkal tud mozogni. A cél ugyanaz, azzal a kiegészítéssel, hogy itt egymásnak sem ütközhetnek a kígyók.

Halál esetén a program leáll és bekéri a játékos(ok) nevét, majd ezt elmenti egy fájlba, amit később a Menüből érhetnek el. Miután beírták a nevüket, megjelenik a rekordlista, és megtekinthetik az aktuális eredményük milyennek bizonyult.

Tesztelési dokumentáció

- Menüben rossz szám megadása: a program jelzi ✓
- Menüben betű beírása: a program kilép a fő menübe.✓
- Rossz pályaméret választása: a program nem engedi ✓
- A kígyó falnak, önmagának vagy másik kígyónak ütközik: meghal ✓
- Egyjátékos módban a kígyót szüneteltetni más gomb lenyomásával: működik ✓
- Étel megevése után új étel generálódik: működik ✓
- Bizonyos mennyiségű étel megevése után a kígyó nő: működik ✓
- A pontszámláló helyes eredményt mutat: működik ✓
- Mindkét kígyó irányítása megfelelő: működik ✓
- Mindkét kígyó meg tudja enni az ételt: működik ✓
- Mindkét játékos eredményét el lehet menteni: működik ✓
- Helyes sorrendben jeleníti meg az eredményeket: működik ✓
- Bármely pályaméret esetén működik a halál: működik ✓