MECHATRONIKA SZAKOSZTÁLY

MSZO projektdélután c++ programozás: snake játék irányítása

Általános adatok

A snake egy először 1976-ban megjelent atari játék. A játék során egy kígyót irányítasz, ami 4 irányba tud mozogni. Feladatod minél több almát megenni, miközben kikerülöd a tested és a falakat. Az almáktól nő a kígyó hossza.

A projektdélutánon egy előre megírt konzolos snake játékot kell beprogramoznotok c++ nyelven, hogy minél ügyesebb legyen. A programozáshoz 3 fájlt kell használni, mindhárom mellékelve lesz egy Visual Studio projektben. Az első kettő a *SnakeGame.h* és *SnakeGame.cpp*, az előbbit includeolni kell. Ezek tartalmazzák a játékot, módosítani őket (alapos indok nélkül) tilos! Megértésük a feladatmegoldáshoz nem szükséges, de szabad. (452 sor, ki van kommentezve) A *SnakeGame.h*-ban vannak a használható függvényeknek, és a SnakeGame osztály tagfüggvényeinek a fejlécei, egy Coord struktúra x és y tagokkal, valamint a definiált konstansok. (A vezérléshez az UP, DOWN, LEFT, RIGHT, STOP (megállni nem lehet, ha már elindultál); és a pálya méretei, WIDTH és HEIGHT) A harmadik fájl az *MSZO_challenge.cpp*, ami egy rövid, ~30 soros példaprogramot tartalmaz, amelyben az irányítás nem automatikus, hanem billentyűzettel történik.

1. ábra: A játék "grafikus" megjelenítése

Az osztályt az alábbi módon lehet használni:

1) Includeolod a headert (a fájlok legyenek egy mappában)
2) Létrehozol egy példányt az osztályból
3) Egy while ciklusban rakod a többi elemet
4) Lekéred a játék állását (SnakeGame::getMap(), getSnake(), getHead, getApple() tagfüggvényekkel)
5) Rajzold ki, mi van (SnakeGame::render())
6) Valahogy döntsd el, merre menjen a kígyó, és léptest is (SnakeGame::move(int direction))

Egy sablon:

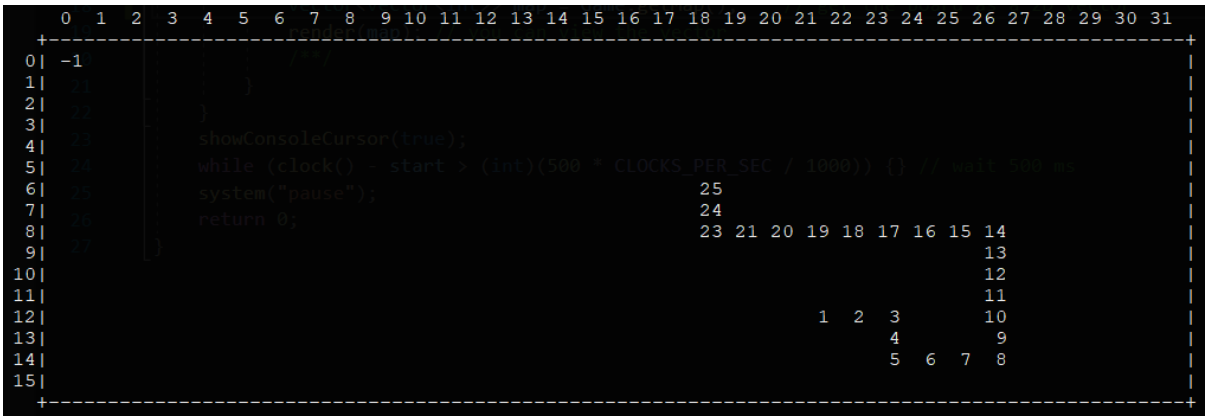
```
#include "SnakeGame.h"      // 1) importáld a játékot

using namespace std;

int main()
{
	int direction = STOP;	// definiálj egy integert, amivel az irányt vezérled
	SnakeGame Game = SnakeGame();	// 2) Hozz létre egy példányt a játékból
	while (!Game.getGameOver())	// 3) játssz, amíg nincs vége a játéknak
	{
		Game.render();	// 5) Néha rajzold ki, hogy épp mi történik
		vector<vector<int>> map = Game.getMap(); // 4) kérdd le a játék állását
		render(map);	// kiírhatod a kapott vektort, ami szerkeszthető is
		/*
			IDE ÍRD AZ IRÁNYÍTÁST!
			...
			direction = ...;
		/**/
		Game.move(direction);	// 6) Léptesd a kígyót
	}
	system("pause");	// ne zárdd be a konzolt
}
```

A fontos függvények leírásai kommentben a kódban vannak. Az alábbiakat érdemes használni:

- `vector<vector<int>> getMap();`
  Egy 2D-s STL vektorral (dinamikus tömb) tér vissza. A tömb akkora, mint a pálya, az üres helyeken 0-t, az alma helyén -1-t, a kígyó fejénél 1-et, a testénél növekvő számokat tartalmaz. Ezt használhatod útkeresésre, módosíthatod, majd debuggolásként kirajzolhatod a `void render(vector<vector<int>> map, int w = 3, int empty = 0);` függvénnyel:



2. ábra: A fenti felállás a render(Game.getMap()); függvénnyel kirajzolva. Az üres helyeken 0 van.

- `Coord getApple();`
  Visszatér az alma koordinátáival
- `Coord getHead();`
  Visszatér a kígyó fejének koordinátáival
- `vector<Coord> getSnake();`
  Visszatér egy vektorban a kígyó testrészeinek koordinátaival. A [0] elem a fej
- `SnakeGame(string title = "Snake Game"); bool move(int dir = STOP); void render();` Lásd a sablonban. Ezeket mindenképp használni fogod.

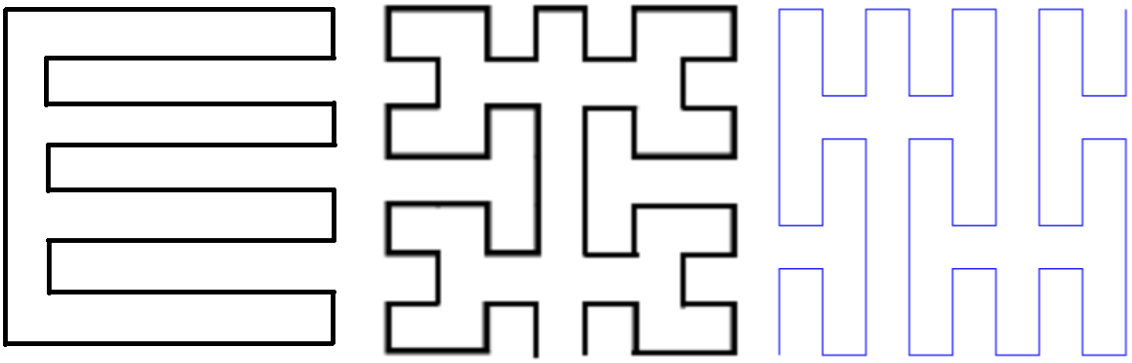
A többi függvényt nem igazán kell használni, de persze lehet.

1. feladat

Írj programot a getApple() és getHead() függvényekkel! A kígyó a legrövidebb úton (mondjuk L alakban) menjen rá az almára, semmit se kell kikerülnie. Ez nyilván rövid életű lesz, de legalább a tied!

2. feladat

Írj programot, ami egy Hamilton kör mentén bejárja az összes mezőt! Írd meg, hogy csak minden n. lépés után rajzolja ki a pályát. Ezt később felhasználhatod, például a kígyó mehet alapból ezen a körön, de ahol tudja, levághatja az utat. Nem muszáj a legegyszerűbben megcsinálni. Példák a bejárásra: (space filling curves)



3. ábra: A leggagyibb megoldás *A Moore görbe* *A Peano görbe*

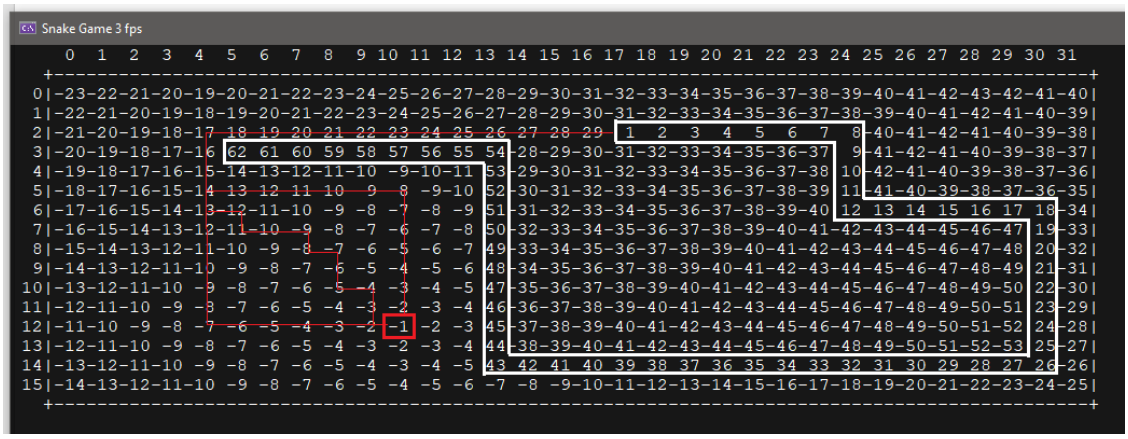
3. feladat

Írj programot a legjobb tudásod szerint. (Szinte) bármit használhatsz! A kígyó növekedjen a lehető leggyorsabban (alapból a lépéseket számoljuk, de a nagyon sokat gondolkozó megoldások szankcionálva lesznek), és élje túl a lehető legtovább (egyen minél több almát). Nem szükséges kivinni a játékot, arra képes a 2. feladatos program is.

Lehetséges megközelítések:

- Neurális háló (reinforcement learning, deep q learning), nem ajánlott

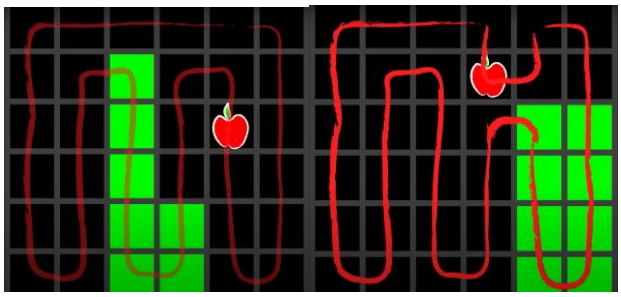
- Útkereső
  Utána olvashatsz a Breadth-first search, Best-first search, A* search útkereső algoritmusoknak, vagy kitalálhatsz valamit te is. Az én megoldásom az alábbi módon tölti ki az üres helyeket, majd a kígyó mindig a legnagyobb nemnegatív szám felé halad. A kitöltéshez valami while ciklusos rekurzív szerű dolgot és egy vektort vagy listát javaslok.



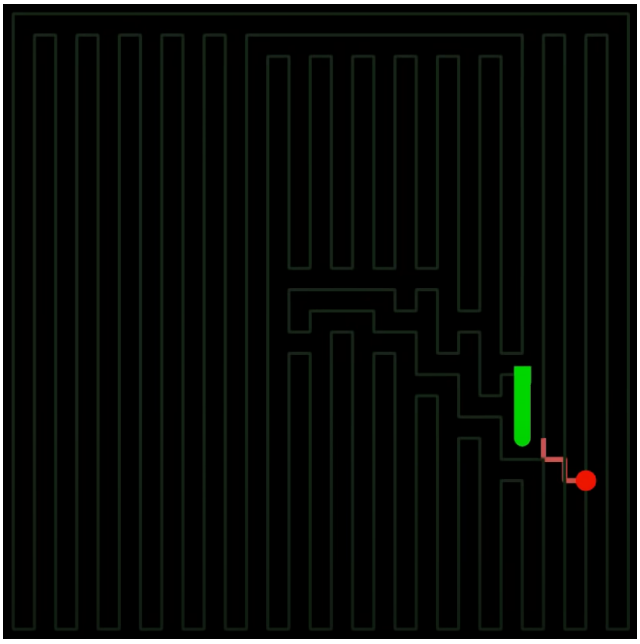
4. ábra: Az útkereső, kiemelve az alma, a kígyó, és néhány lehetséges a legrövidebb utakból

Az ilyen algoritmusok rendszerint előbb utóbb körbe falazzák magukat, és meghalnak, vagy lefagy a program. Érdemes lehet válogatni a lehetséges utak közt, mondjuk mehetsz saját magad vagy a fel mellett, hogy elkerüld ezt. Továbbá ellenőrizni kell, hogy az adott lépés elvág-e téged nagyobb szabad területtől.

- Körbejáró algoritmus, rövidítésekkel
  Az alapötlet, hogy a 2. feladat valamely bejárását alkalmazod, de ha lehet levágod az utat, és egy új Hamilton körre térsz vissza.



5. ábra: Az útvonal levágása



6. ábra: Kígyó, ami a legegyszerűbb Hamilton kört követi, de levágja az útját, ahol tudja