Nagy házi feladat – Amőba játék
Programozói dokumentáció

Szükséges környezet
A program működéséhez szükség van az SDL grafikus könyvtárra, továbbá a szövegek megjelenítéséhez szükséges a LiberationSerif-Regular.ttf fájl.

Felépítés
A program a következő modulokból épül fel:

bot.c - bot.h
Ez a modul a gép által tett lépések meghatározására szolgáló függvényeket tartalmazza. A hozzá tartozó fejlécfájlban van definiálva a Lepes struktúra, továbbá az X_NYERT és az O_NYERT makrók.

grafika.c - grafika.h
Itt találhatók a játék grafikai megjelenítést segítő függvények.

jatek.c - jatek.h
Ez tartalmazza a játékot irányító függvényeket. A fejlécfájlban vannak definiálva a cella és játékállás felsorolt típusok és a játék struktúra.

main.c
Több SDL eseményhurok és a többi modulban található függvények segítségével megjeleníti a játékot.

memoria.c - memoria.h
Itt található a pálya lefoglalásához és felszabadításához használt függvény.

mentes.c - mentes.h
Ebben a modulban vannak a jelenlegi játékállás elmentését és visszatöltését lehetővé tevő függvények.

menu.c – menu.h
Az itt lévő függvények jelenítik meg a játék két menüjét.

Adatszerkezetek

Felsorolt típusok:

- ```
  typedef enum cella{
    ures,
     jatekos_x ,
     jatekos_o
  } cella;
  ```

  Ezt a három értéket tudja felvenni a pálya egy cellája. Ilyen típusú elemekből épül fel a pálya és ilyen típusú érték jelzi a játék struktúrában a következő játékost is.
  ```

-
  ```
  typedef enum jatekallas{
        folyamatban,
        x_nyert,
        o_nyert,
        egyenlo,
        kilepes
  }jatekallas;
  ```

  A játék lehetséges állapotai. Ilyen típusú a játék struktúrában az allapot.

Struktúrák

-
  ```
  typedef struct jatek{
     cella **palya;
     cella jatekos;
     jatekallas allapot;
  } jatek;
  ```

  A játék állásának leírásához szükséges adatokat tárolja, melyek a pálya, a most következő játékos és a játék jelenlegi állapota.

-
  ```
  typedef struct Lepes{
     int x;
     int y;
     int ertek;
  } Lepes;
  ```

  A gép lépéseinek meghatározásához szükséges. Eltárolja az adott lépés koordinátáit, és a lépéshez rendelt értéket, ami azt határozza meg, hogy a lépés mennyire előnyös egy adott játékosnak.

Függvények

jatek.c

- void jatekos_csere(jatek *j):
  a jelenlegi játékost átváltja a másikra
- bool n_egy_iranyban(jatek *pj,int jatekos,int meret,int kezdet_x,int kezdet_y,int irany_x,int irany_y, int n, int ures_vegek):
  ellenőrzi, hogy a játékosnak van-e egy adott irányban n darab egymás melletti bábuból álló sora aminek a végein legalább ures_vegek darab üres cella van. Az üres végek ellenőrzésére a gép lépéseinek meghatározásánál van szükség. Az ures_vegek paraméter 0, 1, vagy 2 értéket vehet fel.
- bool n_sor(jatek *pj, int jatekos, int meret, int n, int ures_vegek):
  minden irányra ellenőrzi, hogy van-e n darab egymás melletti bábuja a játékosnak, ures_vegek darab üres cellával a sor végein
- int cella_szamlal(cella **palya, cella keresett, const int meret):
  megszámlálja az adott típusú bábukat a pályán

- void jatek_vege(jatek *j, const int meret):
  a végeredmény alapján átállítja a játék állapotát
- void lepes(jatek *j, int sor, int oszlop, const int meret):
  a kiválasztott mezőre lerakja a játékos bábuját

grafika.c - grafika.h

- int cella_szelesseg(const int meret):
  kiszámolja egy cella szélességét
- int cella_magassag(const int meret):
  kiszámolja egy cella magasságát
- void sdl_init(int szeles, int magas, SDL_Window **pwindow, SDL_Renderer **prenderer):
  inicializálja az SDL-t
- void szoveg(SDL_Renderer *r, char *c, int betumeret, int x, int y):
  egy megadott helyre kiír egy megadott betűméretű szöveget
- void halo_kirajzol(SDL_Renderer *renderer, const SDL_Color *color, const int meret):
  kirajzolja a négyzetrácsos pályát
- void x_kirajzol(SDL_Renderer *renderer, int sor, int oszlop, const SDL_Color *color, const int meret):
  rajzol egy x-et a megadott helyre
- void o_kirajzol(SDL_Renderer *renderer, int sor, int oszlop, const SDL_Color *color, const int meret):
  rajzol egy o-t a megadott helyre
- void palya_kirajzol(SDL_Renderer *renderer, cella **palya, const SDL_Color *x_szin, const SDL_Color *o_szin, const int meret):
  a pályán kirajzolja a bábukat
- void kozben_kirajzol(SDL_Renderer *renderer, jatek *j, const int meret):
  játék közbeni állapot megjelenítése
- void vegen_kirajzol(SDL_Renderer *renderer, jatek *j, const SDL_Color *color, const int meret):
  játék végén megjelenő képernyő megjelenítése
- void jatek_kirajzol(SDL_Renderer *renderer, jatek *j, const int meret):
  a játék állapotától függően megjeleníti a megfelelő pályát

menu.c

- void fomenu_kirajzol(SDL_Renderer *renderer)
  megjeleníti a játék elején lévő főmenüt
- void uj_jatek_menu_kirajzol(SDL_Renderer *renderer)
  megjelenítí az új játék választása esetén megjelenő menüt

memoria.c

- cella** palya_lefoglal(int x)
  lefoglal egy x szélességű és magasságú pályához szükséges memóriát
- void palya_felszabadit(cella **palya, int x)
  felszabadítja a paraméterként kapott x nagyságú pályához tartozó memóriát

mentes.c

- bool mentes(jatek *j, char const *fajlnev, int meret)
  a megadott fájlba elmenti a játék jelenlegi állását
- bool betolt(jatek *j, char const *fajlnev, int meret)
  a megadott fájlból visszatölti a legutóbb elmentett játékállást

Példa egy mentés után keletkező fájlra:

```
6                ←a pálya mérete
1                ← játékmód (1=egyjátékos, 0=kétjátékos)
2 2 1 2 2 0
2 0 0 1 0 0
0 0 1 0 0 0
1 0 0 0 1 0      ← a pálya (2=O, 1=X, 0=ures)
0 0 0 0 0 0
0 0 0 0 0 0
1                ← következő játékos (1=X, 2=O)
0                ← játék állapota (0=folyamatban)
```

bot.c

- int kiertekel(jatek *j, int meret):
  kiértékeli a pálya jelenlegi állását, és visszaad egy értéket, ami megmutatja, hogy kinek kedvez az állás. Minél nagyobb a visszakapott érték, annál inkább az ”O” van jó helyzetben, ha pedig kisebb akkor pedig az ”X” . A kiértékeléshez azt vizsgálja meg, hogy melyik játékosnak milyen hosszú és hány üres véggel rendelkező sora van a pályán
- Lepes legjobb_lepes_kiertekelve(jatek *j, int meret):
  megkeresi azt az üres mezőt, amelyet választva a gép a legelőnyösebb helyzetbe kerül, az előnyösség mérésére a kiertekel függvényt felhasználva
- Lepes optimalis(jatek *j, int melyseg, int meret, cella jatekos, int vagas, bool legkulso):
  Visszaadja a gép számára legoptimálisabb lépést.
  A mélység paraméter azt szabályozza, hogy a függvényben lévő rekurzív hívás hányszor történjen meg. A vágás azt adja meg, hogy egy bizonyos érték alatt vagy felett lévő lépéseket ne vizsgáljon a függvény, ezzel gyorsítva a programot.
  Mikor a függvény rekurzívan hívja meg saját magát, akkor a visszatérési értékből csak az érték mezőt használja fel. Ebben az esetben a bool legkulso paramétert hamisra állítjuk, hogy ne töltsünk időt az x és y koordináták kiszámolásával. Mikor a függvényt a main.c-ből hívjuk meg, a legkulso paraméter értéke igaz, ezzel jelezve, hogy az x és y koordinátákat is használni fogjuk a visszatérési értékből.