FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Dokumentácia
Projekt č. 11 do predmetu *Modelování a simulace*

Modely přírodních a ekologických katastrof (Lesné požiare)

2020/2021

Jakub MÍkvy (xmlkvy00)
Adam Múdry (xmudry01)

7. decembra 2020

Obsah

1 Úvod

Úlohou tohto projektu bolo naprogramovať program (*ims*), ktorý simuluje prírodné a ekologické katastrofy, v našom príade lesné požiare.

Zoznam odovzdaných súborov:

```
src/main.cpp src/map.hpp src/cell.hpp src/etc.hpp
Makefile
dokumentacia.pdf
README.md
```

1.1 Preklad

Program sa prekladá pomocou nástroja **make** spusteného v koreni priečinku:

```
make
```

alebo

```
make build
```

Pre vytvorenie ladiacej verzie programu použite príkaz:

```
make build-debug
```

Makefile spúšťa program **g++** s nasledujúcimi parametrami:

```
--std=c++17 -Wall -Wpedantic [-O3|-g]
```

a linkuje vytvorené objektové súbory na knižnice pomocou:

```
 -lglut -lGLU -lGL
```

Vytvára sa spustiteľný súbor **ims** na koreni priečinku.

1.2 Použitie

Príkaz na spustenie (GUI verzia):

```
make run
```

alebo verzia pre terminál:

```
make run-terminal
```

alebo priamo cez vygenerovaný spustiteľný súbor:

```
./ims [-h|--help] [-g|--gui]
```

```
Príznaky a argumenty:
    -h, --help        Ukáže pomocnú hlášku
    -g, --gui         Zapne podrobný výpis
```

Pred použitím programu musí existovať jeho spustiteľný súbor.

2 Teória

Lesný požiar sa v realite šíri a udržuje podľa veľkého množstva faktorov. Zohľadnenie týchto faktorov, vedie k extenzívnym ale presným simuláciam. Jedným z týchto modelov je **ABBAMPAU model**, ktorý zahŕňa faktory *výška, typ vegetácie, teplota, vlhkosť, typ horenia, dĺžka horenia, smer vetra, sila vetra, horlavosť*. Riadi sa štvorcovými bunkami a Moorovym algoritmom najbližších susedov (algoritmus sa pozerá na 4-roch hraničných a 4-och diagonálnych susedov).

2.1 SSL

Secure Sockets Layer (doslova vrstva bezpečných socketov) je protokol, resp. vrstva vložená medzi vrstvu transportnú (napr. **TCP/IP**) a aplikačnú (napr. **HTTP**), ktorá poskytuje zabezpečení komunikácie šifrovaním a autentifikáciu komunikujúcich strán. [**?**]

Ukážka HTTP *GET* požiadavky pre získanie správ z určitého kanálu:

```
GET /api/channels/<ID kanálu>/messages HTTP/1.1\r\n
Host: discord.com\r\n
Authorization: Bot <Token>\r\n
\r\n
```

Ukážka HTTP *POST* požiadavky pre odoslanie odpovedi na server:

```
POST /api/channels/<ID kanálu>/messages HTTP/1.1\r\n
Host: discord.com\r\n
Authorization: Bot <Token>\r\n
Content-Type: application/json\r\n
Content-Length: <Dĺžka správy>\r\n
\r\n
{"content": "echo: <Meno používateľa> - <Správa>"}\r\n
\r\n
```

3 Implementácia

Zadanie projektu som implementoval v jazyku C++ písaného v štandarde C++17.

3.1 Načítavanie argumentov

Po spustení programu sa načítavajú argumenty zo štandardného vstupu (*stdin*) pomocou funkcie

```
void parseArgs(int& argc, char* argv[], bool& show_help, bool& verbose, std::string&
    token);
```

ktorá využíva knižnicu *getopt.h* a jednotlivé načítané argumenty vráti do premenných z parametrov funkcie.

3.2 Zahájenie SSL spojenia

Po načítaní *tokenu* a ostatných argumentov zo štandardného vstupu program naviaže SSL spojenie pomocou funkcie

```
int initConnection(SSL** ssl_return);
```

ktorá vracia ukazovateľ na štruktúru **SSL** využíva OpenSSL knižnice *openssl/ssl.h*, *openssl/err.h* a pár iných knižníc určených pre programovanie so sieťami ako napr. *netinet/in.h*. [**?**]

3.3 Získavanie informácii

Získanú *SSL* štruktúru môžeme využiť v nasledujúcich funkciách

```
int sendPacket(SSL* ssl, const char *buf);
std::string recieveResponse(SSL* ssl);
```

z ktorých prvá posiela naše *GET* alebo *POST* požiadavky na Discord API a druhá spracováva odpoveď a vráti ju vo forme *std::string*. [**?**]

Telo HTTP odpovede je vo formáte **JSON**, ktorý program spracuje pomocou funkcií zo štandardnej knižnice C++ **regex** a naplní štruktúry

```
struct Message; // id, user_id, channel_id, timestamp, username, content, ...
struct Channel; // id, guild_id, name, messages, ...
struct Guild;   // id, name, channels, ...
```

ktoré obsahujú dáta ako id, názvy a podobne a sú vzájomne napojené.

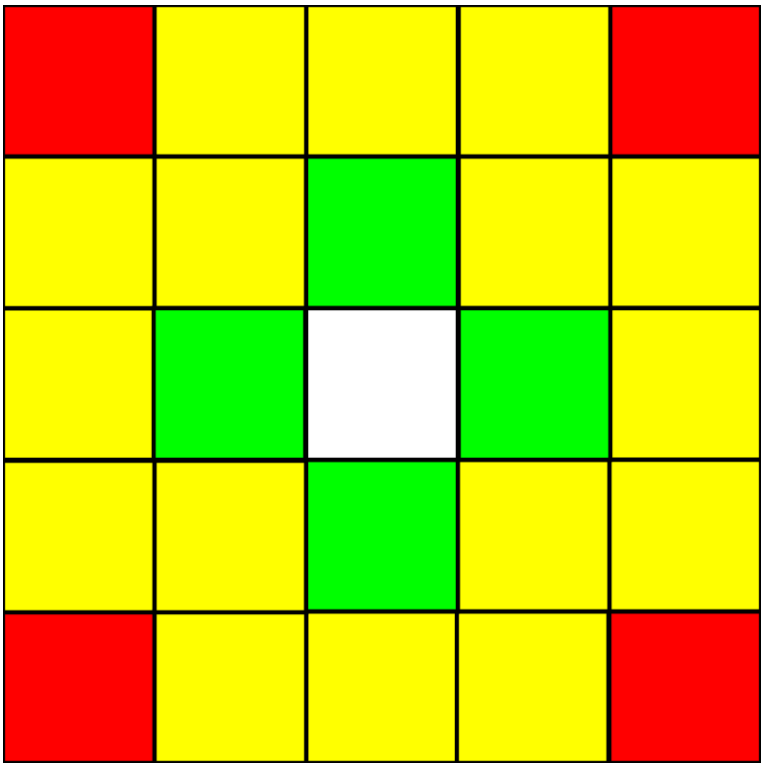
3.4 Odpovedanie na správy používateľov

Po načítaní všetkých potrebných informácii môže program začať načítavať správy z jednotlivých kanálov a odpovedať na ne. Toto sa deje v jednom **while** cykle. Program porovnáva lokálny čas a čas odoslania správy, získaný z *JSON* odpovede Discord serveru. Ak je správa staršia ako porovnaný čas, program ju ignoruje - ak je však správa nová, program ju zaregistruje a pošle odpoveď na server. Program taktiež ignoruje správy, ktoré poslali iný boti.

Isabot odpovedá s krátkym oneskorením, aby nepresiahol limit HTTP požiadaviek určený z Discord API.

4 Testovanie

Program som testoval manuálne pomocou 2 vlastných skupín a kanálov a vlastného Discord účtu.



Obr. 1: Zobrazenie vnímania vzdialenosti od bunky.
Biela - aktuálna bunka, *zelená* - bunky (4) o vzdialenosti 1,
žltá - bunky (16) o vzdialenosti 2, *červená* - bunky (4) o vzdialenosti 3.

5 Bibliografia

Literatúra

[1] Bafas, A. A. D. V. C. S. G.: A cellular automata model for forest fire spread prediction: The case of the wildfire that swept through Spetses Island in 1990. [online], [Applied Mathematics and Computation 204 (2008) 191–201], [vid. 2020-12-06].
URL https://doi.org/10.1016/j.amc.2008.06.046

[2] Javidx9: OneLoneCoder.com - What Is Perlin Noise? [online], [vid. 2020-12-05].
URL https://github.com/OneLoneCoder/videos/blob/master/OneLoneCoder_PerlinNoise.cpp