ÓBUDAI EGYETEM
KANDÓ KÁLMÁN VILLAMOSMÉRNÖKI KAR
MŰSZERTECHNIKAI ÉS AUTOMATIZÁLÁSI INTÉZET

PROGRAMOZÁS I. A GYAKORLATBAN

ÉV VÉGI BEADANDÓ FELADAT

ULTIMATE TIC-TAC-TOE

Készítette:

Fodor András Benedek

G2NFHW

TARTALOMJEGYZÉK

SZOFTVER TERV

A kis táblák mentésére kétdimenziós tömb

```
unsigned char localBoards[9][9] = {0};
```

Lehetséges értékei:

- 0: Üres mező
- 1: X
- 2: O

A nagy tábla mentésére tömb (megnyert kis táblák tárolása)

```
unsigned char localBoards[9][9] = {0};
```

Lehetséges értékei:

- 0: Tábla „lokális üzemmódban” (tehát választható lépéshez)
- 1: X által megnyert tábla
- 2: O által megnyert tábla
- 3: Döntetlenre futott tábla

A kiválasztott tábla tárolására

```
unsigned char selectedBoard = 9;
```

Lehetséges értékek:

- 0-8: Kiválasztott tábla: Nem azonos a számozás a numerikus billentyűzettel: Jobbról balra, felülről lefele számozott.
- 9: Nincs kiválasztott tábla: Most kezdődött a játék, vagy az előző lépés alapján kijelölendő tábla érvénytelen.

Jelenlegi játékos tárolására

```
unsigned char player = 0;
```

Lehetséges értékek:

- 0: X
- 1: O

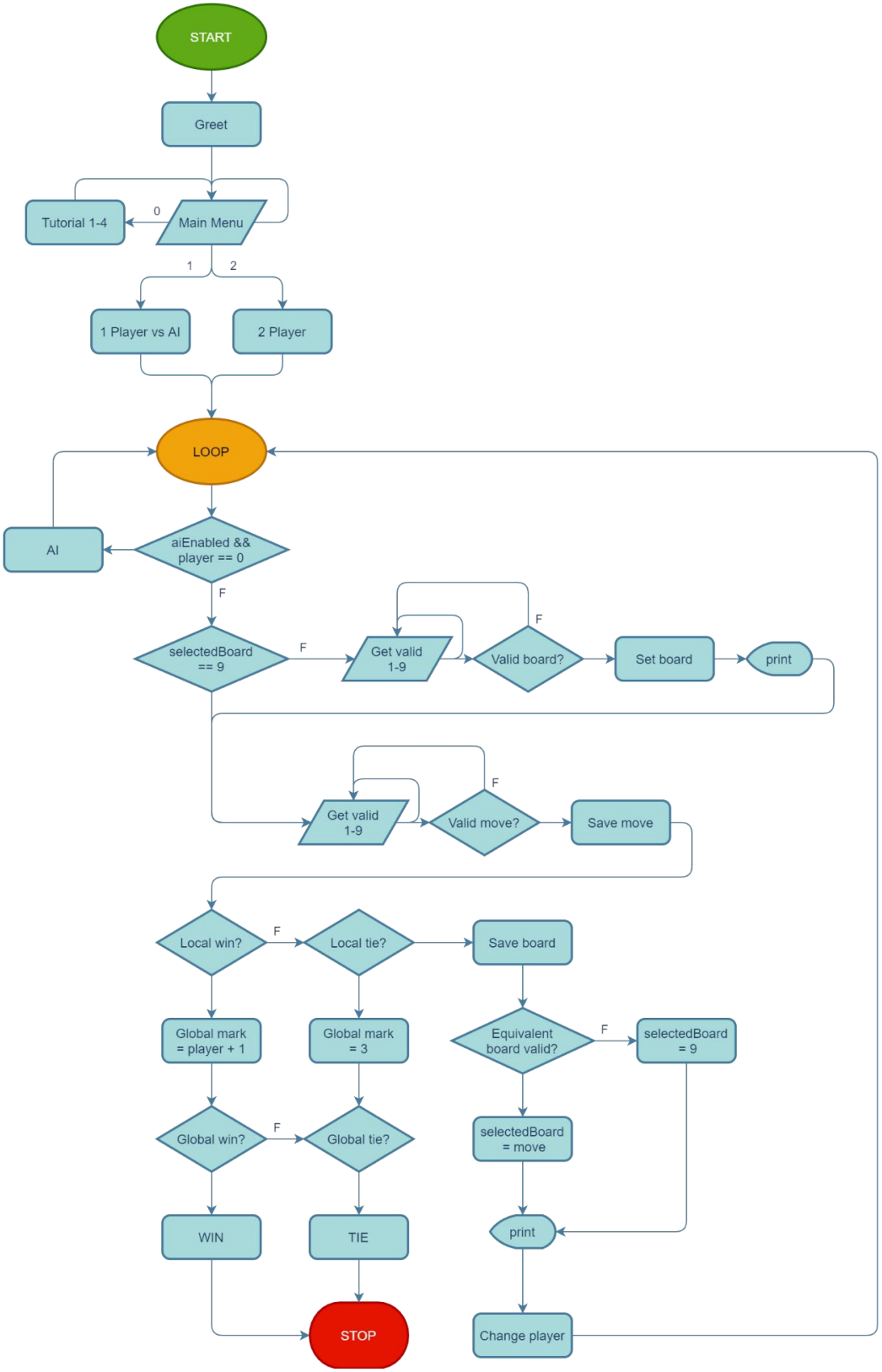
Modulok

- **main.c**
  A program fő struktúrája, legfontosabb függvények
- **utils.c**
  Több helyen is felhasznált, kisebb függvények
- **printGame.c, printGame.h**
  A játéktábla kirajzolásáért felelős függvények
- **globals.h**
  Globális változók és állandók
- **tutorial.h**
  A játékszabály-ismertető szövegét és a kiírásáért felelős függvényt tartalmazza

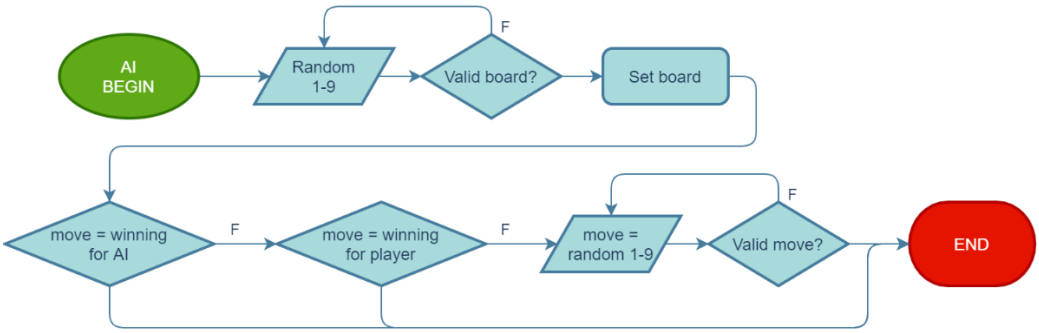
Függvények

main()

A program fő struktúráját tartalmazza, működése folyamatábrán:



Szintén a main függvény része az AI, azaz a gép lépéseit szabályzó algoritmus:



init()

Inicializálja a véletlenszám-generátort, valamint beállítja a konzolt széles karakterek fogadására. Erre a táblán használt táblázatrajzoló karakterek miatt volt szükség, ugyanis ezek nem részei az ASCII táblának. Így azonban az egész kódban széles karakterekkel kellett dolgoznom, a használt beépített függvényekre néhány példa:

- printf() → wprintf()
- _getch() → getwch()
- fprintf() → fwprintf()
- strcpy() → wcscpy()
- sprintf() → swprintf()

gameOver()

Kiírja a nyertest, vagy azt, hogy a játék döntetlennel végződött. Felkínálja a lehetőséget a végső játékállás fájlba mentésére.

saveGame()

Fájlba írja a végső játékállást.

getMove()

A játékostól lépést kér be. Ez használatos kis tábla választásnál, és X vagy O elhelyezésénél is. 1-9-ig fogad el karaktereket, akkor lép tovább, ha ilyet ütött be a játékos.

greet()

A főmenü működését tartalmazza. Bekéri, hogy 1 vagy 2 játékos játszik, kérésre megjeleníti az útmutatót.

printGame()

Kirajzolja a játékállást a képernyőre.

Ez a program legkomplexebb függvénye, amelynek teljes ismertetésébe nem bocsátkozom ebben a dokumentumban, lényegében:

Állandóból olvassa be a játéktáblát alkotó táblázatot, majd erre „szuperponálja” rá a kiválasztott tábla kettős keretét, valamint a táblák tartalmát.

Megírásakor a legnagyobb nehézséget (a „wide character”-ek kezelésén túl) az jelentette, hogy a kis táblánként tárolt adatstruktúrát hogyan alakíthatjuk át a következőképpen:

RENDER

	0	1	2	3	4	5	6	7	8
0	00	10	20	30	40	50	60	70	80
1	01	11	21	31	41	51	61	71	81
2	02	12	22	32	42	52	62	72	82
3	03	13	23	33	43	53	63	73	83
4	04	14	24	34	44	54	64	74	84
5	05	15	25	35	45	55	65	75	85
6	06	16	26	36	46	56	66	76	86
7	07	17	27	37	47	57	67	77	87
8	08	18	28	38	48	58	68	78	88

DATA

	0	1	2	3	4	5	6	7	8
0	00	01	02	10	11	12	20	21	22
1	03	04	05	13	14	15	23	24	25
2	06	07	08	16	17	18	26	27	28
3	30	31	32	40	41	42	50	51	52
4	33	34	35	43	44	45	53	54	55
5	36	37	38	46	47	48	56	57	58
6	60	61	62	70	71	72	80	81	82
7	63	64	65	73	74	75	83	84	85
8	66	67	68	76	77	78	86	87	88

A RENDER táblában a mezők két értéke (oszlop; sor),
míg a DATA táblában (lokális tábla száma; lokális tábla mezeje)

extendLines()

A bejövő értékként kapott helyen található string elé megadott számú sort tesz úgy, hogy ezek a sorok olyan szélességben ki legyenek töltve ’space’-ekkel, hogy az eredmény helyesen szuperponálható legyen a játéktáblára.

superpose()

Ez a függvény végzi a játéktábla különböző rétegeinek egymásra illesztését. Működése egyszerű, használata a program többi részében már kevésbé.

Három bemeneti értéke van, mindhárom egy string címe: kimenet helye, ráillesztendő réteg, alapréteg.
A ráillesztendő rétegből minden karaktert az alapréteg megfelelő helyére másol be, ezzel felülírva azt, kivéve, ha az ’space’.

getWinningFor()

Bemeneti értéke az a játékos, amelyik esetében szeretnénk olyan mezőt találni, amelyre lépve az adott táblát egy lépésből megnyerheti.
Az AI által használt függvény.

winCheck()

Bemeneti értéke egy tábla címe, ezen a táblán ellenőrzi, hogy a megadott játékos nyert-e. A bemeneti címtől függően a nagy táblán és kis táblákon is használható.

isDraw()

A bemeneti értékként kapott táblán ellenőrzi, hogy döntetlen-e a játék. Szintén működik a nagy táblán és kis táblákon is.

isValidBoard()

A bemeneti értékként kapott tábláról visszaadja, hogy kiválasztható-e következő lépés helyeként (egy tábla kiválasztható, ha még nem nyerte meg senki, és még nem telt be).

isValidMove()

A bemeneti értékként kapott lépésről visszaadja, hogy érvényes-e (egy lépés nem érvényes, ha a helyén már szerepel X vagy O).

MŰKÖDÉS ISMERTETÉSE

A program indításkor a főmenübe lép be. Itt a játékos megismerheti a játék szabályait, majd kiválaszthatja, hogy egyedül (a gép ellen), vagy párban egy másik játékos ellen szeretne játszani.

Mindig az X jelű játékos kezd, ha gép elleni játékot választottunk, a gép játszik O-val.

Lépni mindig a numerikus billentyűzet „vizuálisan megfelelő” gombjával lehet, tehát az egyes mezőkhöz tartozó gombok:

```
789
456
123
```

Lépések után mindig újra kirajzolja a program a játékállást, az előző állásokat nem törli. Szándékosan nem törlöm a parancssort a lépések után, ugyanis így (animációk híján) sokkal könnyebben megfigyelhetők a tábla változásai.

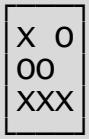
A játékszabályokat a következő fejezetben ismertetem.

Ha valamelyik játékos nyer, vagy döntetlennel végződik a játszma, van lehetőség a végső játékállás elmentésére egy fájlba.

JÁTÉKSZABÁLYOK

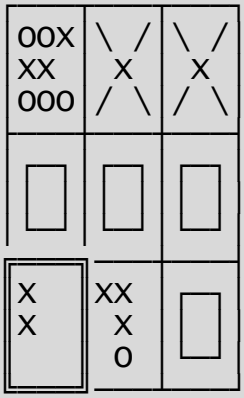
A Tic-Tac-Toe-ról

A Tic-Tac-Toe az amőba egy 3x3-as táblán játszódó változata: 3 jelet kell letenni a táblán egymás mellé, alá, vagy átlósan.
A következő átlagos Tic-Tac-Toe játékot az X nyerte meg:



Az Ultimate Tic-Tac-Toe

Az UTT nem csupán egy, hanem 3x3 ilyen táblán játszódik!
A következő UTTT játékot az O nyerte meg:

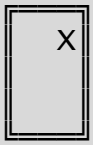


Ahogy láthatjuk, a teljes játék megnyeréséhez egy sorban, oszlopban vagy átlóban kell megnyerni 3 kis táblát.

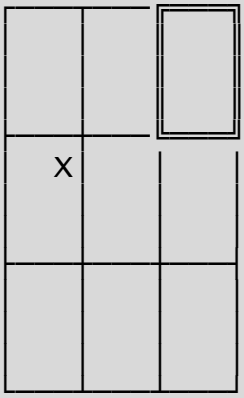
Csavar a lépésekben

Nem szabadon dönthetjük el, hogy melyik kis táblán szeretnénk lépni, hanem ezt az előző játékos lépése határozza meg.

Tegyük fel, hogy az előző játékos (X) az egyik kis táblán a jobb felső helyre lépett (most csak ezt az egy kis táblát figyeljük, itt mindegy, a nagy táblán ez hol helyezkedik el):



Mivel az X a jobb felső helyre lépett a kis táblán, a következő játékos a nagy tábla jobb felső kis tábláján léphet majd (a kiválasztott kis táblát duplavonal jelöli):



Persze, ha a kiválasztandó kis tábla már betelt, vagy valaki korábban megnyerte, akkor nem válhat kijelöltté; ebben az esetben a következő játékos szabadon választhatja ki a kis táblát, ahol lépni szeretne.

A játék célja

Megnyerni a kis táblákat egy oszlopban, sorban vagy átlóban, közben figyelve arra, hogy hova lépünk, mert ez határozza meg, hogy melyik táblán játszhat az ellenfelünk.

FEJLESZTÉSI LEHETŐSÉGEK

A játék bővíthető lenne egy pontozási rendszerrel, toplistával, akár online többjátékos móddal is; azonban már léteznek jól működő böngészővel játszhat verziói, ezért erre „piaci igény” valószínűleg nem lenne.

Lehetséges lenne továbbá egy ügyesebb AI megírása, a jelenlegi véletlenszerűen lép (két kivétellel: egy lépéssel megnyerheti egy táblát, egy lépéssel megakadályozhatja, hogy a játékos megnyerjen egy táblát).
Szükséges lenne olyan logikát írni, ami nem csak a kis táblák szintjén, hanem a nagy táblán is megfelelő stratégiával tud játszani.