Programozás alapjai 1.

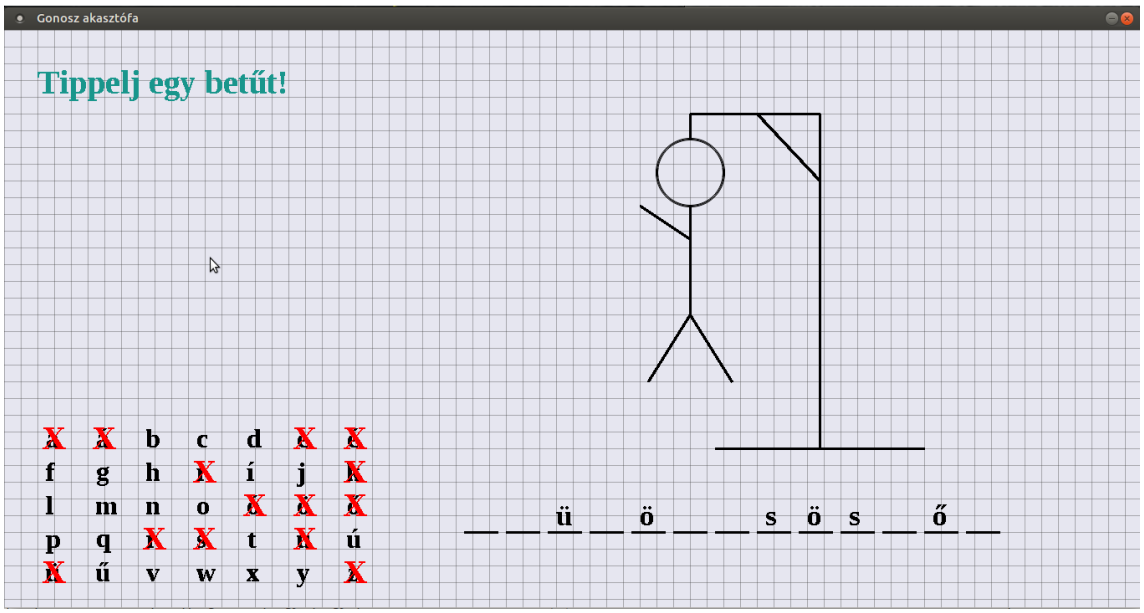
Nagy HF Dokumentáció

Horváth Ildikó (CW1TIZ)

2016 November

1. Bevezetés

A program egy „Gonosz akasztófa” játék, ami abban különbözik a hagyományos akasztófa játéktól, hogy a látszattal ellentétben valójában a gép nem „gondol egy szóra”, hanem a felhasználó tippjei alapján mindig úgy szűkíti a szóba jöhető szavak halmazát, hogy a lehető legkevesebb segítséget adjon a felhasználónak.



1. ábra. A játék kinézete

A játék indításakor a lehetséges szavak listáját a program egy fájlból beolvassa, majd az 1. ábrán is látható képernyő jelenik meg. A játékot ezek után a betűkre való kettintással lehet játszani.

2. Adatszerkezet

2.1. Az ékezetes karakterek

A program úgy készült, hogy képes legyen kezelni az UTF-8 kódolású ékezetes karaktereket is. Ezek a karakterek azomban nem feltétlenül egy bájton tárolódnak, így a programban a betűk nem lehettek karakter típusú változók és ezáltal a szavak sem egyszerű sztringek. A problémát úgy oldottam meg, hogy egy betűt egy legfejebb $3 + 1$ hosszúságú sztringben tároltam el (három bájton már az összes magyar ékezetes betű kifér) és így egy szó, a betűiből álló tömb, tehát egy két dimenziós karaktertömb lett.

2.2. A láncolt lista

A programban a fájlból beolvasott szavak egy egyszeresen láncolt listában tárolódnak el: Az újjonnan beolvasott szó a lista elejére kerül, így a szavak éppen fordított sorrendben vannak a listában, mint a beolvasás sorrendje. Azért választottam ezt, mert így volt a legkönnyebb felépíteni a listát, és a használt algoritmus nem követeli meg, hogy a lista bármilyen módon rendezett legyen.

```
typedef struct szavak{
    char szo[30][3+1];
    int betuszam;
    struct szavak *kovetkezo;
}szavak;
```

A lista elemeit a fentebb látható strukúra adja. Az adattagok egy legfejebb 30 betűből álló szó illetve a külön eltárolt *betuszam* – ez a szónak a hossza – amire azért volt szükség, mer az ékezetes karakterek miatti adatstrúktúrán a beépített sztringkezelő függvényeket nem lehetett a megszokott módon használni és a külön eltárolt betűszám sok esetben egyszerüsítette a kódot, illetve így nem volt szükség lezáró karakterre a szavak végén. A struktúra természetesen tartalmazza a listában a következő elemre mutató pointert is.

2.3. Globális változók

A programban két dologhoz – az *ABC*-hez és a *megfejtes*-hez – az algoritmust végrehajtó, a grafikus megjelenítésért felelős és a listát kezelő függvényeknek is hozzá kell tudni a férni és módosítaniuk is kell ezeket, így jobbnak láttam ez esetben a globális változók használatát.

```
extern char ABC[35][4];
extern char megfejtes[30+1][4];
```

Az ABC-ben kezdetben a magyar ABC kisbetűi találhatóak, majd amikor a felhasználó tippel egy betűt az utána az ABC-ből törlődik, egy szóköz kerül a helyére. A megfejtes kezdetben a szóközökből álló szó, amibe a játék folyamán bele kerülnek a betűk, amiket a felhasználó eltalált.

Ezen kívül globális változók még a színek illetve a megnyitott SDL ablak magassága és szélessége, mert ezeket az adatokat is hesználja a program több modulja is viszont ezek nem változnak a program futása alatt.

3. Grafikus felület

A grafikus megjelenítést az SDL könyvtár segítségével készítettem el. Ehhez az www.infoc.eet.bme.hu/sdl/ oldalon található segédletet és az előkészített Code::Blocks projectet használtam.

A program az egérkatintásokat egy eseményhurokban dolgozza fel, elindulás és inicializálás után a program minden esetben várakozik addig, míg érvényes egér kattintást nem észlel (olyan betűre kattintunk, ami még nem volt). Ez addig tart, míg a felhasználó az ablakot be nem zárja.

4. Algoritmus

A program indulásakor felépül a láncolt lista a beolvasott szavakból, megnyílik az SDL ablak, kirajzolódik a háttér és az ABC betűi valamint megjelenik a szöveges utasítás a játékhoz. A lista szavainak betűszámát figyelembe véve sorsolódik egy szóhossz és kirajzolódik ennek megfelelő számú vonalka is. A megfejtés feltöltődik szóközökkel és egy lezáró nullával a szóhossznek megfelelő helyen.

A láncolt listából minden esetben törölni kell azokat a szavakat, amik már nem, lehetnek a megfejtés. Mivel már adott a megfejtés hossza a listából ekkor törlődnek a nem megfelelő hosszúságú szavak.

A programban két számláló – az *akasztofaszamlalo* és a *betuszamlalo* – segíti a vezérlést. Ha az akasztófa számláló eléri a 14-et (ekkorra rajzolódik ki a teljes akasztott emberke) a játékos veszített. Ha a betű számláló – amely azt számolja, hogy eddig hány betűt talált el a játékos – eléri azt a számot, amilyen hosszú a megfejtés, akkor pedig nyert.

4.1. A „gonoszság”

Amikor a felhasználó tippel egy betűt, akkor a program megnézi, hogy van-e még a listában olyan szó, amelyik nem tartalmazza a tippelt betűt. Ha van, akkor „úgy tesz” mintha a felhasználó nem találta volna el egyik betűt sem a megoldásból és kirajzol egy újabb darabot az akasztószából. Ezek után természetesen ki kell törölni a listából azokat a szavakat amelyek tartalmazzák a tippelt betűt. Ezek már nem lehetnek a megfejtés.

Ha már nincsen a listában olyan szó, amely ne tartalmazná a tippelt betűt akkor már „muszály segítenie” a játéknak, de ekkor is próbál a lehető legkevesebbet. Ez esetben a program megkeresi azokat a szavakat amelyek a legkevesebbszer tartalmazzák a tippelt betűt és ezek közül választ egyet, amely meghatározza melyik helyeken legyen lerakva a tippelt betű.

Ez esetben nem rajzolódik ki újabb akasztófa darab, hanem kiíródik a tippelt betű a megfelelő vonalakra, és a listából kitörlődnek azok a szavak, amelyek nem a megfelelő helyeken tartalmazzák a tippelt betűt.

A játék végét a már említett számlálók valamelyike fogja meghatározni. A programban van egy logikai tipúsú változó amely azt tárolja, hogy kattinthat-e a felhasználó. A játék elején ez igaz értékű és amikor vége a játéknak hamissá válik. Ez biztosítja, hogy ne lehessen tovább tippelni a játék befejezése után.

Miután a felhasználó bezárta az ablakot a lista felszabadul és vége a programnak.

5. Modulok

A program három modulból épül fel, a *main*-ből, amely az eseményhurkot tartalmazza és a vezérlésért felelős, a *lista* modulból, amely kezeli a láncolt listát, beolvassa, törli és felszabadítja a lista elemeit valamint a *rajzol* modulból, amely a grafikus megjelenítésért felelős.

5.1. A main függvényei

`int random_betuszam(szavak *eleje):`

Végignézi a lista elemeinek betűszámait, megkeresi a maximumot és a minimumot, aztán sorsol egy random számot a minimum és a maximum között. Ellenőrzi, hogy van-e ilyen hosszúságú szó és ha igen visszatér a sorsolt számmal. Ha nincsen sorsol egy új értéket.

`bool poziciobol_betu(char* eredmeny, int x, int y):`

Egy megkapott (x, y) koordinátát átlakít betűvé, ha az egy betűnek felel meg a képernyőn. Ekkor visszatérési értéke igaz, egyébként hamis. Az átalakított betűt az eredmény változóba írja.

`bool van_rossz_szo(szavak *eleje, char *betu):`

A kapott betű és szólista alapján végignézi a listát és visszatérési értéke igaz, ha a lista tartalmaz még olyan szót, amely nem tartalmazza a paraméterként kapott betűt. Egyébként hamis értékkel tér vissza.

`int min_db_betu(szavak *eleje, char *betu):`

Végignézi a kapott szólistát és megszámolja, hogy az egyes szavak hányszor tartalmazzák a paraméterként kapott betűt. Ezek közül a legkisebb értékkel tér vissza.

5.2. A lista függvényei

`szavak *szavak_beolvas(char *file_nev):`

A paraméterben kapott fájlból beolvassa a szavakat és hozzáfűzi a listához. A visszatérési értéke a lista elejére mutató pointer.

```
szavak *torol_mas_betuszamuak(int betuszam,szavak *eleje),
szavak *torol_ha_van_benne(char *betu,szavak *eleje),
szavak *torol_nem_min(int minimum, char *betu ,szavak *eleje),
szavak *torol_ha_rossz_helyen(szavak *eleje):
```

A feltételnek megfelelő elemeket törli a listából. A visszatérési értéke a lista elejére mutató pointer.

```
int tartalmazza(szavak *szo, char* betu):
```

Megadja, hogy a paraméterként kapott szó hányszor tartalmazza a szintén paraméterként kapott betűt.

```
void feltolt_adattal(szavak *elem ,char *szo):
```

Ezt a függvényt a listát beolvasó függvény használja arra, hogy a fájlból beolvasott szót a megfelelő kétdimenziós karaktertömbbé alakítsa. A függvény beolvas egy bájtot és annak UTF-8 karakter kódja alapján eldönti, hogy egy, kettő vagy három bájtos karaktert lát-e és ennek megfelelően egy, kettő vagy három bájtnyi helyre olvassa be. Minden esetben a lezáró nullát is a karakter végére írja.

5.3. A rajzol függvényei

A rajzol modul függvényei az alábbiak, ezek működését nem részletezem, mert a függvények neve alapján ez egyértelmű.

```
void hatter_kirajzol(void)
void vonalak_rajzol(int db)
void ablak_megnyit(void)
void betutipus_betolt(int betumeret)
void ABC_kirir(void)
void szoveg_kiir(char *szoveg, int hely_x, int hely_y,int betumeret,
SDL_Color betuszin)
void akasztofa_darab_rajzol(int darab)
void ABCbol_kihuz(char *betu)
void kiir_betuk(int betuszam, SDL_Color szin)
void kiir_megfejtes(int betuszam, szavak *szo)
```

6. A felhasználónak

A programban használt lehetséges gondolt szavakat egy *szavak_Utf8.txt* nevű fájl tartalmazza, amiben alap esetben több mint 70000 ékezetes magyar szó található, ezek egymástól enterrel vannak elválasztva. A játék játszható ezekkel az alap szavakkal, de a lista tetszőlegesen bővíthető, szűkíthető a fájl módosításával. Megadható teljesen más fájl is (ha azonos a formátum, tehát a szavak egymástól enterrel vannak elválasztva) így játszható a játék más nyelvek szavaival, vagy speciális csoport számára is, például óvodásoknak csak állatnevekkel vagy színekkel.

7. A tesztelésről

A programmal többen is játszottak, hibát a működésben nem sikerült találniuk, minden az elvárásoknak megfelelően működött. A fájlkezelést teszteltem úgy, hogy rossz fájlnevet adtam meg, vagy hiányzott a fájl a mappából, ez esetben a megfelelő hibaüzenetet kaptam, miszerint nem sikerült megnyitni a fájlt. Megpróbáltam úgy is elindítani a programot, hogy a szavakat tartalmazó fájl üres volt, ekkor is a megfelelő hibaüzenetet kaptam miszerint nem sikerült szavakat beolvasni.