МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КПІ»

Кафедра

Автоматизованих систем обробки інформації та управління

Звіт  
з лабораторної роботи з дисципліни  
«Методи та системи створення штучного інтелекту»

|  |
| --- |
| Виконала  Коробова Елеонора |
| Студентка гр. ІС-32, ФІОТ |
| 3 курс,  Залікова книжка  № IC-3206 |

|  |
| --- |
|  |
|  | |  |
|  | |  |

Київ 2016

## Завдання.

1) Необхідно розробити програми раціональних агентів двох типів: простого рефлексного агента та рефлексного агента, якій заснований на моделі. Для створення власного агенту потрібно модифікувати файли agent.cpp та agent.h оригінального проекту. Агент з моделлю середовища може отримувати інформацію про стан середовища через клас Environment (файли environment.cpp/.h)

2) Створити декілька власних карт середовища (на основі прикладу agent.map)

3) Провести серію експериментів для визначення ефективності роботи програм агентів. Порівняти два створених агенти між собою, а також з оригінальним агентом (який йде в дистрибутиві проекту «Vacuum Cleaner World») та діє випадковим чином. Порівняння зробити для всіх створених карт, а також оригінальної карти agent.map. Вхідним параметром для експериментів вважати час моделювання (наприклад: 100, 500, 1000, 2000, 5000, 10000). За проведеними серіями необхідно визначити: • середній ступінь бруду (average dirty degree); • середню використану енергію (average consumed energy). Результати експериментів представити у вигляді таблиць та графіків.

## Принцип роботи розроблених агентів

* Простий рефлексний агент.

Простий агент обирає шлях випадковим чином, та рухається кілька кроків в обраному напрямку. У випадковий момент часу робот звертає у бік, генеруючи новий шлях. Якщо робот врізається у перешкоду, то його шлях знову генерується випадковим чином.

* Агент, що використовує модель.

Перші 200 кроків для цього агенту є навчальними, тож він діє як звичайний простий агент, запам’ятовуючи мапу місцевості. Кожен наступний крок агенту визначається наступним чином:

* Проаналізувати сусідні 4 клітинки (верхню, нижню, праву та ліву).
* Користуючись мапою, яка вже зіставлена, визначити, які клітинки є перешкодами.
* Для пустих клітинок (без перешкод) визначити рівень бруду та піти у ту, де він є найбільшим.
* Якщо ж рівень бруду для всіх сусідніх клітинок однаковий, то агент обирає будь-яку з них.

Навіть якщо за 200 перших кроків агент не встиг запам’ятати всю карту, то він зробить це надалі, пересуваючись картою. Це надає перевагу у тому випадку, якщо розміщення перешкод постійно змінюється.

## Результати проведення серії експериментів

* Рівень бруду. (1 мапа, 500 кроків)

Рис.1

* Рівень спожитої енергії. (1 мапа, 500 кроків)

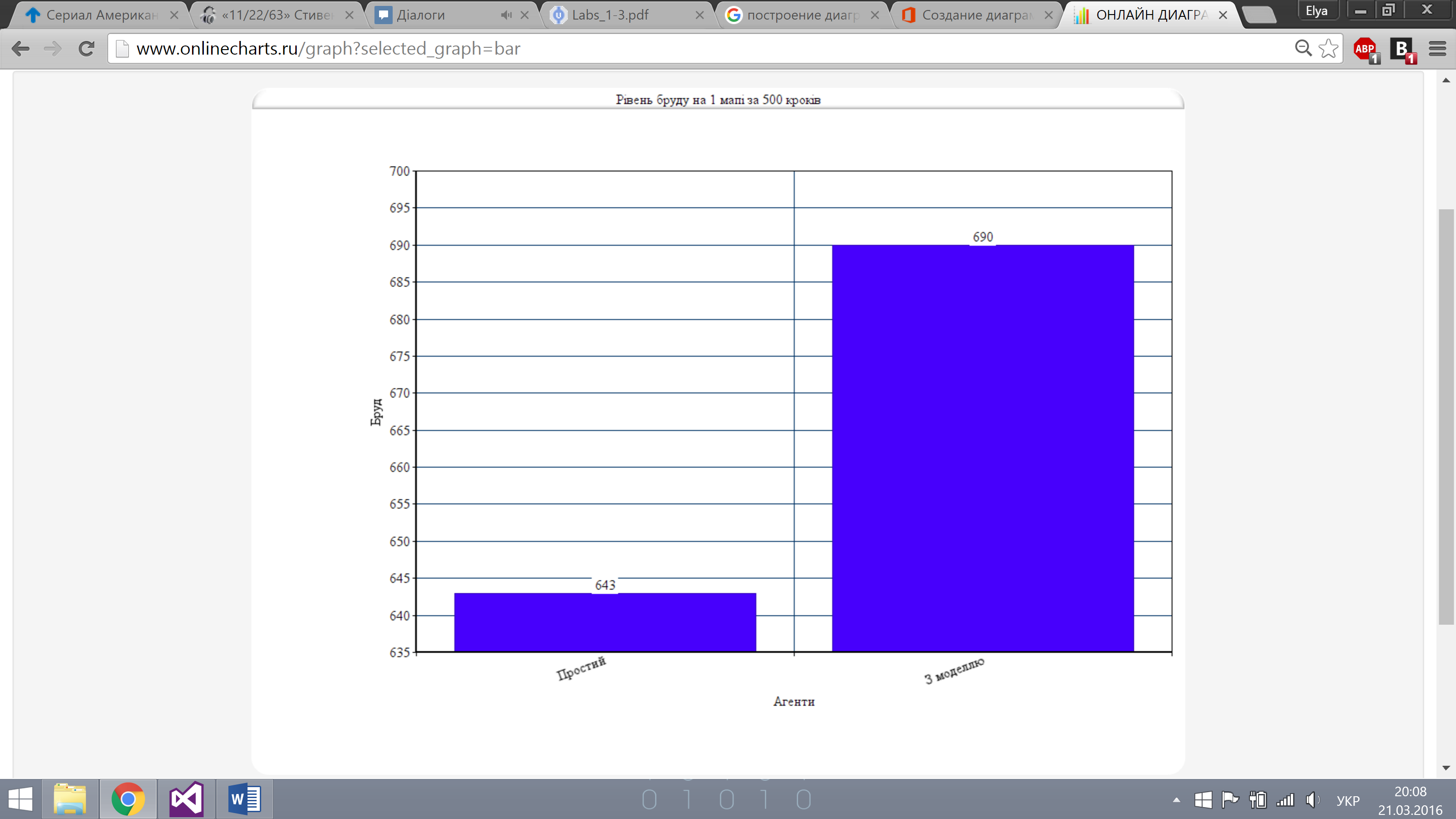


Рис.2

* Рівень бруду. (1 мапа, 1000 кроків)

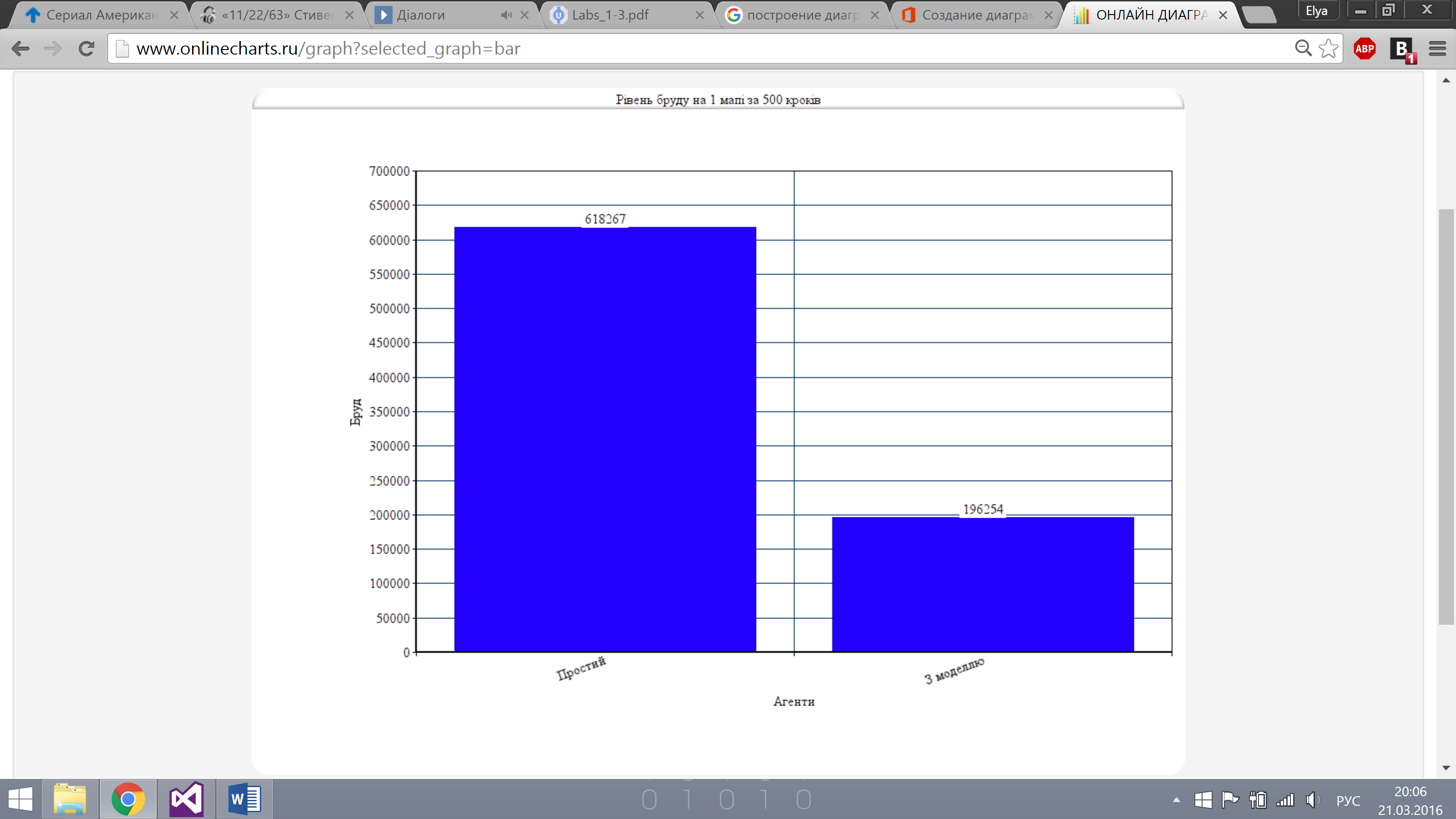


Рис.3

* Рівень спожитої енергії. (1 мапа, 1000 кроків)

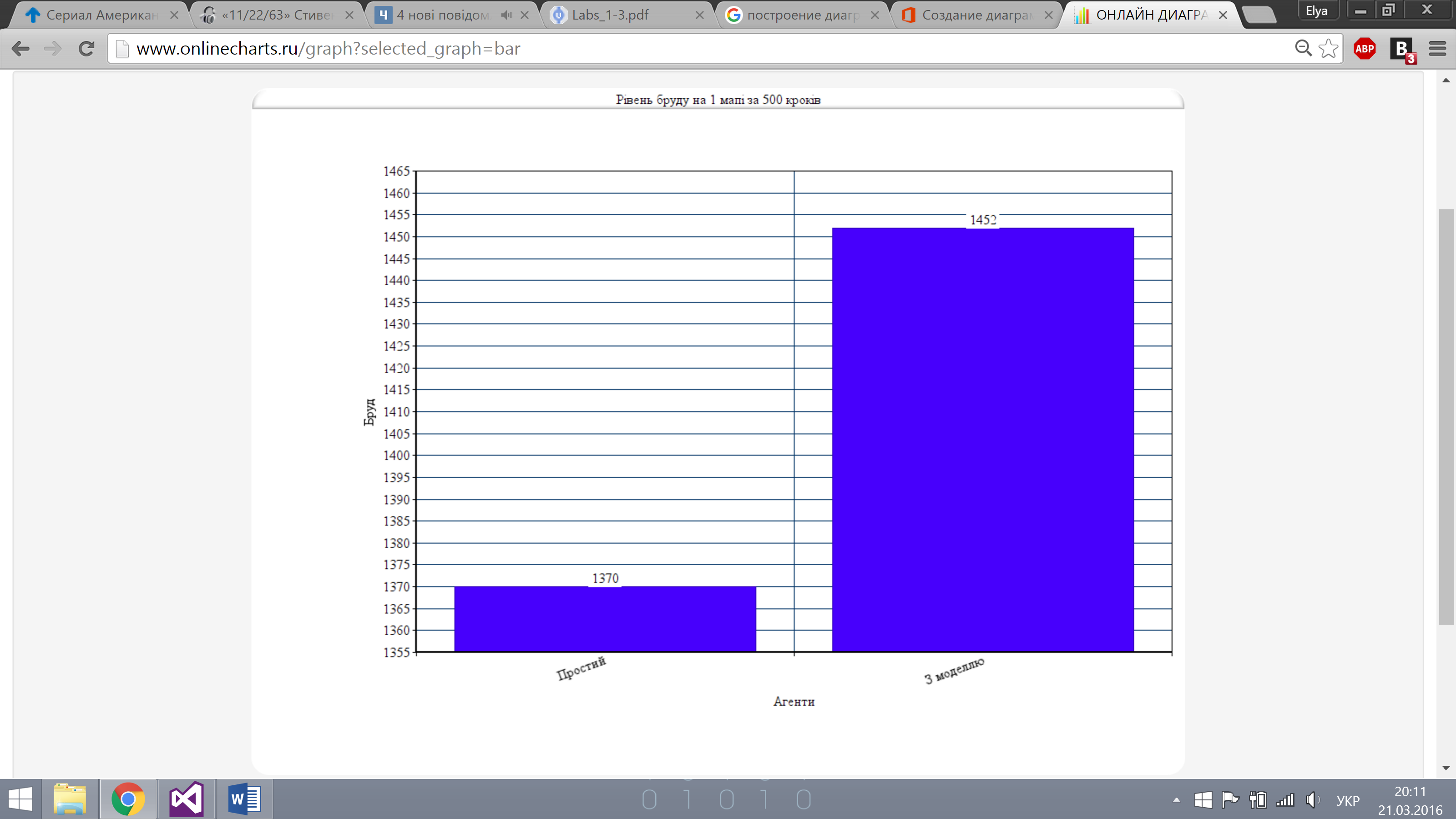


Рис.4

* Рівень бруду. (2 мапа, 500 кроків)

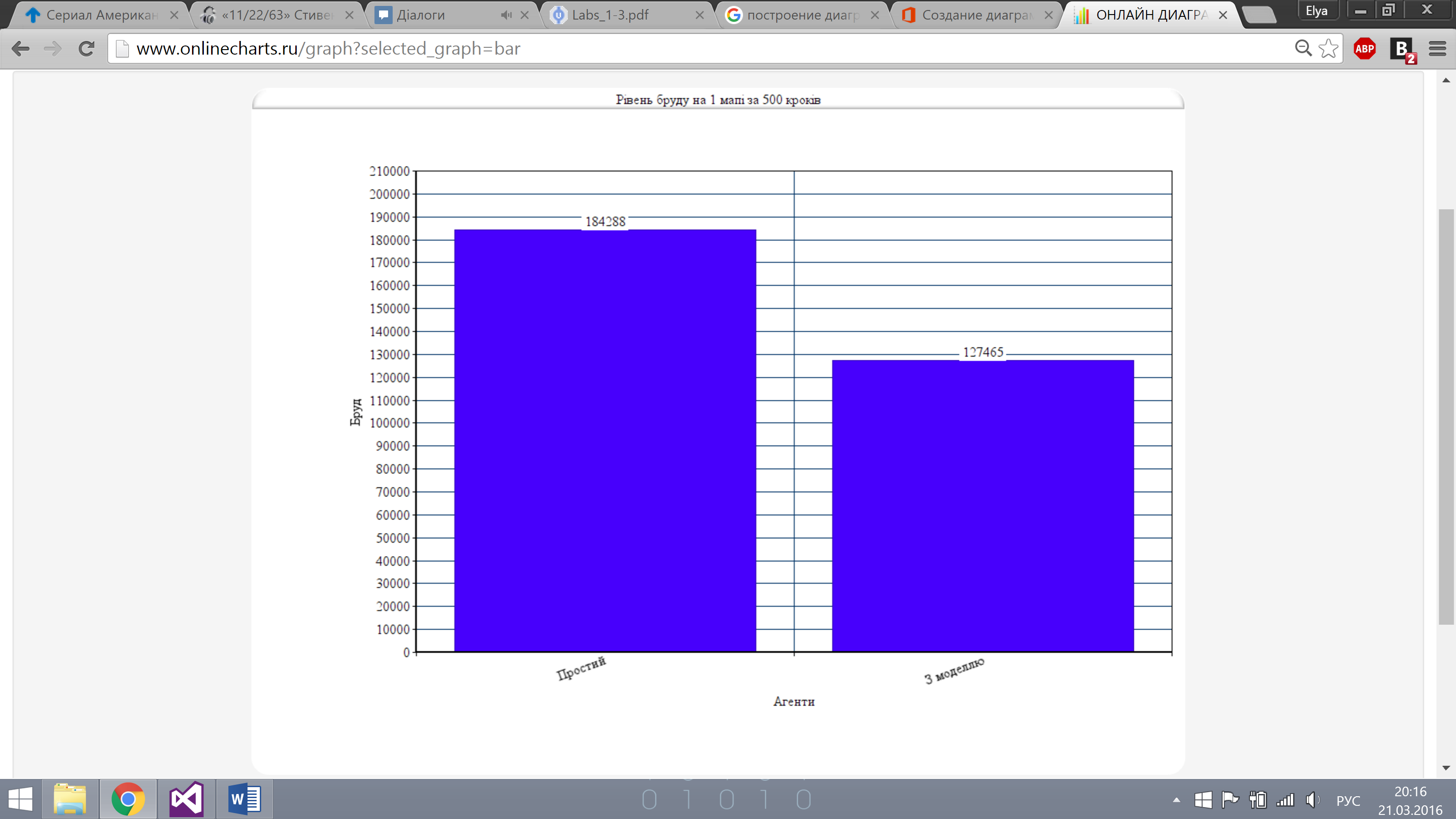


Рис.5

* Рівень спожитої енергії. (2 мапа, 500 кроків)

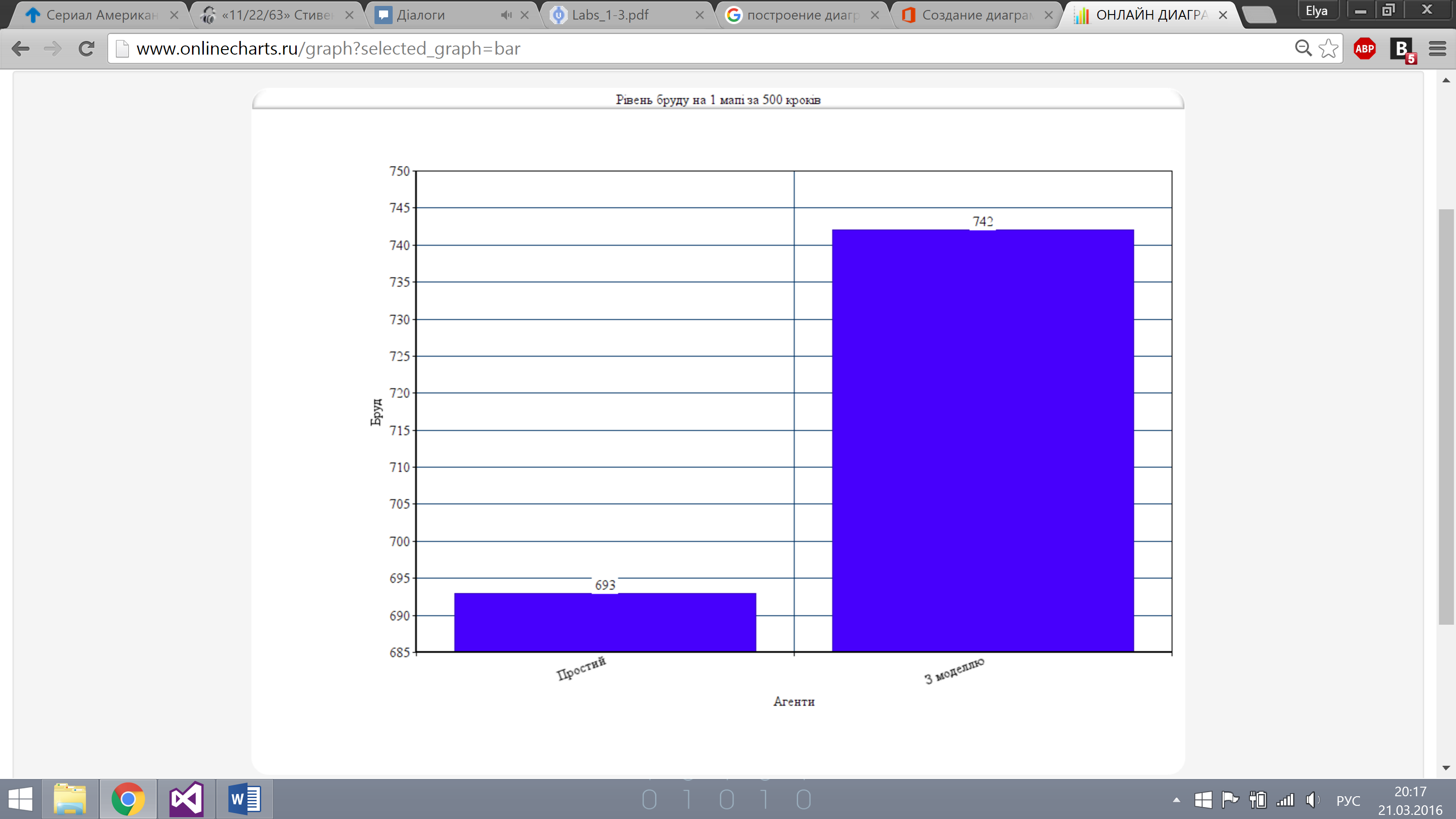


Рис.6

* Рівень бруду. (2 мапа, 1000 кроків)

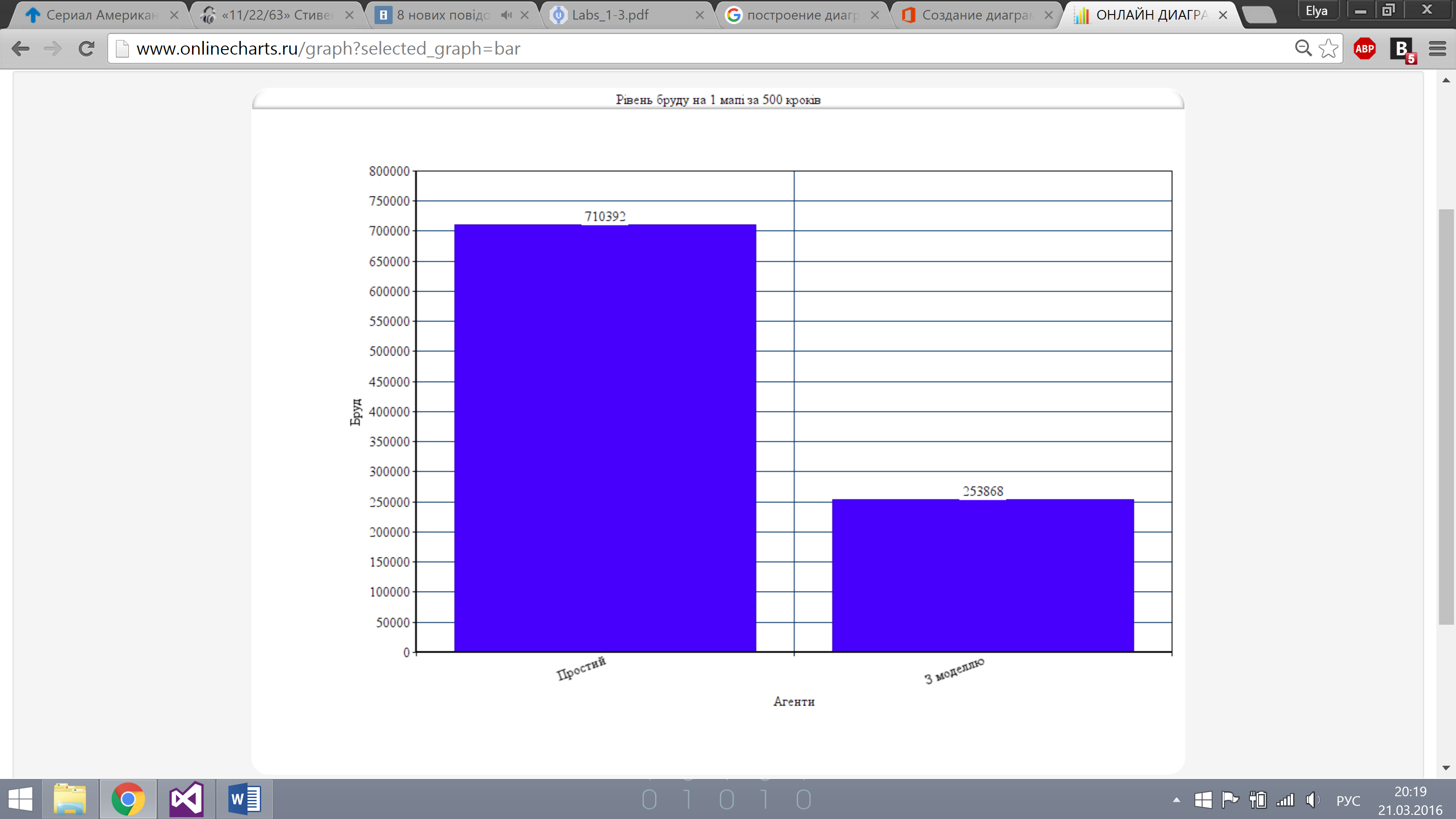


Рис.7

* Рівень спожитої енергії. (2 мапа, 1000 кроків)

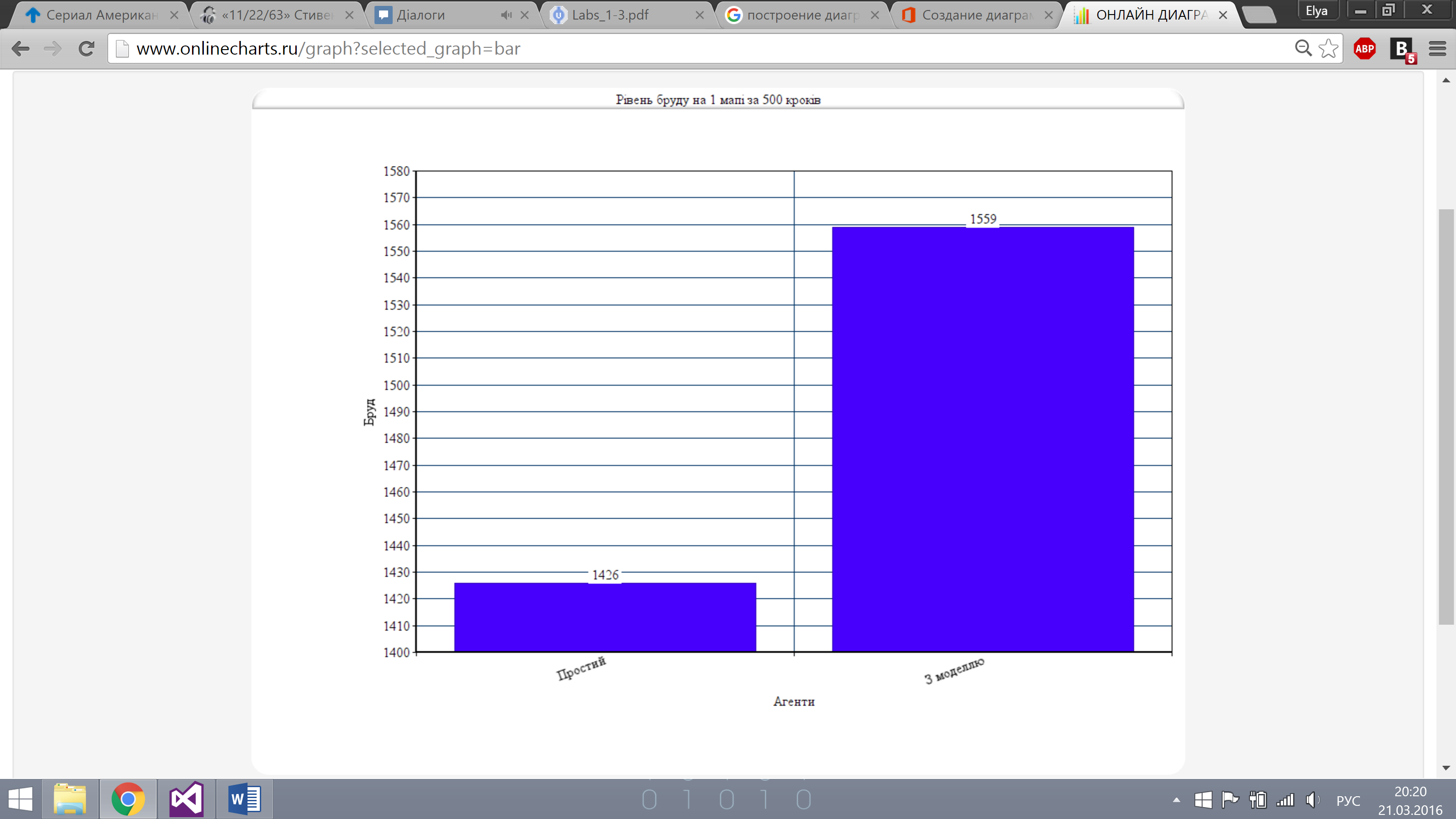


Рис.8

* Рівень бруду. (3 мапа, 500 кроків)

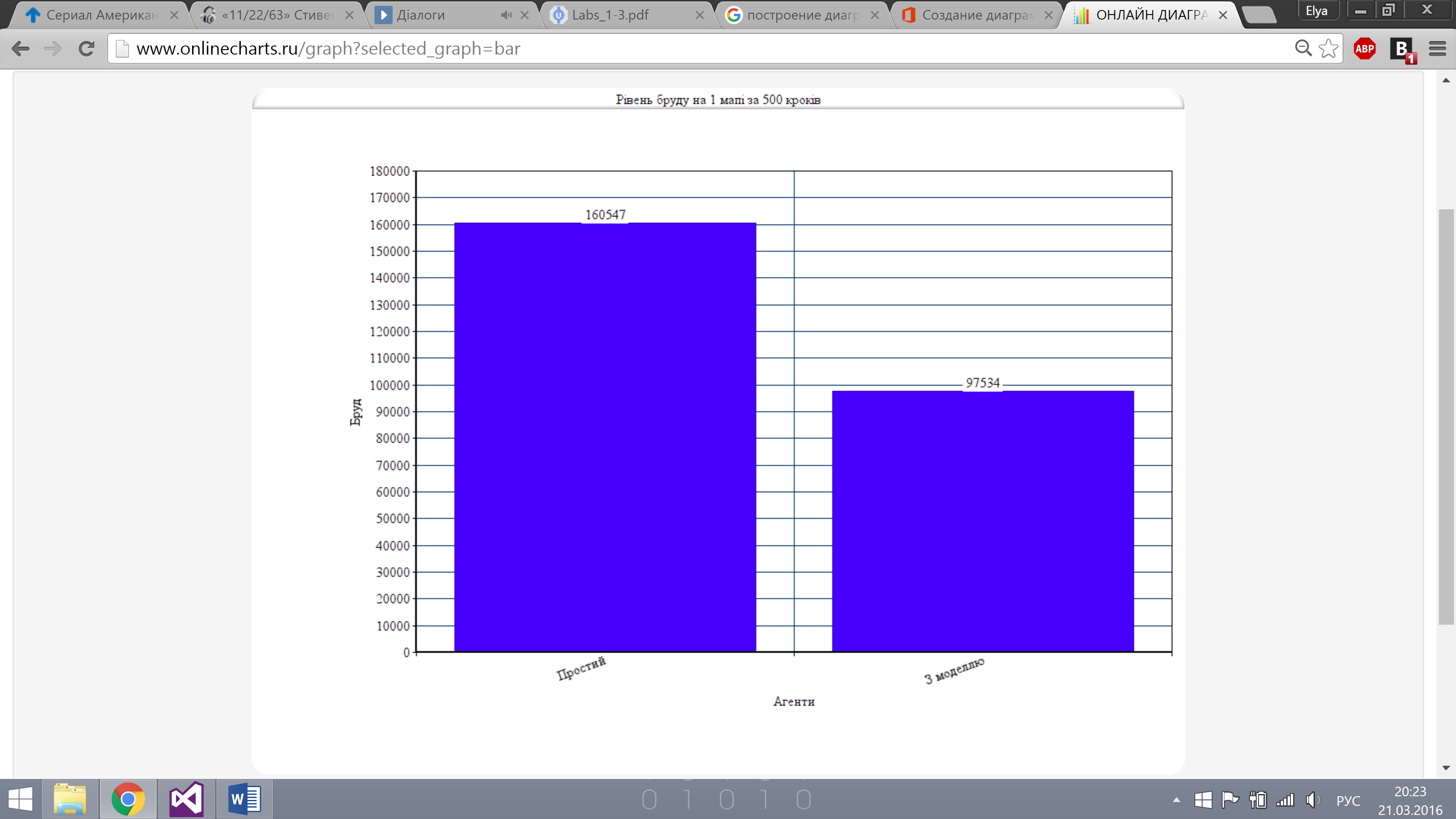


Рис.9

* Рівень спожитої енергії. (3 мапа, 500 кроків)

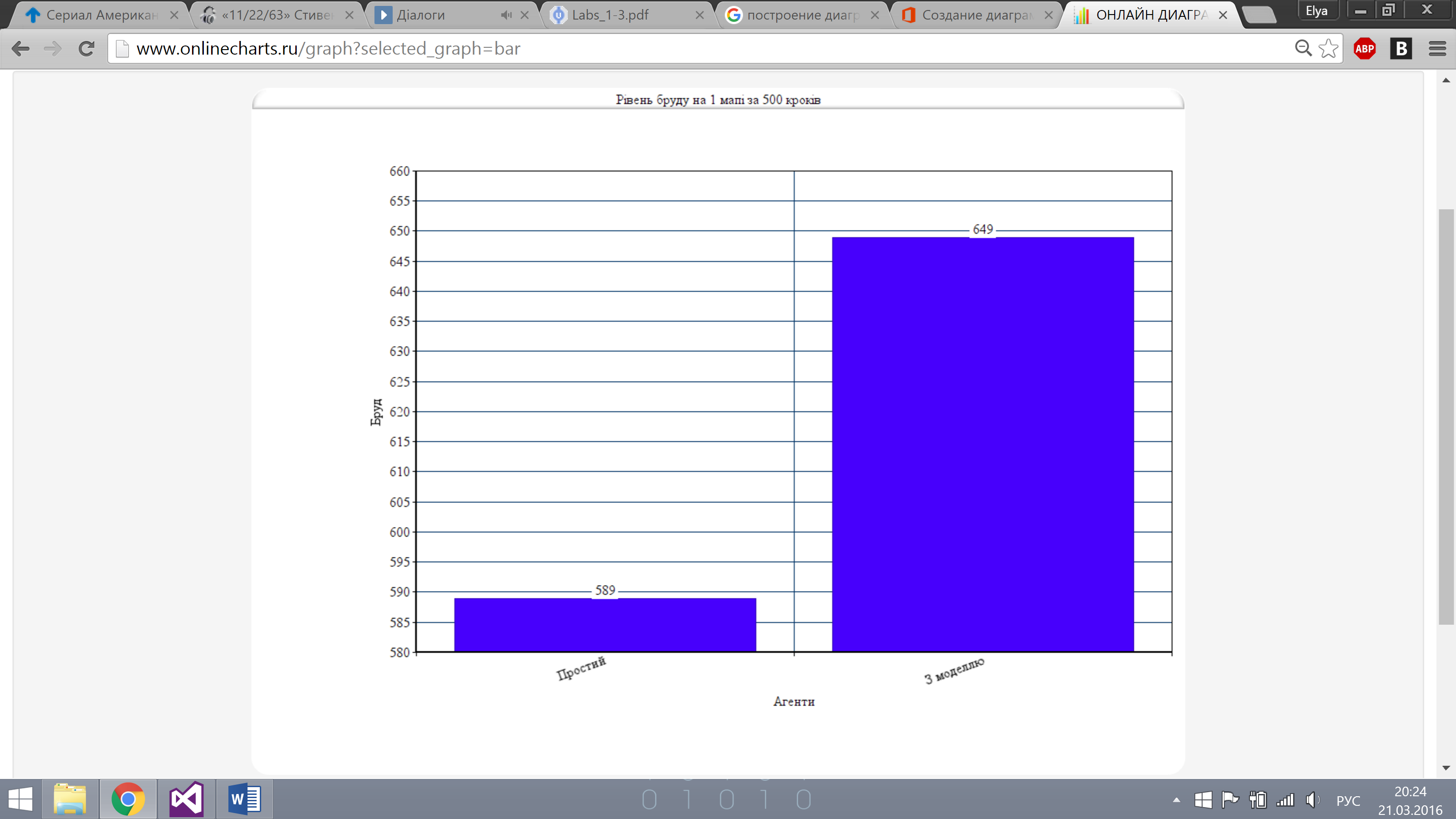


Рис.10

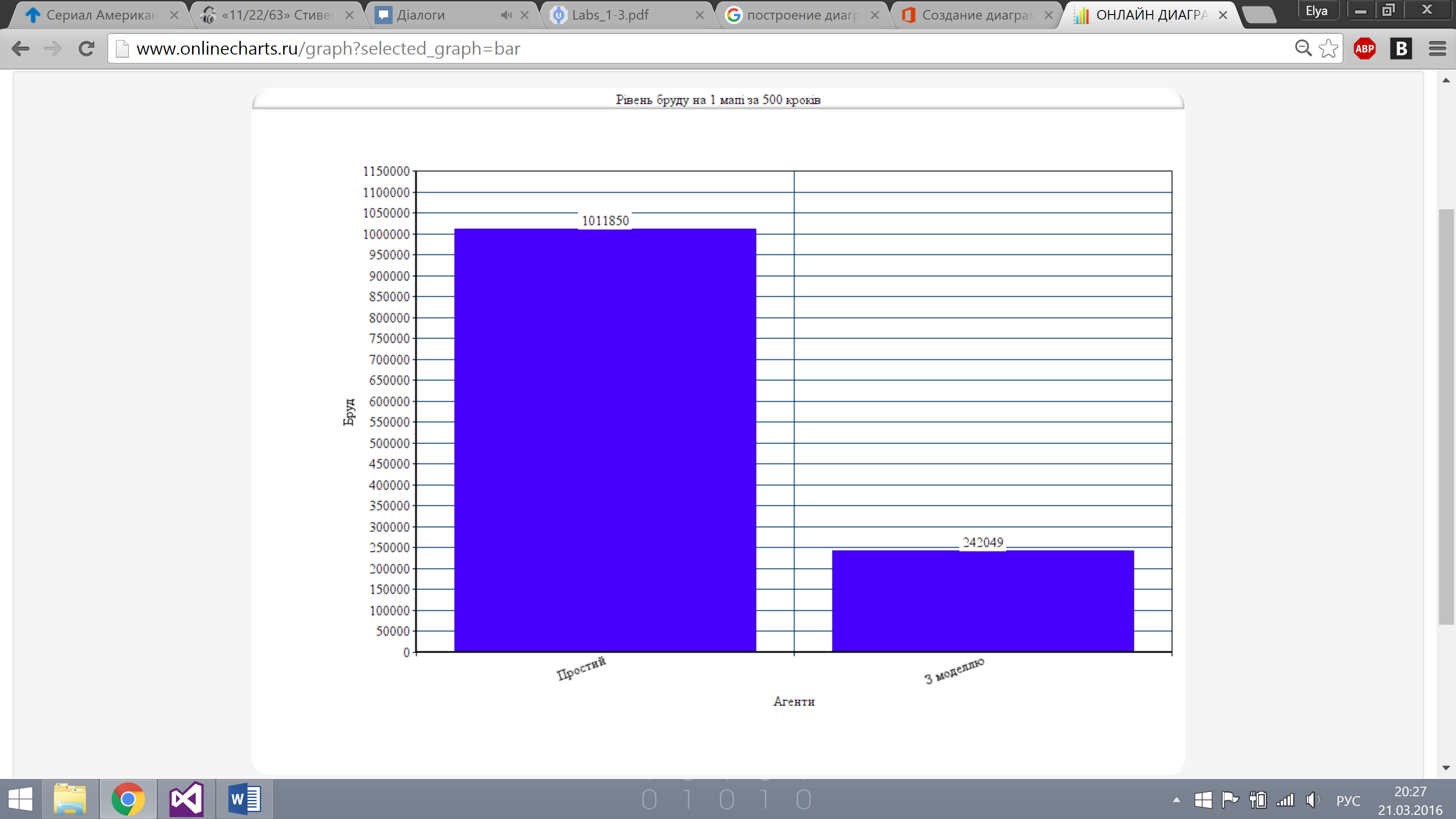
* Рівень бруду. (3 мапа, 1000 кроків)

Рис.11

* Рівень спожитої енергії. (3 мапа, 1000 кроків)

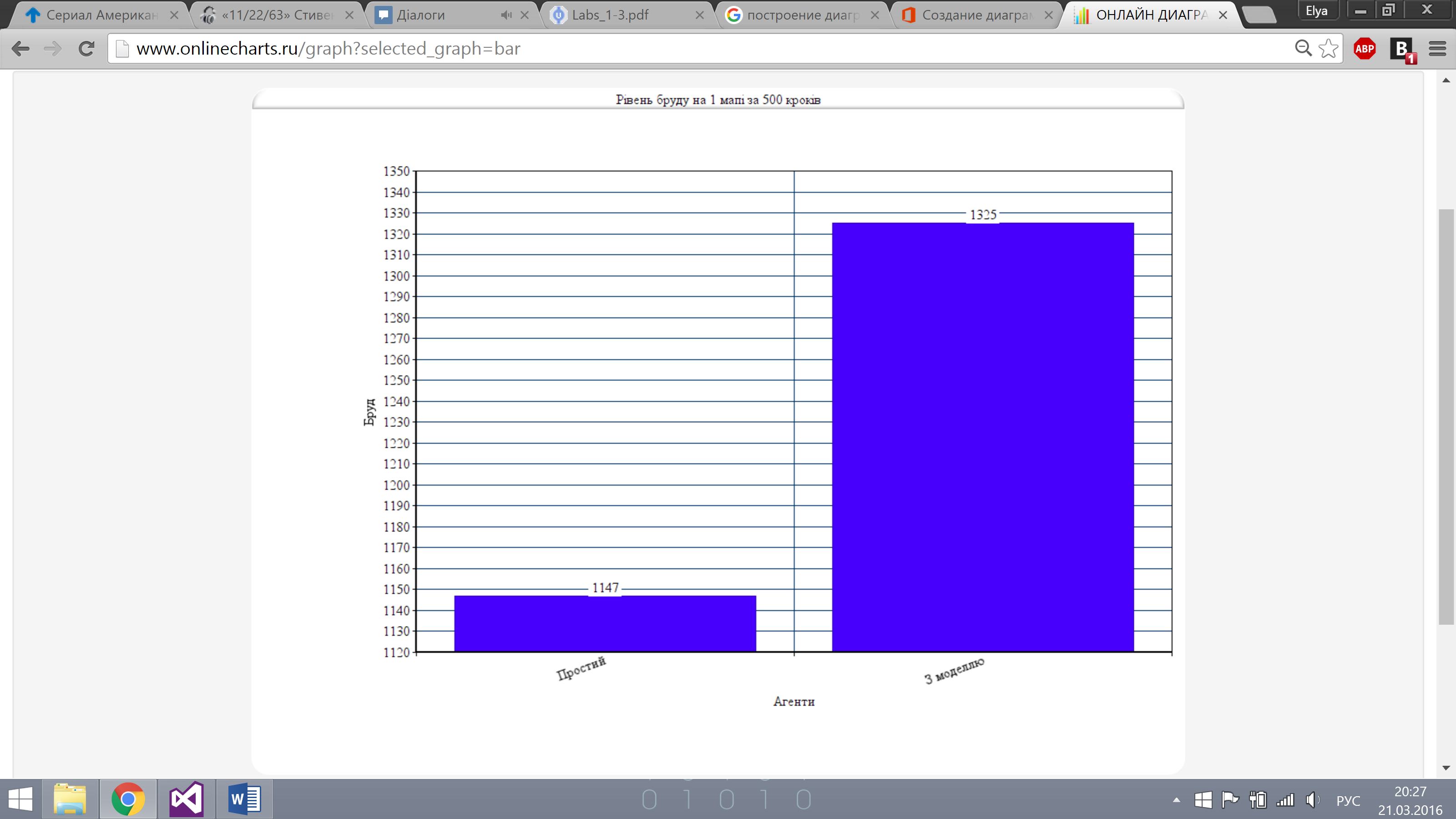


Рис.12

Представимо дані у вигляді таблиці.

Таблиця 1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Мапа | Показник | Простий | | З моделлю | |
|  |  | 500 | 1000 | 500 | 1000 |
| 1 | Бруд | 150071 | 618267 | 102095 | 196254 |
| Енергія | 643 | 1370 | 690 | 1452 |
| 2 | Бруд | 184288 | 710392 | 127465 | 253868 |
| Енергія | 693 | 1426 | 742 | 1559 |
| 3 | Бруд | 160547 | 1011850 | 97534 | 242049 |
| Енергія | 589 | 1147 | 649 | 1325 |

## Аналіз отриманих результатів

Отже, як можна бачити з графіків та таблиці, у середньому кількість енергії, яку витрачає агент, що працює на моделі, більша на 8%. На коротких дистанціях (в експериментах на 500 кроках), агент на моделі прибирає на 30-50% більше, аніж звичайний. Але надалі (500 – 1000 кроки) розрив збільшується до 300-500%.

Також агент, що працює на моделі, краще справляється на мапі із складною топологією, що дозволяє йому не витрачати час та енергію на врізання у перешкоди.

## Висхідні коди

Простий агент. Файл SimpleReflexAgent.h

#pragma once

class Environment;

class SimpleReflexAgent : public Agent {

private:

Environment::Location \_location;

Agent::ActionType way;

int n;

public:

SimpleReflexAgent(): Agent() {

n = 5;

way = ActionType(rand() % 4);

}

ActionType Think() override

{

if (dirty\_) {

return actSUCK;

}

if (!bump\_) {

if (n == 2 + rand() % 6)

way = ActionType(rand() % 4);

return way;

}

else {

way = ActionType(rand() % 4);

}

}

void SimpleReflexAgent::Perceive(const Environment &env) override

{

Agent::Perceive(env);

\_location = env.GetAgentLocation();

}

};

Агент, що працює на моделі. Файл AgentBasedOnModel.h

#pragma once

#include <vector>

#include <algorithm>

class Agent;

class AgentBasedOnModel : public Agent {

private:

Agent::ActionType way;

vector<vector<int>> map;

vector<int> dirt;

vector<int> dirt\_ind;

int steps = 0;

int posx;

int posy;

int n = 5;

int x[4] = { -1, 1, 0, 0 };

int y[4] = { 0, 0, -1, 1 };

int\* state;

public:

AgentBasedOnModel() : Agent() {

state = new int[4]{ 0 };

way = ActionType(rand() % 4);

dirt.resize(4);

map.resize(Environment::MAZE\_SIZE);

/\*for (int i = 0; i < map.size(); i++)

{

vector <int> row;

for (int j = 0; j < Environment::MAZE\_SIZE; j++)

{

row.push\_back(2);

}

map.push\_back(row);

}\*/

for (int i = 0; i < map.size(); i++)

map[i].resize(Environment::MAZE\_SIZE);

/\*for (int i = 0; i < map.size(); i++)

for (int j = 0; j < map[i].size(); j++)

map[i][j] = 2;\*/

}

~AgentBasedOnModel() {

delete[] state;

}

ActionType Think() override

{

if (dirty\_)

return actSUCK;

AgentBasedOnModel::BuildMap();

if (steps < 200) {

steps++;

/\*for (int i = 0; i < map.size(); i++) {

for (int j = 0; j < map[i].size(); j++)

cout << map[i][j];

cout << endl;

}\*/

if (!bump\_) {

if (n == 2 + rand() % 6)

way = ActionType(rand() % 4);

return way;

}

else {

way = ActionType(rand() % 4);

}

}

else {

dirt.clear();

dirt\_ind.clear();

//string act[] = {"actLeft"};

for (int i = 0; i < 4; i++) {

//cout << posx + x[i] << " " << posy + y[i];

//cout << map[posx + x[i]][posy + y[i]] << " state: " << state[i] << endl;

if (map[posx + x[i]][posy + y[i]] != 2) {

dirt.push\_back(state[i]);

dirt\_ind.push\_back(i);

}

}

//auto maxim = std::max(dirt.begin(), dirt.end(), [](const std::pair<int, int> &left, const std::pair<int, int> &right) {

// return left.second < right.second;

//});

if (\*max\_element(std::begin(dirt), std::end(dirt)) == 0) {

int ind = rand() % dirt\_ind.size();

return static\_cast<ActionType>(ActionType(dirt\_ind[ind]));

}

for (int i = 0; i < 4; i++) {

if (state[i] == \*max\_element(std::begin(dirt), std::end(dirt))) {

return static\_cast<ActionType>(ActionType(i));

}

}

/\*cout << "MAX: " << \*maxim << endl;

cout << "----------------------------------" << endl;\*/

/\*float maxim = max(max(state[0], state[1]), max(state[2], state[3]));

for (int i = 0; i < 4; i++) {

if (map[posx + x[i]][posy + y[i]] == 2) {

continue;

}

if (state[i] == maxim) {

return static\_cast<ActionType>(ActionType(i));

}

}\*/

//return static\_cast<ActionType>(5);

//way = ActionType(rand() % 4);

//return static\_cast<ActionType>(way);

}

}

void AgentBasedOnModel::Perceive(const Environment &env) override

{

Agent::Perceive(env);

posx = env.GetAgentX();

posy = env.GetAgentY();

state[0] = env.DirtAmount(env.GetAgentX() - 1, env.GetAgentY());

state[1] = env.DirtAmount(env.GetAgentX() + 1, env.GetAgentY());

state[2] = env.DirtAmount(env.GetAgentX(), env.GetAgentY() - 1);

state[3] = env.DirtAmount(env.GetAgentX(), env.GetAgentY() + 1);

}

void AgentBasedOnModel::BuildMap() {

map[posx][posy] = 1;

for (int i = 0; i < 4; i++) {

map[posx + x[i]][posy + y[i]] = state[i] == -1 ? 2 : 1;

}

/\*if (state[0] == -1) {

map[posx - 1][posy] = 2;

}

else {

map[posx - 1][posy] = 1;

}

if (state[1] == -1) {

map[posx + 1][posy] = 2;

}

else {

map[posx + 1][posy] = 1;

}

if (state[2] == -1) {

map[posx][posy - 1] = 2;

}

else {

map[posx][posy - 1] = 1;

}

if (state[3] == -1) {

map[posx][posy + 1] = 2;

}

else {

map[posx][posy + 1] = 1;

}\*/

}

};